Petr Krysl

Thermal and Stress Analysis with the Finite Element Method

Accompanied by the MATLAB® toolbox FAESOR

December 13, 2010

Pressure Cooker Press
San Diego

Contents

1

Model of a Taut Wire

This chapter will formulate a relatively simple model for the so-called initial boundary value problem that describes the deflection or vibrations of a taut string. In the next chapter, we will seek approximate solutions to this model with the Galerkin method.

1.1 Deriving the PDE model

Figure 1.1 illustrates an idealization of a taut wire. The wire is under prestress by the force P, assumed to be uniform along the length of the wire. The left-hand end is immovably fixed, while the right-hand end is held in a fixture which can slide perpendicularly to the axis of the wire (occasionally referred to as a "roller"). A transverse force F_L is applied at the movable end. In addition, there may be some distributed force q (in physical units of force per unit length) acting along the length (for instance gravity). The transverse displacement is a function of both the axial coordinate x and the time t, $w = w(x,t)$. The transverse displacement is assumed to be very small compared to the length of the wire. The deformation in Figure 1.1 is highly magnified in order to be apparent.

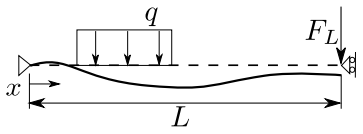


Fig. 1.1. Schematic of taut wire

1.2 Balance equation

We take a segment of length Δx of the wire (see Fig. 1.2). The forces acting on the segment are the prestressing forces in either cross-section and the resultant of the distributed load. By assumption the deflection is very small compared to the span, $w \ll L$, and we also assume that the slope of the wire is very small, $w' = \partial w/\partial x \ll 1$. These geometrical features are introduced into the balance of all the forces. In the horizontal direction we have just the two prestressing forces in opposite directions and hence they cancel. In the vertical direction we add up the components of the prestressing forces in the vertical direction with the transverse load, where we take the Taylor-series approximation for the slope at $x + \Delta x$

$$w'(x + \Delta x) \approx w'(x) + w''(x)\Delta x \ ,$$

and

$$w''(x) = \frac{\partial^2 w(x)}{\partial x^2} \ ,$$

and we equate the resultant of the vertical forces to the inertial force (Newton's law). This leads to a ***balance equation*** for the taut wire

$$Pw'' + q = \mu\ddot{w} \ , \tag{1.1}$$

where $\ddot{w} = \dfrac{\partial^2 w}{\partial t^2}$ is the acceleration.

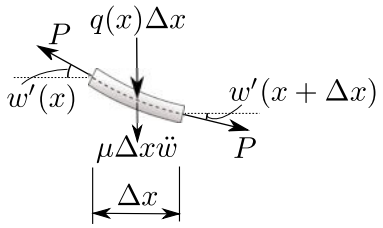


Fig. 1.2. The forces acting on a segment of the taut wire

1.3 Boundary conditions

The function w that describes the transverse deflection takes two arguments, x, and t. It is defined on a rectangular domain shown in Fig. 1.3: $0 \le x \le L$, and $0 \le t \le \bar{t}$. The deflection function needs to be determined to satisfy the balance equation (1.1). However, derivatives with the respect to the variables x and t needs to be "integrated" in order to arrive at a solution, and that implies the presence of integration "constants". In order to determine the solution uniquely we need to resolve the integrations, and for that we need additional equations. Indeed, there *are* other things we would require a solution to satisfy, namely the conditions at the ***boundaries*** of the domain rectangle.

How many pieces of information do we need to know? A reasonable answer is, 'Enough to make the solution unique.' Using the definitions

$$v = \frac{\partial w}{\partial t} \ , \qquad \theta = \frac{\partial w}{\partial x} \ ,$$

we may rewrite the balance equation that involves the second derivatives of the function w as a system of first order partial differential equations

$$\begin{aligned} \frac{\partial \theta}{\partial t} &= \frac{\partial v}{\partial x} \\ P\frac{\partial \theta}{\partial x} + q - \mu\frac{\partial v}{\partial t} &= 0 \end{aligned}$$

For each derivative $\dfrac{\partial v}{\partial x}$, $\dfrac{\partial \theta}{\partial x}$, one boundary condition (integration constant) will be needed. Similarly, for each of the time derivatives $\dfrac{\partial v}{\partial t}$, and $\dfrac{\partial \theta}{\partial t}$ one boundary condition along the time axis will be required.

1.4 Boundary conditions (in space)

The conditions on w along the edges of the domain rectangle parallel to the time axis are known (for historical reasons) as *the* ***boundary conditions***. (Perhaps also because they are applied along the *physical boundaries* of the structure.)

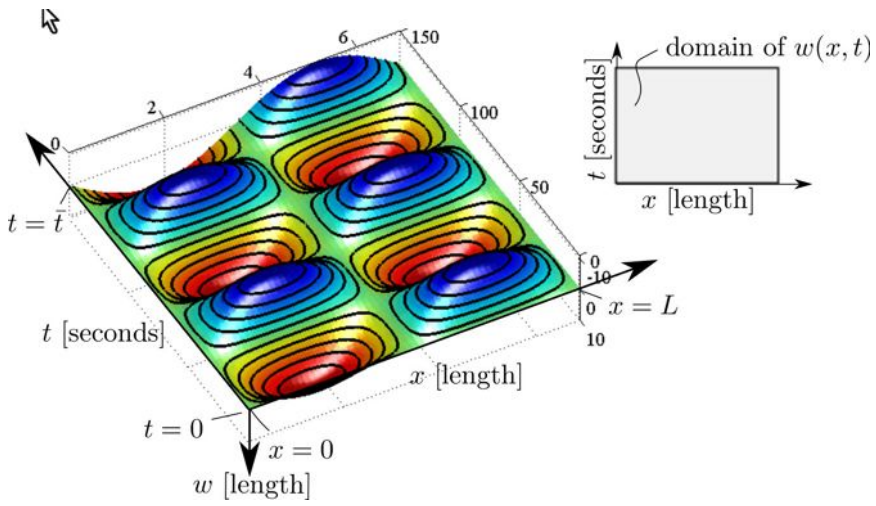


Fig. 1.3. The domain of the deflection function w. The function w that represents the response of a simply supported wire going through slightly more than two cycles of vibration at the second natural frequency is shown as a surface raised above the domain and level curves.

It needs to be realized that the domain of the wire, that is the interval $0 \leq x \leq L$, has only one boundary, namely the two endpoints, $x = 0$ and $x = L$. Since these two points are disjoint, the boundary of the interval consists of two disjoint sets. As discussed in more detail in Chapter 5, we are really prescribing a single boundary condition. Since it happens to be applied at two disjoint points, we loosely use the plural "boundary conditions".

In this example, at the left-hand end of the wire we are prescribing in general nonzero displacement,

$$w(0,t) = \bar{w}_0(t) \; . \tag{1.2}$$

As we shall find out, there is a good reason why this kind of condition is commonly called the ***essential boundary condition***.

At the other end the boundary condition is of a different nature. It is also a bit more interesting, as we have to derive it. Again, we take a short section of the wire of length Δx (see Fig. 1.4). This time there are terms that are multiplied by Δx, but there are also others which are not. Only the latter survive when we make Δx go to zero

$$-P\frac{\partial w}{\partial x}(L,t) + F_L(t) = 0 \; . \tag{1.3}$$

This boundary condition is simply the balance of forces at the end of the wire. Boundary conditions of this kind are called ***natural boundary conditions***.

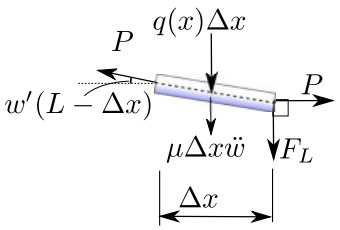


Fig. 1.4. The forces acting on the right-hand end of the taut wire

Exercise 1.

Derive the force boundary condition at $x = 0$.

Solution: Take an infinitesimally short piece of wire starting at the left side end. Express the balance of forces (the force in the wire P, and the force F_0 applied at the left-hand side of the wire as a load or reaction). From the geometry of the wire we have

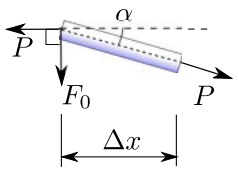


Fig. 1.5. The forces acting on the left-hand end of the taut wire

$$\alpha = \arctan w'(0)\,,$$

and assuming the deflection w is very small compared to the span of the wire, and the slope w' being very small compared to one, we have

$$\cos\alpha \approx 1\,, \sin\alpha \approx \alpha$$

In the vertical direction the sum of forces yields

$$F_0 + P\sin\alpha + q\Delta x = \mu\ddot{w}\Delta x$$

which gives with the geometrical simplifications

$$F_0 + P\alpha = F_0 + Pw'(0) + q\Delta x = \mu\ddot{w}\Delta x$$

Since the piece of the wire is infinitesimally short, $\Delta x \to 0$, we can omit the transverse load and the inertial force. This is then the boundary condition to control the slope at the left-hand side end

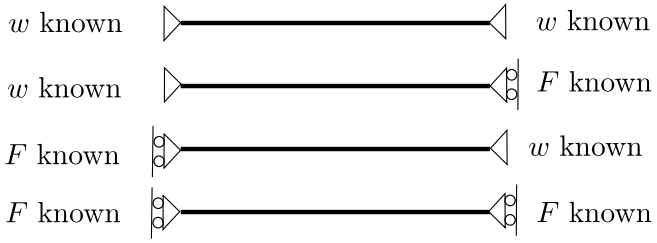
$$w'(0) = -F_0/P$$

in dependence on the prestressing force and the given force F_0.

Exercise 2.

List possible combinations of force and displacement boundary conditions for prestressed cable.

Solution: At either end the transverse motion of the cable may be eliminated (known to be zero) or prescribed at given nonzero value –pin support. Alternatively, at either end the transverse motion of the cable may be unknown –roller support. When an end of the cable is supported by a pin, the associated reaction will generally be unknown. On the other hand, at the end supported by the roller we may apply a known force, zero or nonzero. Hence the possible combinations are as shown here:



1.5 Initial conditions (boundary conditions in time)

Along the edges of the domain rectangle that are parallel to the space axis we also apply two pieces of information. However, as we are all aware, the time direction is special. Therefore, it will probably come to us naturally to expect to know something about the deflection at *one point* in time, typically at $t = 0$. Because this is the initial point along the time axis, this condition is known as the ***initial condition***. We need *two* equations, one for each variable (i.e. for each derivative), in order to compensate for prescribing the condition at one point only:

$$w(x,0) = \bar{W}(x), \quad \frac{\partial w}{\partial t}(x,0) = \bar{V}(x) \ , \tag{1.4}$$

where $\bar{W}(x)$ (the initial deflection) and $\bar{V}(x)$ (the initial velocity) are known functions.

1.6 Initial Boundary Value Problem

The balance equation (1.1), the boundary conditions (1.2) and (1.3), and the initial conditions (1.4) are all we need to fully define the model. It is an ***initial boundary value problem***, and as such it is quite typical of the models with which structural engineers deal in practice. In what follows, we shall find out how to formulate an algorithm, the so-called Galerkin finite element method, which will find an approximate solution to this problem. In the following exercises we will also solve some simple static problems analytically, in order to familiarize ourselves with some aspects of the solutions.

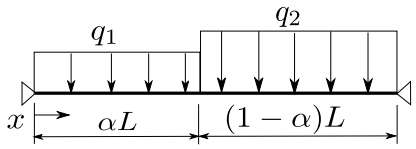
1.7 Examples

Here are a few analytical solutions for selected statically loaded prestressed wire configurations.

Note on the solutions to the solved exercises: we describe here solutions developed with Matlab, but using Matlab is not required. All the analytical solutions below can be obtained by hand, or perhaps with another computer algebra system (Mathcad, Maple, or Mathematica).

Exercise 3.

Solve analytically for the static deflection of the shown prestressed pinned- pinned wire with piecewise uniform transverse load.



Solution: Solving for the static deflection amounts to finding a solution of the following boundary value problem:

$$\begin{aligned} Pw'' + q_1 &= 0 \quad 0 \le x \le \alpha L \\ Pw'' + q_2 &= 0 \quad \alpha L \le x \le L \\ w(0) &= 0 \\ w(L) &= 0 \end{aligned}$$

Note that the balance equation needs to be written for each interval separately since the transverse load is discontinuous. A consequence of the discontinuous transverse load is that the curvature of the cable (the second derivative of the deflection) is also discontinuous at $x = \alpha L$. The first derivative of the deflection however must be continuous at $x = \alpha L$, otherwise the second derivative would be infinite at that point. (Consider that the second derivative corresponds to the curvature of the

cable; also, the curvature is 1/(radius of the osculating circle); the radius of a sharp corner is zero and therefore, the curvature at the sharp corner is infinite.) Therefore, we can supplement the above BVP with two additional conditions: the deflection and the slope of the deflection curve must be continuous at $x = \alpha L$.

$$\begin{aligned} Pw'' + q_1 = 0 \quad & 0 \le x \le \alpha L \\ Pw'' + q_2 = 0 \quad & \alpha L \le x \le L \\ w(0) = 0 & \\ w(L) = 0 & \\ w((\alpha L)_-) = w((\alpha L)_+) & \\ w'((\alpha L)_-) = w'((\alpha L)_+) & \end{aligned}$$

where $x = (\alpha L)_-$ and $x = (\alpha L)_+$ mean immediately to the left and immediately to the right of $x = \alpha L$. Because in each interval the transverse load is constant, we can deduce that in each interval the deflection curve is going to be quadratic in x. Therefore we will write

$$w = A_1 + B_1 x + q_1 C_1 x^2 \,, \quad 0 \le x \le \alpha L$$

and

$$w = A_2 + B_2 x + q_2 C_2 x^2 \,, \quad \alpha L \le x \le L$$

Substituting into the balance equation we obtain

$$Pw'' + q_1 = 2Pq_1 C_1 + q_1 = 0 \Rightarrow C_1 = -\frac{1}{2P}$$

and similarly $C_2 = -\frac{1}{2P}$. Therefore, there are four coefficients to be determined to complete the solution, A_1, B_1, A_2, B_2. Fortunately, we have the boundary conditions and the conditions of continuity where the load changes. Four conditions for four unknown coefficients. Here is a short program in Matlab to solve for the coefficients (it would be of course possible to solve the resulting system by hand):

```
% Solve for the deflection of a prestressed cable
% with piecewise uniform distributed load
syms A1  B1 A2  B2  alpha  L P q1 q2 x real
w1 = @(x)(A1+B1*x-q1*x^2/(2*P));
dw1 = @(x)(B1-q1*x/P);
w2 = @(x)(A2+B2*x-q2*x^2/(2*P));
dw2 = @(x)(B2-q2*x/P);

Solution =solve([char(w1(0)) '=0'],...
    [char(w2(L)) '=0'],...
    [char(w1(alpha*L))  '=' char(w2(alpha*L))],...
    [char(dw1(alpha*L))  '=' char(dw2(alpha*L))],...
    'A1','B1','A2','B2');
```

The values of the constants in terms of the variables in the problem are:

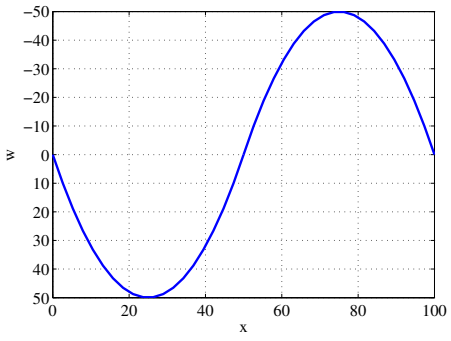
```
pretty(Solution.A1)
  0
pretty(simplify(Solution.B1))
                                          2           2
  L (q2 + 2 alpha q1 - 2 alpha q2 - alpha  q1 + alpha  q2)
  ------------------------------------------------------
                            2 P
pretty(simplify(Solution.A2))
```

```
   2      2
  L  alpha  (q1 - q2)
  ------------------
          2 P
pretty(simplify(Solution.B2))
                2           2
  L (q2 - alpha  q1 + alpha  q2)
  ------------------------------
                2 P
```

Now we can pick some particular numbers for the parameters and plot the deflection curve. Note that the two intervals are of the same length and the two distributed loads are equal in magnitude but of opposite sign.

```
alpha = 0.5; q1 =  4; q2 = -4; L = 100; P = 25;
A1 =Solution.A1; B1 =Solution.B1;
A2 =Solution.A2; B2 =Solution.B2;
x= linspace(0,alpha*L, 20);
plot (x,eval((A1+B1*x-q1*x.^2/(2*P))));
hold on
x= linspace(alpha*L,L, 20);
plot (x,eval((A2+B2*x-q2*x.^2/(2*P))));
set(gca,'ytick',-50:10:50)
xlabel ('x')
ylabel ('w')
grid on
left_handed_axes% positive deflection downwards
```

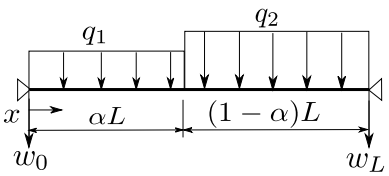
Note that the function `left_handed_axes` [1] orients the axes of the figure in order to display the positive deflection downwards.



Exercise 4.

Solve analytically for the static deflection of the shown prestressed pinned- pinned cable with piecewise uniform transverse load and support-settlement loads.

[1] Folder: `FAESOR/util`



Solution: Solving for the static deflection amounts to finding a solution of the following boundary value problem:

$$\begin{aligned} Pw'' + q_1 = 0 \quad & 0 \le x \le \alpha L \\ Pw'' + q_2 = 0 \quad & \alpha L \le x \le L \\ w(0) = w_0 & \\ w(L) = w_L & \end{aligned}$$

The balance equation needs to be written for each interval separately since the transverse load is discontinuous. Again, the BVP needs to be supplemented with two additional conditions: the deflection and the slope of the deflection curve must be continuous at $x = \alpha L$.

$$\begin{aligned} Pw'' + q_1 = 0 \quad & 0 \le x \le \alpha L \\ Pw'' + q_2 = 0 \quad & \alpha L \le x \le L \\ w(0) = w_0 & \\ w(L) = w_L & \\ w((\alpha L)_-) = w((\alpha L)_+) & \\ w'((\alpha L)_-) = w'((\alpha L)_+) & \end{aligned}$$

where $x = (\alpha L)_-$ and $x = (\alpha L)_+$ mean immediately to the left and immediately to the right of $x = \alpha L$. Because in each interval the transverse load is constant, we can deduce that in each interval the deflection curve is going to be quadratic in x. Therefore we will write

$$w = A_1 + B_1 x + q_1 C_1 x^2 \,, \quad 0 \le x \le \alpha L$$

and

$$w = A_2 + B_2 x + q_2 C_2 x^2 \,, \quad \alpha L \le x \le L$$

Substituting into the balance equation we obtain

$$Pw'' + q_1 = 2Pq_1 C_1 + q_1 = 0 \Rightarrow C_1 = -\frac{1}{2P}$$

and similarly $C_2 = -\frac{1}{2P}$. Therefore, there are four coefficients to be determined to complete the solution, A_1, B_1, A_2, B_2. Fortunately, we have the boundary conditions and the conditions of continuity where the load changes. Four conditions for four unknown coefficients.

For expediency here is a short program in Matlab to solve for the coefficients:

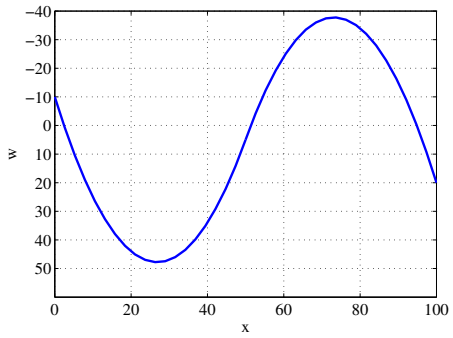
```
% Solve for the deflection of a prestressed cable
% with piecewise uniform load + support settlement
syms A1  B1 A2  B2  alpha  L P q1 q2 x w0 wL real
w1 = @(x)(A1+B1*x-q1*x^2/(2*P));
dw1 = @(x)(B1-q1*x/P);
w2 = @(x)(A2+B2*x-q2*x^2/(2*P));
dw2 = @(x)(B2-q2*x/P);

Solution =solve([char(w1(0)) '=' char(w0)],...
    [char(w2(L)) '='  char(wL)],...
    [char(w1(alpha*L))  '=' char(w2(alpha*L))],...
    [char(dw1(alpha*L))  '=' char(dw2(alpha*L))],...
    'A1','B1','A2','B2');
```

The values of the constants in terms of the variables in the problem are:

```
>> pretty(simplify(Solution.A1))

  w0
>> pretty(simplify(Solution.B1))

                                        2           2
L (q2 + 2 alpha q1 - 2 alpha q2 - alpha  q1 + alpha  q2)   w0 - wL
---------------------------------------------------------- - -------
                          2 P                                  L
>> pretty(simplify(Solution.A2))

        2      2
       L  alpha  (q1 - q2)
  w0 + --------------------
               2 P
>> pretty(simplify(Solution.B2))

               2           2
  L (q2 - alpha  q1 + alpha  q2)   w0 - wL
  ------------------------------ - -------
               2 P                    L
```

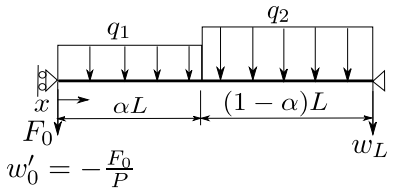
Now we can pick some particular numbers for the parameters and plot the deflection curve. Note that the two intervals are of the same length and the two distribute loads are equal in magnitude but of opposite sign.

```
alpha = 0.5; q1 =  4; q2 = -4; L = 100; P = 25;
w0 =-10; wL = 20;
A1 =Solution.A1; B1 =Solution.B1;
A2 =Solution.A2; B2 =Solution.B2;
x= linspace(0,alpha*L, 20);
plot (x,eval((A1+B1*x-q1*x.^2/(2*P))));
hold on
x= linspace(alpha*L,L, 20);
plot (x,eval((A2+B2*x-q2*x.^2/(2*P))));
set(gca,'ytick',-50:10:50)
xlabel ('x')
ylabel ('w')
grid on
left_handed_axes% positive deflection downwards
```



Exercise 5.

Solve analytically for the static deflection of the shown prestressed roller- pinned cable with piecewise uniform transverse load, concentrated force F_0 at the left-hand side end, and support-settlement loads.



Solution: Solving for the static deflection amounts to finding a solution of the following boundary value problem:

$$\begin{aligned} Pw'' + q_1 = 0 \quad & 0 \le x \le \alpha L \\ Pw'' + q_2 = 0 \quad & \alpha L \le x \le L \\ w'(0) = -\tfrac{F_0}{P} & \\ w(L) = w_L & \end{aligned}$$

The balance equation needs to be written for each interval separately since the transverse load is discontinuous. Again, the BVP needs to be supplemented with two additional conditions: the deflection and the slope of the deflection curve must be continuous at $x = \alpha L$.

$$\begin{aligned} Pw'' + q_1 = 0 \quad & 0 \le x \le \alpha L \\ Pw'' + q_2 = 0 \quad & \alpha L \le x \le L \\ w'(0) = -\tfrac{F_0}{P} & \\ w(L) = w_L & \\ w((\alpha L)_-) = w((\alpha L)_+) & \\ w'((\alpha L)_-) = w'((\alpha L)_+) & \end{aligned}$$

where $x = (\alpha L)_-$ and $x = (\alpha L)_+$ mean immediately to the left and immediately to the right of $x = \alpha L$. Because in each interval the transverse load is constant, we can deduce that in each interval the deflection curve is going to be quadratic in x. Therefore we will write

$$w = A_1 + B_1 x + q_1 C_1 x^2\,, \quad 0 \le x \le \alpha L$$

and

$$w = A_2 + B_2 x + q_2 C_2 x^2\,, \quad \alpha L \le x \le L$$

Substituting into the balance equation we obtain

$$Pw'' + q_1 = 2Pq_1 C_1 + q_1 = 0 \Rightarrow C_1 = -\frac{1}{2P}$$

and similarly $C_2 = -\frac{1}{2P}$. Therefore, there are four coefficients to be determined to complete the solution, A_1, B_1, A_2, B_2. Fortunately, we have the boundary conditions and the conditions of continuity where the load changes. Four conditions for four unknown coefficients. As a for way will be well served by a short program in Matlab. Here is the symbolic solution for the coefficients from the boundary conditions and continuity conditions. Note the change with the respect to the previous codes due to the roller boundary condition.

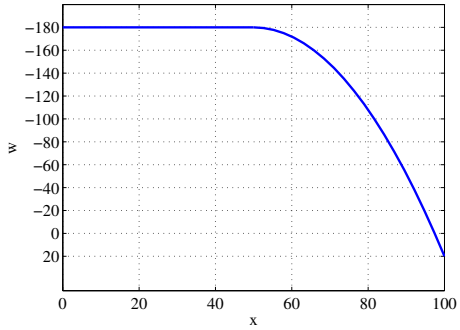
```
syms A1  B1 A2  B2  alpha  L P q1 q2 x F0 wL real
w1 = @(x)(A1+B1*x-q1*x^2/(2*P));
dw1 = @(x)(B1-q1*x/P);
w2 = @(x)(A2+B2*x-q2*x^2/(2*P));
dw2 = @(x)(B2-q2*x/P);

Solution =solve([char(dw1(0)) '=' char(-F0/P)],...
    [char(w2(L)) '='  char(wL)],...
    [char(w1(alpha*L))  '=' char(w2(alpha*L))],...
    [char(dw1(alpha*L))  '=' char(dw2(alpha*L))],...
    'A1','B1','A2','B2');
```

The values of the constants in terms of the variables in the problem are:

```
>> pretty(simplify(Solution.A1))

   2          2      2      2      2      2          2
2 L  alpha - L  alpha      L  alpha  - 2 L  alpha + L      F0 L
-------------------- q1 + -------------------------- q2 + ---- + wL
         2 P                           2 P                 P

>> pretty(simplify(Solution.B1))

    F0
  - --
    P
>> pretty(simplify(Solution.A2))

       L (2 F0 + L q2 + 2 L alpha q1 - 2 L alpha q2)
  wL + ---------------------------------------------
                           2 P
>> pretty(simplify(Solution.B2))

    F0 + L alpha q1 - L alpha q2
  - ----------------------------
                 P
```

Now we can pick some particular numbers for the parameters and plot the deflection curve. Note that the two intervals are of the same length. Note also that the distributed load in the left-hand side interval is zero, and hence the deflection curve there has a zero curvature (straight line).

```
alpha = 0.5; q1 =  0; q2 = -4; L = 100; P = 25;
F0 =0; wL = 20;
A1 =Solution.A1; B1 =Solution.B1;
A2 =Solution.A2; B2 =Solution.B2;
x= linspace(0,alpha*L, 20);
plot (x,eval((A1+B1*x-q1*x.^2/(2*P))));
hold on
x= linspace(alpha*L,L, 20);
plot (x,eval((A2+B2*x-q2*x.^2/(2*P))));
set(gca,'ytick',-180:20:20)
xlabel ('x')
ylabel ('w')
grid on
left_handed_axes% positive deflection downwards
```


−180
−160
−140
−120
−100
−80
−60
−40
−20
0
20
w
0
20
40
60
80
100
x

2

The Method of Galerkin

We continue working with the prestressed cable model. In this chapter we will begin to come to grips with the possibility of satisfying the equations of the model not exactly but only approximately. There's going to be an error in the balance equation (which we shall call a residual; another appropriate label might be imbalance). Similarly, the natural (force) boundary condition may not be satisfied exactly, and will also produce a residual.

In this book, the approximate solutions are obtained with the Galerkin method. Boris Grigoryevich Galerkin became a teacher of structural mechanics in St. Petersburg Polytechnical Institute in 1908. Among his contemporaries, also active in St. Petersburg, were I. G. Bubnov, A. N. Krylov, and S. P. Timoshenko, all well-known names in various areas of mechanics. In 1915 Galerkin published an article, in which he put forward an idea of an approximate method to solve differential boundary value problems (he was working on plate and shell models at that time). Around that time Bubnov developed similar variational approach, hence this method is also known as the Bubnov-Galerkin method.

It is an extremely general and hence very valuable approach. In particular, it is applicable in various areas of nonlinear computational mechanics, for instance for inelastic deformation of materials, large deflection and large strain deformation, and so on. In this book we will restrict ourselves to linear models however.

2.1 Residual of the balance equation

For the moment we will reduce the complexity of the prestressed wire IBVP model by considering only statics. The omission of dynamics will not invalidate the following developments, and including dynamics will be shown to be straightforward. Thus in the following we will consider the statics BVP, consisting of the balance equation

$$Pw''(x) + q(x) = 0 \ , \tag{2.1}$$

and the boundary conditions

$$w(0) = \bar{w}_0 \ . \tag{2.2}$$

and

$$-Pw'(L) + F_L = 0 \ . \tag{2.3}$$

The balance equation (2.1) may be written in the residual form as

$$Pw''(x) + q(x) = r_B(x) \ . \tag{2.4}$$

The residual r_B is identically zero if w is the exact solution. For an approximate solution, the residual r_B varies from point to point, and is in general nonzero.

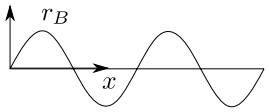


Fig. 2.1. Residual that integrates to zero, but is not *identically* zero

Checking that the balance residual is identically zero at each point x does not provide us with anything we can use to talk about approximate solutions: the residual is either zero or it isn't. So how do we measure whether the approximate solution, for which the residual is not zero, is in some sense satisfactory (or not)?

2.2 Integral test of the residual

One possible choice of a quality measure is to integrate the residual over the domain (length of the wire). We could think of the integral

$$\int_0^L r_B(x)\,\mathrm{d}x \tag{2.5}$$

as a test: if the residual is identically zero, this integral will also come out zero. However, Eq. (2.5) may be zero even when the residual is not identically zero. In other words, if we wanted to prove that the residual corresponded to an exact solution, this would be an incomplete and flawed test. Consider Fig. 2.1: the integral (2.5) is zero, but the residual itself may be very large (for instance, when $r_B = A\sin(2\pi nx/L)$, with $n = 1, 2, ...$).

2.3 Test function

A remedy that addresses this blindness of (2.5) to the shape of the residual may be to use a "window" (test) function $\eta(x)$

$$\int_0^L \eta(x) r_B(x)\,\mathrm{d}x\ . \tag{2.6}$$

Note that $\eta(x)$ is an arbitrary function. In particular, it could be a function of the shape shown in Fig. 2.2, which is certainly going to give a nonzero value for (2.6) (the hatched area at the bottom). Therefore, it correctly indicates that the residual does not correspond to the exact solution. Equation (2.6) is known as the ***weighted residual statement***, because each test function η applies a variable weight to the residual in different parts of the domain. Approximate approaches that start from the weighted residual statement are known as weighted residual methods.

Equation (2.6) is a reliable way of testing the residual, but computationally it seems hardly less difficult than testing the residual at each point of the domain: equation (2.6) needs to be evaluated for an infinite number of functions η in order to make sure there are no bumps in the residual. The job will still take an infinite time.

Let us contemplate a tangible analogy of what we're trying to do in Eq. (2.6). Imagine our job is to hold an inflatable balloon in a box, so that it does not jut out anywhere. Use the fingers of one hand to press down on the balloon, so that the balloon is at the top of the box in the spot where it is being held by the finger. If we put down all five fingers, the situation is as shown on the left in Fig. 2.3. Each of the fingers may be thought of as a single test function η that pushes down the residual in some spots.

Evidently, the balloon bulges out a little bit in between the fingers, and a lot everywhere else. However, we have the option of pressing down on the balloon with the fingers of our other hand, and if we enrol our friends and relatives, and the chance passersby, and distribute the pressing fingers

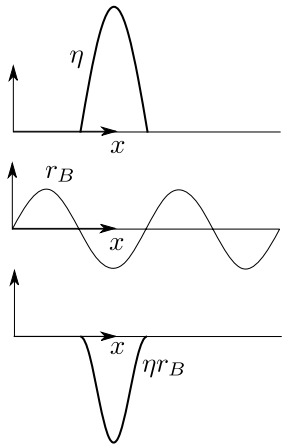


Fig. 2.2. Nonzero residual which is detected in the integral (2.6)

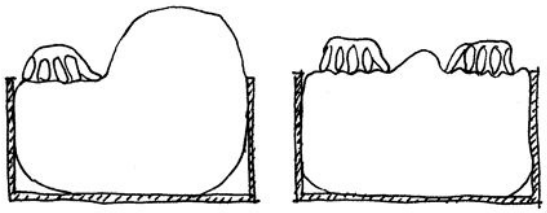


Fig. 2.3. Stuffing a balloon into a box

wisely, we will manage to do a better and better job of stuffing the balloon into the box and holding it so that it does not protrude very much. Indeed, with an infinite number of fingers, we can hold the balloon so that it does not protrude at all. Note that we have to distribute the pressing fingers in some sense ***densely*** and ***uniformly***– no parts of the interval $0 \leq x \leq L$ may be left out, since the residual could stay nonzero there.

In this way, we may begin to see how a trial-and-test approximate method may be formulated. Selecting a finite number of suitable functions η_j (fingers), we may be able to control distribution and magnitude of the residual (but, in general, it will remain nonzero). By applying larger numbers of test functions, we will be able to reduce the error in the residual and get a better solution. Also, for each η_j, $j = 1, ..., N$, we will make the integral (2.6) vanish

$$\int_0^L \eta_j(x) r_B(x)\, \mathrm{d}x = 0 \ , \tag{2.7}$$

which provides us with the means of calculating N coefficients (numbers) from these N equations.

2.4 Trial function

The task of formulating the approximate solution requires describing the shape of the deflection w. This can be done in a variety of ways, but for reasons that we shall give later, a piecewise linear representation is a good choice. Figure 2.4 illustrates this concept by showing how the shape may be defined by the N coefficients w_j. The attentive reader will at this point fidget: the piecewise linear shape of the deflection curve is not going to allow us to express the second order derivatives $w'' = \partial^2 w/\partial x^2$. At the corners, the first derivatives will be discontinuous, and hence the second derivative will be a spike (the so-called Dirac delta function). We can choose either to abandon the piecewise linear shape, and pass a smooth curve through the filled-circle points, or, we could change

the rules of the game by getting rid of the second-order derivatives. As we shall presently see, the latter choice is commonly preferred.

In any case, the Eqs. (2.7) may be used to calculate the values of w_j, $j = 1, ..., N$. The function that describes the shape of the approximate solution (with the N free parameters) is known as the ***trial function***. It describes a possible (candidate, trial) shape of the approximate solution; which becomes *the* solution once the values of the free parameters are known.

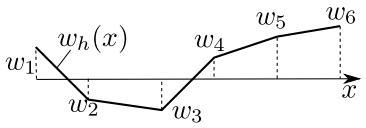


Fig. 2.4. Piecewise linear trial function

2.5 Shifting derivatives

Substituting for the balance residual, we get two terms

$$\int_0^L \eta_j(x) r_B(x)\ \mathrm{d}x = \int_0^L \eta_j(x) P w''(x)\ \mathrm{d}x + \int_0^L \eta_j(x) q(x)\ \mathrm{d}x\ . \tag{2.8}$$

Integration by parts may be applied to the first term on the right-hand side using the identity

$$(\eta_j P w')' = \eta_j{}' P w' + \eta_j P w''$$

and integrating from 0 to L we obtain for the left-hand side

$$\int_0^L (\eta_j P w')'\ \mathrm{d}x = [\eta_j P w']_0^L = \eta_j(L) P w'(L) - \eta_j(0) P w'(0)$$

Therefore we see that we can replace the term with the second derivatives as

$$\int_0^L \eta_j P w''\ \mathrm{d}x = \eta_j(L) P w'(L) - \eta_j(0) P w'(0) - \int_0^L \eta_j{}' P w'\ \mathrm{d}x\ . \tag{2.9}$$

This does the trick: the second derivatives are gone from the trial function w. All that is left are first derivatives, and the piecewise linear functions that we spoke about are now acceptable.

2.6 Essential boundary condition

We also must satisfy the essential boundary condition (2.2). We may realize that at the left-hand end of the wire we can quite simply *design* the trial function to satisfy this condition identically. This is not difficult, and we have therefore the following requirements on the trial function at this point: to satisfy the essential boundary condition we require $w(0) = \bar{w}_0$, and further the trial function $w(x)$ needs to be sufficiently smooth in x for the derivatives in the residual r_B to exist.

2.7 Natural boundary condition

In general it will not be possible to come up with a trial function that would satisfy (2.3) at the right-hand end of the wire ($x = L$) identically. We would have to control the slope of w at $x = L$, and that is awkward at best. Therefore in addition to the residual r_B along the length of the wire we

will need to put up with an imbalance (residual) r_F at $x = L$ where the natural boundary condition is applied

$$r_F = -Pw'(L) + F_L \ . \tag{2.10}$$

We may take a list from the book written for the balance residual: multiply the residual with a test function and integrate. In this case integration means integrate over the boundary at $x = L$, that's where r_F "lives". Since we have taken $\eta_j(x)$ for the balance residual, we may take the same function here, evaluated on the boundary. Therefore the weighted residual for the natural boundary condition may be written as

$$\eta_j(L) r_F = \eta_j(L)(-Pw'(L) + F_L)$$

Unfortunately, we cannot just set

$$\eta_j(L) r_F = \eta_j(L)(-Pw'(L) + F_L) = 0 \tag{2.11}$$

There are two ways in which this could happen: either the parenthesis is identically zero (which as we said above is in general not going to happen), or we choose $\eta_j(L) = 0$. The latter is not helpful however: all it means is that the force F_L does not play any role in the solution, and that is definitely not what we want. So we cannot keep the natural boundary condition in a weighted residual of its own. Over the years, the following clever manipulation was developed to resolve this dilemma: note that $\eta_j(L)Pw'(L)$ appears both in (2.9) and in the natural boundary condition weighted residual equation above. Therefore, we add the two equations (2.8) and (2.11), and introduce (2.9) to give

$$\begin{aligned} &\int_0^L \eta_j(x) r_B(x)\, \mathrm{d}x + \eta_j(L) r_F = \\ &\underline{\eta_j(L)Pw'(L)} - \eta_j(0)Pw'(0) - \int_0^L \eta_j{}' Pw'\, \mathrm{d}x + \int_0^L \eta_j(x) q(x)\, \mathrm{d}x + \underline{\eta_j(L)}(\underline{-Pw'(L)} + F_L) = 0 \ , \end{aligned} \tag{2.12}$$

where the underlined terms cancel, and the expression simplifies to

$$\begin{aligned} &\int_0^L \eta_j r_B\, \mathrm{d}x + \eta_j(L) r_F = \\ &-\eta_j(0)Pw'(0) - \int_0^L \eta_j{}' Pw'\, \mathrm{d}x + \int_0^L \eta_j q\, \mathrm{d}x + \eta_j(L) F_L = 0 \ , \end{aligned} \tag{2.13}$$

So far so good. We have combined the balance residual with the natural boundary condition residual to obtain an expression which has only the first-order derivatives on the test and the trial functions, and which incorporates the given force F_L.

Unfortunately, there is one more snag: At the left-hand side end of the wire (at the pin support) the value of the force $(-Pw'(0))$ is unknown – it is a reaction. We would prefer not to have this force in the weighted residual. We do have the option of requiring that η_j vanish at $x = 0$, and thus eliminate $(-Pw'(0))$. This will burden all the test η_j's with a condition, $\eta_j(x = 0) = 0$, but that is something we can probably afford. At this point we will require all test functions to become zero where the essential boundary conditions are prescribed ($x = 0$). Later we will relax this condition since for some applications it is worthwhile to be able to compute reactions too.

In summary: We have satisfied the essential boundary condition by design of the trial function, and the force boundary condition (2.10) was merged into the balance residual (which is by the way why we call this the "natural" boundary condition: it appears naturally in the model equations). Hence we will try to find the approximate solution w to satisfy the balance equation combined with the natural boundary condition in the residual form

$$\eta_j(L) F_L - \int_0^L \eta_j{}' Pw'\, \mathrm{d}x + \int_0^L \eta_j q\, \mathrm{d}x = 0, \quad j = 1, ..., N \ , \tag{2.14}$$

where we subject the test and trial function to the conditions

$$\begin{aligned} &\eta_j(x=0)=0, \quad \eta_j \in C^0, \quad j=1,...,N\ , \\ &w(x=0)=\bar{w}_0, \quad w \in C^0. \end{aligned} \tag{2.15}$$

We write for the trial function $w \in C^0$ and similarly for the test functions. This literally means that the functions are continuous (C^0 denotes the set of functions that are continuous on the real line); that is a substitute for a more precise mathematical statement, but one that nevertheless ensures that the integrals in (2.14) exist.

2.8 Stiffness matrix and load vector

Let us revisit the choice of the test and trial functions. As advertised in Section 2.4, we have been able to change the requirements on the test and trial function: Their derivatives are now balanced– only the first-order derivatives are needed for either. Therefore, the piecewise linear interpolation function of Fig. 2.4 is now a possibility. However, we can still forge ahead while keeping our options open. In this section we therefore still do not commit to a specific form of the trial and test function.

To describe the trial function, we will resort to a common technique in interpolation and approximation literature which is to write the trial function as a linear combination of ***basis functions***. Therefore, let us assume that the trial function is written as

$$w(x)=\sum_{i=1}^{N} N_i(x) w_i\ , \tag{2.16}$$

where the w_i's are the coefficients of the linear combination (real numbers); these coefficients are also called ***degrees of freedom***. The $N_i(x)$'s are known (suitably chosen) basis functions. Note that the number of terms N in the trial function matches the number of the test functions used. That is because the number of unknowns needs to be matched to the number of equations available.

Substituting into (2.14), we obtain

$$\eta_j(L)F_L - \int_0^L \eta_j{}' P \sum_{i=1}^{N} N_i{}' w_i \,\mathrm{d}x + \int_0^L \eta_j q \,\mathrm{d}x = 0, \quad j=1,...,N\ , \tag{2.17}$$

which may be simplified to

$$\begin{aligned} &\eta_j(L)F_L - \sum_{i=1}^{N} \left(\int_0^L \eta_j{}' P N_i{}' \,\mathrm{d}x \right) w_i + \int_0^L \eta_j q \,\mathrm{d}x = 0, \quad j=1,...,N \\ &\qquad \eta_j(x=0)=0, \quad \eta_j \in C^0, \quad j=1,...,N\ , \quad w(x=0)=\bar{w}_0, \quad w \in C^0.\ . \end{aligned} \tag{2.18}$$

With the definitions

$$K_{ji} = \int_0^L \eta_j{}' P N_i{}' \,\mathrm{d}x\ , \tag{2.19}$$

and

$$L_j = \eta_j(L)F_L + \int_0^L \eta_j q \,\mathrm{d}x \tag{2.20}$$

we may write (2.18) in the form

$$\sum_{i=1}^{N} K_{ji} w_i = L_j, \quad j=1,...,N \tag{2.21}$$

that makes it clear we have converted the original BVP to a linear-algebra problem of a system of coupled linear algebraic equations. The coefficients K_{ji} are usually referred to as the ***stiffness matrix*** elements, and L_j are the elements of the ***load vector***. Note well that the above describes a Galerkin method, but it has nothing to do yet with finite elements.

Exercise 6.

Solve for the approximate deflection of a simply-supported prestressed cable with uniform load using the Galerkin method. Take as the trial function $N_1(x) = \sin(\pi x/L)$, and set the test function to coincide with the trial function, $\eta_1 = N_1$. Compare the midpoint deflection computed analytically and approximately.

Solution: We are solving the boundary value problem:

$$\begin{aligned} Pw'' + q &= 0 \\ w(0) &= 0 \\ w(L) &= 0 \end{aligned}$$

The trial function is

$$w(x) = a_1 N_1(x)$$

Note that the trial function satisfies the essential boundary conditions because

$$N_1(0) = N_1(L) = 0$$

The coefficient a_1 is the only unknown. Therefore one test function is sufficient since we need only one equation.

Equation (2.18) simplifies for no natural boundary conditions to

$$-\int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\, \mathrm{d}x + \int_0^L \eta_j(x) q(x)\, \mathrm{d}x = 0\ .$$

Substituting the test and trial function we have

$$-\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x} a_1\, \mathrm{d}x + \int_0^L N_1(x) q\, \mathrm{d}x = 0\ .$$

and simplifying further

$$-a_1 \int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x}\, \mathrm{d}x + q \int_0^L N_1(x)\, \mathrm{d}x = 0\ .$$

Just five lines of Matlab symbolic algebra do the job:

```
syms   L P q x real
N1=sin(pi*x/L);
K=int(diff(N1)*P*diff(N1),0,L)
F=q*int(N1,0,L)
a1=K\F
```

yielding

```
K =
(P*pi^2)/(2*L)
F =
(2*L*q)/pi
a1 =
(4*L^2*q)/(P*pi^3)
```

The analytical solution is

$$w_{ex}(x) = \frac{qx(L-x)}{2P}$$

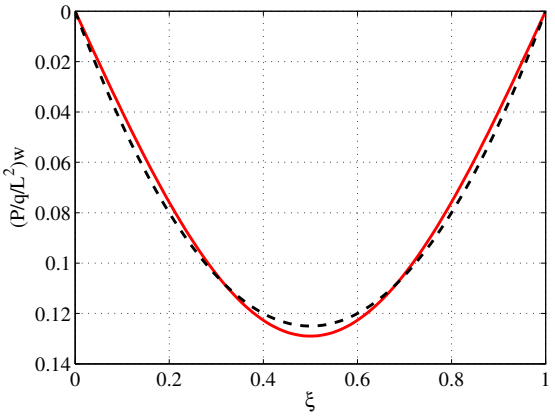
We can compare the midpoint deflection, for instance: analytical

$$w_{ex}(L/2) = \frac{qL^2}{8P} = \frac{0.125qL^2}{P}$$

versus approximate

$$w_{app}(L/2) = \frac{4qL^2}{\pi^3 P} \approx \frac{0.129qL^2}{P}$$

The analytical (dashed black) and approximate (solid red) curves of the deflections scaled by $(P/q/L^2)$ versus the nondimensional coordinate $\xi = x/L$ are shown in the figure below.



Exercise 7.

For the approximate solution computed in the exercise 6 evaluate the balance residual.

Solution: The Galerkin approximate solution to

$$Pw'' + q = 0 \quad w(0) = 0, \quad w(L) = 0$$

was found as

$$w_{app}(x) = \frac{(4L^2 q)}{(P\pi^3)} \sin(\pi x/L)$$

The balance residual is

$$r_B = Pw'' + q$$

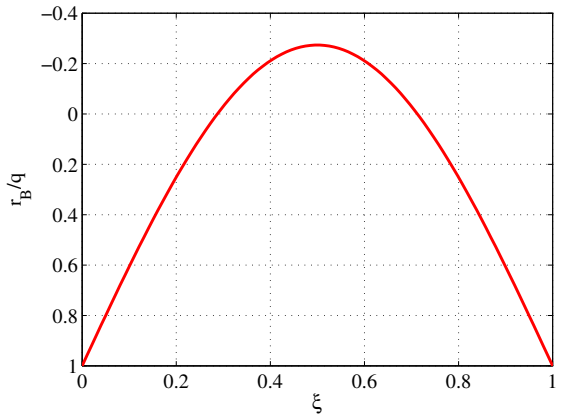
A bit of symbolic algebra

```
syms   L P q x real
N1=sin(pi*x/L);
K=int(diff(N1)*P*diff(N1),0,L);
F=q*int(N1,0,L);
a1=K\F;
w=N1*a1;
rB =P*diff(diff(w))+q
```

yields the balance residual as a function in x

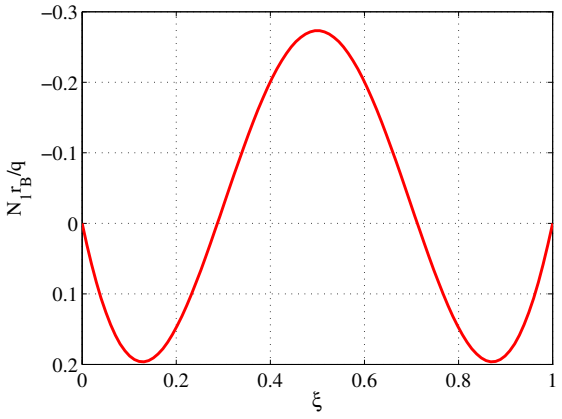
```
rB =
q - (4*q*sin((pi*x)/L))/pi
```

The balance the residual is not identical is identically zero – after all this is an approximate solution. Here is the residual scaled by the transverse load:



Plotting the product $N_1 r_B$, that is the product of the test function and the residual, illustrates that the solution was found by making the residual orthogonal to the test function: when the residual multiplied by the test function is integrated over the length of the domain of the cable we obtain zero, the areas above the horizontal axis canceling those below.

```
>> int(N1*rB,0,L)
ans =
0
```



Exercise 8.

Solve for the approximate deflection of a simply-supported prestressed cable with uniform load using the Galerkin method. Take as the trial functions $N_1(x) = \sin(\pi x/L)$, and $N_2(x) = \sin(3\pi x/L)$ and set the test functions to coincide with the trial functions, $\eta_1 = N_1$, $\eta_2 = N_2$. Compare the midpoint deflection computed analytically and approximately.

Solution: We are solving the boundary value problem:

$$\begin{aligned} Pw'' + q &= 0 \\ w(0) &= 0 \\ w(L) &= 0 \end{aligned}$$

The trial function is

$$w(x) = a_1 N_1(x) + a_2 N_2(x)$$

Note that the trial function satisfies the essential boundary conditions because

$$N_j(0) = N_j(L) = 0 \quad \text{for } j = 1, 2$$

The coefficients a_1, a_2 are the unknowns. Therefore two test functions are needed to generate to equations from which to solve for the unknowns.

Equation (2.14) simplifies for statics (zero accelerations) and no natural boundary conditions to

$$-\int_0^L \frac{\partial \eta_j(x)}{\partial x} P \frac{\partial w(x)}{\partial x}\, \mathrm{d}x + \int_0^L \eta_j(x) q\, \mathrm{d}x = 0 \quad \text{for } j = 1, 2\,.$$

Substituting the test and trial function we have

$$\begin{aligned} &-\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x} a_1\, \mathrm{d}x + -\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_2}{\partial x} a_2\, \mathrm{d}x + \int_0^L N_1(x) q\, \mathrm{d}x = 0\,, \\ &-\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_1}{\partial x} a_1\, \mathrm{d}x + -\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_2}{\partial x} a_2\, \mathrm{d}x + \int_0^L N_2(x) q\, \mathrm{d}x = 0\,, \end{aligned}$$

and simplifying further

$$\begin{aligned} &-\left(\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x}\, \mathrm{d}x\right) a_1 + -\left(\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_2}{\partial x}\, \mathrm{d}x\right) a_2 + \int_0^L N_1(x) q\, \mathrm{d}x = 0\,, \\ &-\left(\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_1}{\partial x}\, \mathrm{d}x\right) a_1 + -\left(\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_2}{\partial x}\, \mathrm{d}x\right) a_2 + \int_0^L N_2(x) q\, \mathrm{d}x = 0\,, \end{aligned}$$

Here the terms

$$K_{i,j} = \int_0^L \frac{\partial N_i}{\partial x} P \frac{\partial N_j}{\partial x}\, \mathrm{d}x$$

are the elements of the stiffness matrix, and

$$F_i = \int_0^L N_i(x) q\, \mathrm{d}x$$

are the elements of the load vector. These objects can be computed symbolically using only a few lines of Matlab symbolic algebra (note that

```
>> diff([N1;N2])*P*diff([N1,N2])
ans =
[(P*pi^2*cos(pi*x/L)^2)/L^2, (3*P*pi^2*cos(pi*x/L)*cos(3*pi*x/L))/L^2]
[(3*P*pi^2*cos(pi*x/L)*cos(3*pi*x/L))/L^2, (9*P*pi^2*cos(3*pi*x/L)^2)/L^2]
```

is a 2×2 matrix):

```
>> syms   L P q x real
N1=sin(pi*x/L);
N2=sin(3*pi*x/L);
```

```
K=int(diff([N1;N2])*P*diff([N1,N2]),0,L)
F=q*int([N1;N2],0,L)
a=K\F
K =
[ (P*pi^2)/(2*L),                 0]
[              0, (9*P*pi^2)/(2*L)]
F =
     (2*L*q)/pi
 (2*L*q)/(3*pi)
a =
    (4*L^2*q)/(P*pi^3)
 (4*L^2*q)/(27*P*pi^3)
```

yielding The analytical solution is

$$w_{ex}(x) = \frac{qx(L-x)}{2P}$$

which can be compared with the approximate Galerkin solution

$$w_{app}(x) = \frac{4L^2q}{P\pi^3}\sin(\pi x/L) + \frac{4L^2q}{27P\pi^3}\sin(3\pi x/L)$$

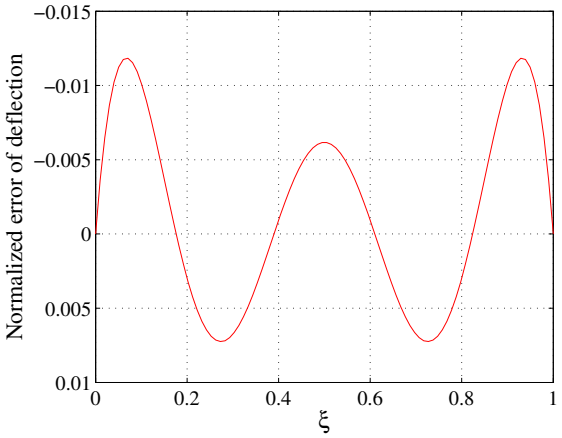
The midpoint deflections are: analytical

$$w_{ex}(L/2) = \frac{qL^2}{8P} = \frac{0.125qL^2}{P}$$

versus approximate

$$w_{app}(L/2) = \frac{104qL^2}{27\pi^3 P} \approx \frac{0.1242qL^2}{P}$$

The deflection error, that is the difference between the approximate and analytical deflections normalized by $w_{ex}(L/2)$ versus the nondimensional coordinate $\xi = x/L$ is shown in the figure below. Clearly the largest error is just a little bit over 1%.



Exercise 9.

Solve for the approximate deflection of a prestressed table with uniform load, simply supported at $x = 0$ and force-free boundary condition at $x = L$, using the Galerkin method. Take as the trial function basis the to functions $N_1(x) = x$ and $N_2(x) = x^2$. The test functions are $N_j, j = 1, 2$.

Solution: We are solving the boundary value problem:

$$\begin{aligned} Pw'' + q &= 0 \\ w(0) &= 0 \\ w'(L) &= 0 \end{aligned}$$

The trial function is

$$w(x) = a_1 N_1(x) + a_2 N_2(x)$$

Note that the trial function satisfies the essential boundary condition $w(0) = 0$ because

$$N_1(0) = N_2(0) = 0$$

The coefficients a_1, a_2 are the unknowns.

Equation (2.14) simplifies for statics (zero accelerations) and a natural boundary conditions at $x = L$ to

$$\eta_j(L) F_L - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x} \, \mathrm{d}x + \int_0^L \eta_j(x) q(x) \, \mathrm{d}x = 0 \, .$$

However, since $F_L = 0$ (that is the meaning of "force-free"), there will be no contribution of the first term in the final equations.

Substituting the test and trial function we have

$$\begin{aligned} -\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x} a_1 \, \mathrm{d}x + -\int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_2}{\partial x} a_2 \, \mathrm{d}x + \int_0^L N_1(x) q \, \mathrm{d}x = 0 \, , \\ -\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_1}{\partial x} a_1 \, \mathrm{d}x + -\int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_2}{\partial x} a_2 \, \mathrm{d}x + \int_0^L N_2(x) q \, \mathrm{d}x = 0 \, , \end{aligned}$$

The Matlab symbolic algebra is again helpful (note how minimal the needed changes are with respect to the code in exercise 8– can you think of a better argument for using computers for the grunt work?):

```
>> syms   L P q x real
N1=x;
N2=x^2;
K=int(diff( [N1;N2])*P*diff([N1,N2]),0,L)
F=q*int([N1;N2],0,L)
a=K\F
```

The quantities of the discrete system are

```
K =
[   L*P,       L^2*P]
[ L^2*P, (4*L^3*P)/3]
F =
 (L^2*q)/2
 (L^3*q)/3
a =
  (L*q)/P
 -q/(2*P)
```

This yields the approximate solution as

```
>> w=simplify(N1*a(1)+N2*a(2))
w =
(q*x*(2*L - x))/(2*P)
```

and since we obtained a quadratic polynomial we may begin to suspect that we arrived at the exact solution. Firstly we check the balance equation

```
>> P*diff(diff(w))+q
ans =
0
```

and secondly we check the boundary condition at $x = L$

```
>> subs(diff(w),x,L)
ans =
0
```

Since those equations are satisfied, we have verified that the approximate solution is in fact exact.

Exercise 10.

Solve for the approximate deflection of a prestressed table without any distributed load, simply supported at $x = 0$ and nonzero concentrated force boundary condition at $x = L$, using the Galerkin method. Take as the trial function basis the two functions $N_1(x) = \sin(\pi x/L)$ and $N_2(x) = (3/2)\sin(\pi x/L)$. The test functions are $N_j, j = 1, 2$.

Solution: We are solving the boundary value problem:

$$\begin{aligned} Pw'' &= 0 \\ w(0) &= 0 \\ Pw'(L) &= F_L \end{aligned}$$

The trial function is

$$w(x) = a_1 N_1(x) + a_2 N_2(x)$$

Note that the trial function satisfies the essential boundary condition $w(0) = 0$ because

$$N_1(0) = N_2(0) = 0$$

The coefficients a_1, a_2 are the unknowns.

Equation (2.14) simplifies for statics (zero accelerations) and a natural boundary conditions at $x = L$ to

$$\eta_j(L)F_L - \int_0^L \frac{\partial \eta_j}{\partial x} P \frac{\partial w}{\partial x}\, \mathrm{d}x + \int_0^L \eta_j(x) q(x)\, \mathrm{d}x = 0\ .$$

However, this time $F_L \neq 0$ and there will be a contribution of the first term in the final equations. Also note that the two test functions $N_j, j = 1, 2$ must not be simultaneously zero at $x = L$. If that was the case, all effects of the applied force F_L would be erased from the formulation, and we couldn't possibly get a meaningful result.

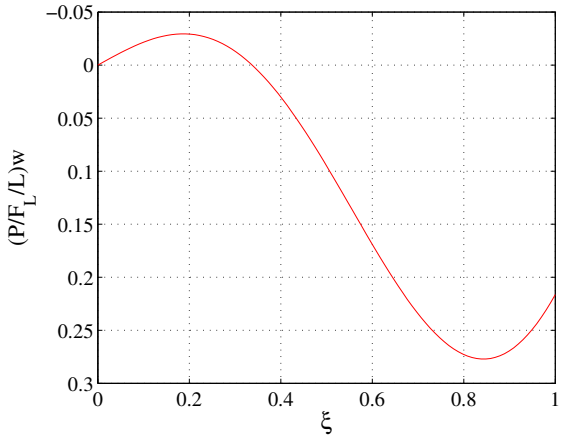
Substituting the test and trial function we have

$$\begin{aligned} F_L N_1(L) - \int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_1}{\partial x} a_1\, \mathrm{d}x - \int_0^L \frac{\partial N_1}{\partial x} P \frac{\partial N_2}{\partial x} a_2\, \mathrm{d}x = 0\ , \\ F_L N_2(L) - \int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_1}{\partial x} a_1\, \mathrm{d}x - \int_0^L \frac{\partial N_2}{\partial x} P \frac{\partial N_2}{\partial x} a_2\, \mathrm{d}x = 0\ , \end{aligned}$$

The Matlab symbolic algebra reads

```
>> syms   L P q x FL real
q=0;
N1=sin(pi*x/L);
N2=sin(3/2*pi*x/L);
K=int(diff( [N1;N2])*P*diff([N1,N2]),0,L)
F=q*int([N1;N2],0,L)+ [FL*subs(N1,x,L);FL*subs(N2,x,L)]
a=K\F
K =
[ (P*pi^2)/(2*L),   (9*pi*P)/(5*L)]
[ (9*pi*P)/(5*L), (9*P*pi^2)/(8*L)]
F =
   0
 -FL
a =
  (80*FL*L)/(P*pi*(25*pi^2 - 144))
 -(200*FL*L)/(9*P*(25*pi^2 - 144))
```

This yields the approximate solution as shown in the figure below.



While the approximate solution is computed correctly, it is far from satisfactory. The curvature of the cable is clearly nonzero, and that contradicts the balance equation which gives zero curvature when the transverse load is zero. The force balance at the right hand side end of the cable is definitely not satisfied: the force F_L points downward, and so does the prestressing force P– they cannot balance. The bad features of the approximate solution are the result of the choice of the basis functions. Sinusoidal curves are wrong for the present purpose!

2.9 Piecewise linear basis functions

Let us recall the piecewise linear approximation proposed for the trial function in Section 2.4. The broken line cannot be represented as a linear combination of linear functions that are all defined on the whole interval $0 \leq x \leq L$ (only two such functions are linearly independent, and these functions cannot represent the corners in the broken line). Therefore, we have to describe the piecewise linear curve interval by interval.

The interpolant may be written as a linear combination of basis functions. In one dimension, the piecewise linear basis function is called the ***hat function***. The six functions that are shown in Fig. 2.5, all are examples of hat functions. For reasons that will be discussed later, we would want the hat functions in a linear combination to be able to reproduce an arbitrary linear function over

the whole interval. Because of the way in which we construct the hat functions in Fig. 2.5, this property is automatically available.

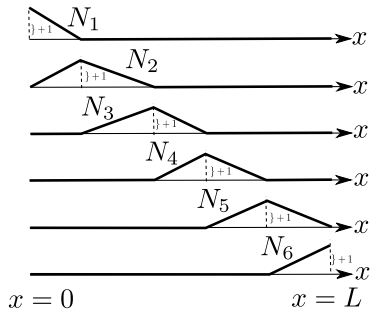


Fig. 2.5. Piecewise linear basis functions

Let us describe the construction of the piecewise linear basis functions. First, the length of the wire is divided into disjoint subintervals. These subintervals are the ***finite elements*** for the one-dimensional domain. The end-points of the finite elements are called ***nodes***. Together, the finite elements and the nodes are known as the ***finite element mesh***: see Fig. 2.6 (the element numbers are in the boxes; nodes are indicated by filled circles). In this book, the one-dimensional elements with two nodes at the end points are going to be referred to as L2.

We are going to construct the basis functions so that they are piecewise linear, and they assume values that are zero at all nodes of the mesh except one. We will associate basis functions with nodes so that when a basis function is nonzero at node K the function will be called N_K. We say ***basis function*** N_K ***is associated with node*** K.

Since all basis functions are constructed in the same way, we describe the procedure for the basis function N_3 (i.e. associated with node 3): as shown in the Fig. 2.5, it is nonzero over two elements, 2 and 3; zero everywhere else. To be able to write it down over the two adjacent elements, we have to agree on the value of N_3 at node 3 (i.e. $N_3(x_3)$), which is shared by elements 2 and 3. Choosing $N_3(x_3) = 1$ has certain advantages, which will be introduced momentarily. Using the concept of Lagrange interpolation polynomials, we may write the function N_3 within element 2 as

$$N_3(x) = \frac{x - x_2}{x_3 - x_2}, \quad x_2 \le x \le x_3 \ ,$$

and within element 3 as

$$N_3(x) = \frac{x - x_4}{x_3 - x_4}, \quad x_3 \le x \le x_4 \ .$$

Note that the basis function are "non-dimensional": the formula above says length/length= no dimension (we may say that the physical units are [ND])!

All the other functions N_i are expressed analogously. Putting them together in a linear combination for the trial function, we write

$$w(x) = \sum_{i=1}^{N} N_i(x) w_i \ , \tag{2.22}$$

(for simplicity, we omit the time argument). The physical unit of the deflection $w(x)$ is [length], and since the basis functions themselves are non-dimensional, the degrees of freedom w_i must have the physical units of [length].

Evaluating $w(x)$ at the node k, we obtain

$$w(x_k) = \sum_{i=1}^{N} N_i(x_k) w_i \ ,$$

where the crucial expression is $N_i(x_k)$: by definition, the basis function N_k has value $+1$ at x_k, while all other functions $N_i, i \neq k$ are zero at x_k. This property is usually expressed mathematically as

$$N_i(x_k) = \delta_{ik} \ , \tag{2.23}$$

where the symbol δ_{ik} is known as the ***Kronecker delta***

$$\delta_{ik} = \begin{cases} 1, \text{ if } i = k; \\ 0, \text{ otherwise.} \end{cases}$$

Because of this property, the value of $w(x_k)$ is

$$w(x_k) = \sum_{i=1}^{N} N_i(x_k) w_i = \sum_{i=1}^{N} \delta_{ik} w_i = w_k \ ,$$

and we see that the parameters w_i have the physical meaning of the value of the interpolated function at the node i. The w_i's are usually called the ***degrees of freedom***, since, being the control parameters of the trial function, they determine the shape of the actual solution from all the possible shapes of the trial function. They are the objects that our numerical method solves for.

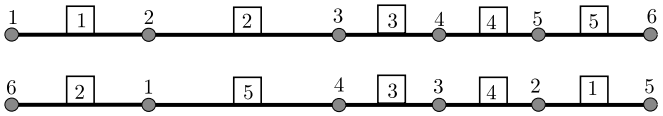
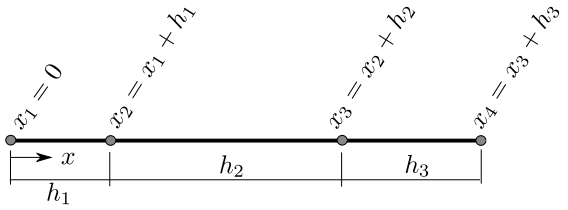


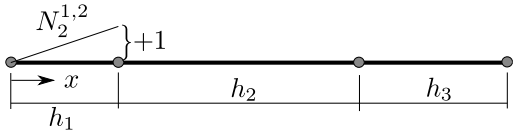
Fig. 2.6. The finite element mesh. Two different but completely equivalent numberrings of the nodes and elements. The top mesh consists of elements 1:(1,2), 2:(2,3), 3:(3,4), 4:(4,5), 5:(5,6). The bottom mesh consists of elements 1:(2,5), 2:(6,1), 3:(4,3), 4:(3,2), 5:(1,4).

Exercise 11.

For the shown finite element mesh express the finite element basis functions and their derivatives as expressions in the independent variable x. Associate the basis functions with the nodes j whose locations are x_j.



Solution: We shall use the so-called Lagrange interpolation polynomials to construct the segments of the individual basis functions. All the segments of all the basis functions are linear functions. For instance, the segment of the basis function N_2 over the element $1, 2$, which we will call $N_2^{1,2}$ passes through the points $(x_1, 0)$ and $(x_2, +1)$.



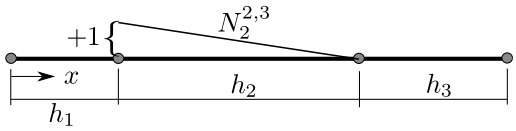
The linear function

$$(x - x_1)$$

becomes zero at $x = x_1$. It is not equal to $+1$ at $x = x_2$. In order to enforce this property we normalize by the value of the function $(x - x_1)$ at $x = x_2$ to obtain

$$N_2^{1,2}(x) = \frac{(x - x_1)}{(x_2 - x_1)}$$

The segment of the basis function N_2 over the element $2, 3$, which we will call $N_2^{2,3}$ passes through the points $(x_2, +1)$ and $(x_3, 0)$.



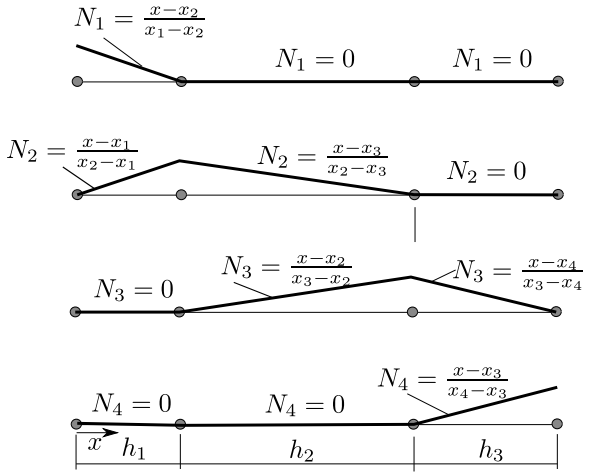
To design this segment we realize that the linear function

$$(x - x_3)$$

becomes zero at $x = x_3$, and since it is not equal to $+1$ at $x = x_2$ we normalize by the value of the function $(x - x_3)$ at $x = x_2$ to obtain

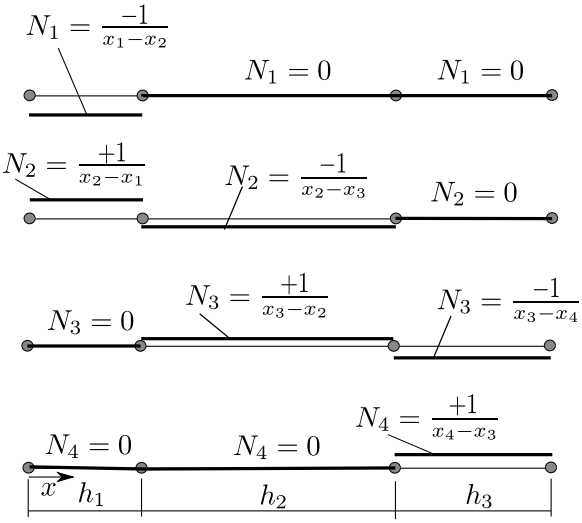
$$N_2^{2,3}(x) = \frac{(x - x_3)}{(x_2 - x_3)}$$

All four basis functions are summarized in this figure:



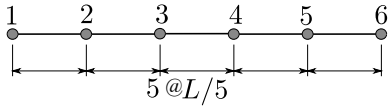
Note that a basis function is nonzero only over the two elements which share the node at which the basis function assumes value $+1$.

The derivatives follow by simple differentiation of the segments of the basis functions with respect to x. Note that the derivatives are piecewise constant as they measure the slope of the linear segments of the basis functions.



Exercise 12.

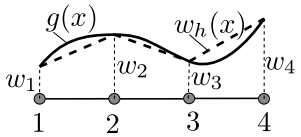
Interpolate the function $\cos(\pi x/L)$ on the interval $0 \le x \le L$ using a mesh of five L2 (i.e. linear) finite elements of equal length.



Solution: Interpolation of the given function on the mesh is understood as the task of constructing a linear combination of the basis functions defined on the mesh

$$w_h(x) = \sum_j N_j(x) w_j$$

so that the linear combination is equal to the interpolated function $g(x)$ at the nodes.



Mathematically, we say

$$g(x_k) = w_h(x_k) \quad \text{for all } k$$

which we call the *interpolating condition*. The coefficients of the linear combination w_j need to be determined from the interpolating conditions. For the finite element basis functions this is a breeze because of the Kronecker delta property

$$N_j(x_k) = \begin{cases} +1, \text{ if } j = k; \\ 0, \quad \text{otherwise.} \end{cases}$$

This gives

$$w_h(x_k) = \sum_j N_j(x_k) w_j = \underbrace{N_1(x_k)}_{0} w_1 + \ldots + \underbrace{N_k(x_k)}_{+1} w_k + \ldots + \underbrace{N_n(x_k)}_{0} w_n = w_k$$

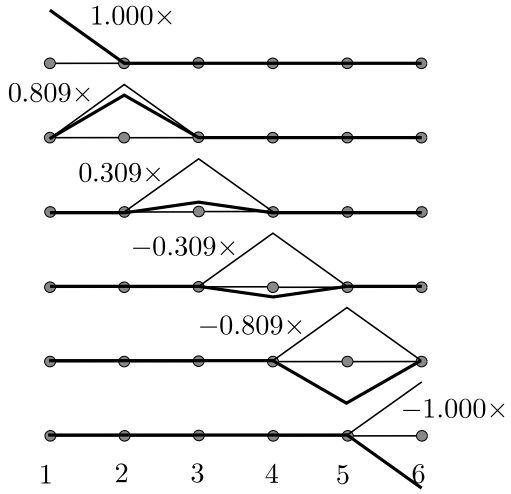
so that $w_k = g(x_k)$. For our given function $g(x)$ we can therefore write the interpolation function as

$$w_h(x) = \sum_j N_j(x) \cos(\pi x_j / L)$$

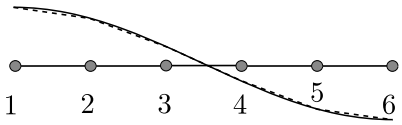
and numerically the coefficients of the linear combination are

```
>> cos(pi*[0:0.2:1.0])
ans =
    1.0000    0.8090    0.3090   -0.3090   -0.8090   -1.0000
```

The construction of the linear combination is depicted in this figure:



And here are the interpolated (solid line) and interpolating (dashed line) functions.



Exercise 13.

Interpolate the function $g(x) = Ax^2 + Bx + C$ on the interval $0 \le x \le h$ using a single L2 finite element mesh. Discuss the interpolation error.

Solution: Interpolation of the given function on the finite element mesh is understood as a linear combination of the basis functions defined on the mesh

$$w_h(x) = \sum_j N_j(x) w_j$$

so that the linear combination is equal to the interpolated function $g(x)$ at the nodes. For a single-element mesh there are only two basis functions that are nonzero within the element. As shown in exercise 12, using the interpolating conditions to determine the coefficients of the linear combination we can write

$$w_h(x) = N_1(x) w_1 + N_2(x) w_2$$

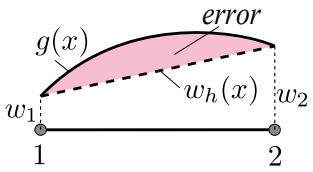
where

$$N_1(x) = (x - x_2)/(x_1 - x_2), \quad N_2(x) = (x - x_1)/(x_2 - x_1)$$

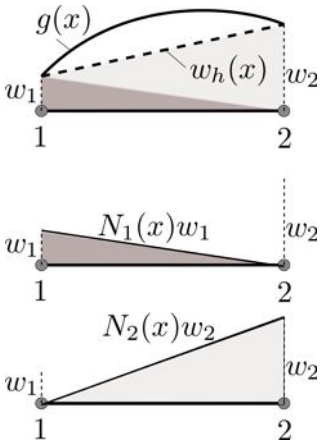
and

$$w_1 = g(x_1) = Ax_1^2 + Bx_1 + C\,, \quad w_2 = g(x_2) = Ax_2^2 + Bx_2 + C$$

Since the two basis functions are linear, their linear combination will also be linear. The error is the difference between the interpolated function $g(x)$ and the interpolating (linear) function $w_h(x)$



Evidently, the error is zero at the nodes. The graphics below shows how the two basis functions are multiplied by the coefficients of the linear combination and added together to form $w_h(x)$.



Since the error is zero at the nodes (the interpolating conditions!), and since a linear function is defined uniquely by two points, we may suspect that $w_h(x)$ can exactly (without error) interpolate (or as we say "reproduce") an arbitrary linear function. In fact a little bit of symbolic manipulation confirms this: We form symbolically the two functions $g(x)$ and $w_h(x)$

```
>> syms   A B C x1 x2 x real
g=A*x^2+B*x+C;
w1 =subs(g,x,x1);
w2 =subs(g,x,x2);
wh=(x-x2)/(x1-x2)*w1+(x-x1)/(x2-x1)*w2;
```

and then we compute the error

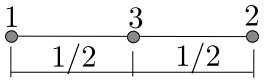
```
simplify(g-wh)
ans =
A*(x - x1)*(x - x2)
```

The error term is a quadratic expression in x, and it is zero when $A = 0$. In other words, when the interpolated function is linear $g(x) = Bx + C$, the interpolation on the mesh is without error. That makes sense: Note that the coefficient A is related to the curvature: $g'' = 2A$. A piecewise linear function will not be able to match a curve with nonzero curvature.

Exercise 14.

Interpolate the function $g(x) = -x^2+1.3x+1/3$ on a two-element mesh on the interval $0 \le x \le 1$. Discuss the approximation of the derivative by the interpolating function.

Solution: For variety we will set up the mesh as shown: elements 1:(3,2), 2:(1,3).



The interpolating function is constructed element-by-element. The basis functions are associated with nodes, basis function N_j with node j.

Within the extent of element 1 the interpolating function is

$$w_h(x) = N_3(x)w_3 + N_2(x)w_2$$

where

$$N_3(x) = (x - x_2)/(x_3 - x_2), \quad N_2(x) = (x - x_3)/(x_2 - x_3)$$

and

$$w_3 = -(1/2)^2 + 1.3(1/2) + 1/3 = 22/30\,, \quad w_2 = -(1)^2 + 1.3(1) + 1/3 = 19/30$$

Within the extent of element 2 the interpolating function is

$$w_h(x) = N_1(x)w_1 + N_3(x)w_3$$

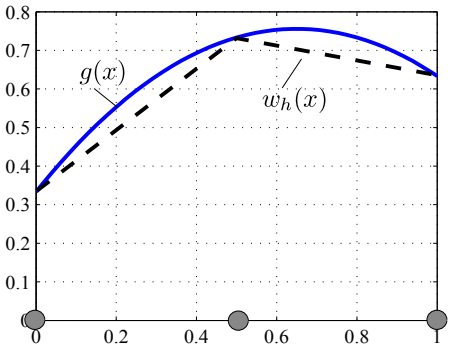
where

$$N_1(x) = (x - x_3)/(x_1 - x_3), \quad N_3(x) = (x - x_1)/(x_3 - x_1)$$

and

$$w_3 = -(1/2)^2 + 1.3(1/2) + 1/3 = 22/30\,, \quad w_1 = -(0)^2 + 1.3(0) + 1/3 = 10/30$$

The interpolated and interpolating functions are shown here:



To compare their derivatives we just differentiate both the interpolated and interpolating function. The derivative of the interpolated function is continuous, linear, $g'(x) = -2x + 1.3$. The derivative of the interpolating function needs to be again computed element-by-element. Within the extent of element 1 the interpolating function derivative is

$$w_h'(x) = N_3'(x)w_3 + N_2'(x)w_2$$

where

$$N_3'(x) = 1/(x_3 - x_2), \quad N_2'(x) = 1/(x_2 - x_3)$$

and the coefficients w_3, w_2 are as computed above. Consequently we get

$$w_h'(x) = \frac{w_2 - w_3}{x_2 - x_3} = -1/5$$

that is rise-over-run as expressed from the geometry of a straight line.

Within the extent of element 2 the derivative of the interpolating function is

$$w_h'(x) = N_1'(x)w_1 + N_3'(x)w_3$$

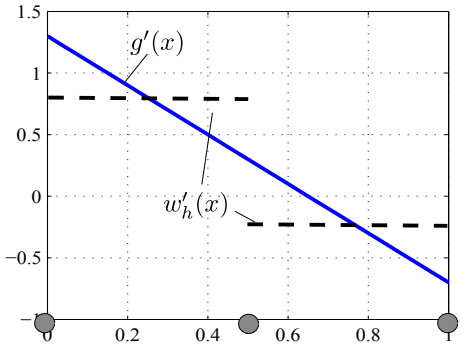
where

$$N_1'(x) = 1/(x_1 - x_3), \quad N_3'(x) = 1/(x_3 - x_1)$$

and w_3, w_1 are as above. Substitution leads in element 2 to

$$w_h'(x) = \frac{w_3 - w_1}{x_3 - x_1} = 4/5$$

Note well that the derivative lowered the order of the functions, the derivative of the quadratic interpolated function being linear, and the derivative of the piecewise linear interpolating function being piecewise constant. As a consequence, the derivative of the interpolating function is discontinuous. Therefore, taking the slope of the first derivative of $w_h'(x)$ would be tricky. Usually the consequence of negotiating the discontinuity at the node means following a positive or negative infinite slope.



2.10 Bookkeeping in the finite element method

In order to automate processing of the information in a finite element model we introduce some bookkeeping devices.

All nodes are numbered sequentially from 1 to N, but in an arbitrary order. All elements connect some nodes. In the cable problem, each element connects two nodes. The so-called element ***connectivity*** are the numbers of the nodes, given in the order left to right (in other words, the nodes are given in the order in which their coordinates increase).

Perhaps most importantly, all the degree of freedom parameters in the model will be numbered sequentially. First the ***free degrees of freedom*** will be numbered (those degrees of freedom that are unknown and need to be solved for), and only then the degrees of freedom that are ***prescribed*** (known from the boundary conditions) will be numbered. Since eventually all the parameters are associated with equations in which the parameters occur we also refer to this numbering as the "equation numbering".

Figure 2.7 illustrates the bookkeeping information. Node numbers are in circles. Element numbers are in boxes. Element 1 connects nodes 1 and 2, element 2 connects nodes 2 and 3. There are two free degrees of freedom, 1 and 2. The deflections w_1, w_2 are unknown. There is one prescribed degree of freedom, w_3, which is given by the pinned boundary condition as $w_3 = 0$.

The basis functions are always associated with nodes: basis function N_1 with node 1 and so on. On the other hand, the numbering of the unknowns does not necessarily correspond to the numbering of the nodes. Therefore we resort to the following notation: the finite element expansion is written as

$$w_h(x) = \sum_{j=1}^{N} N_j(x) w_{(j)} \tag{2.24}$$

where ***by*** $w_{(j)}$ ***we mean the unknown at node*** j. For the mesh in Figure 2.7 the finite element expansion becomes

$$w_h(x) = N_1(x)w_{(1)} + N_2(x)w_{(2)} + N_3(x)w_{(3)} = N_1(x)w_3 + N_2(x)w_1 + N_3(x)w_2$$

which corresponds to the degree of freedom 3 being at node 1 (so $(1) \leftrightarrow 3$), and so on (i.e. $(2) \leftrightarrow 1$, $(3) \leftrightarrow 2$).

We might also need to find the node at which a given degree of freedom resides. We will use the notation $\langle j \rangle$ to ***mean the node with the degree of freedom*** j. For the mesh in Figure 2.7 we have $\langle 1 \rangle \leftrightarrow 2$, $\langle 2 \rangle \leftrightarrow 3$, and $\langle 3 \rangle \leftrightarrow 1$.

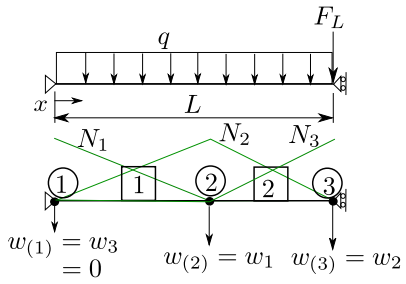


Fig. 2.7. Finite element mesh with bookkeeping information

Next we will introduce the finite element trial expansion (2.24) into the Galerkin method by using it for the trial function, and by setting the test functions to be equal to the basis functions themselves. We shall start with an example. We will solve for the deflection of the shown prestressed cable using the finite element Galerkin method (see Figure 2.8). We will use a mesh of two L2 finite elements of equal length.

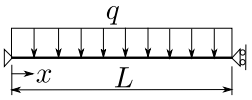


Fig. 2.8. Prestressed cable configuration

The Galerkin weighted residual statement for the BVP for the static deflection of the pin-roller cable is written as shown in equation (2.14) (the first term is eliminated since F_L is zero)

$$-\int_0^L \eta_j{}' P w' \, \mathrm{d}x + \int_0^L \eta_j q \, \mathrm{d}x = 0, \quad j = 1, ..., N_\mathrm{f} \ ,$$

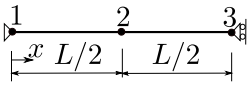
where we require for the test and trial function

$$\eta_j(x=0)=0, \quad \eta_j \in C^0, \quad j=1,...,N_{\mathrm{f}}\ ,$$
$$w(x=0)=\bar{w}_0, \quad w \in C^0,$$

and N_{f} is the number of free degrees of freedom.

The mesh consists of two L2 elements

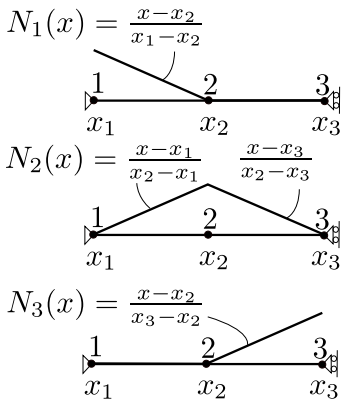
Element	Nodes
1	1,2
2	2,3



The degrees of freedom will be also numbered. We start the numbering with the unknowns (deflections at nodes 2, 3), and we conclude the numbering with the deflection at node 1 which is known. Sometimes we refer to the numbers of degrees of freedom interchangeably as "equation numbers".

Node	Degree of freedom (equation) number
1	3
2	1
3	2

The basis (hat) functions are associated with the nodes.



The trial function is written as

$$w(x)=\sum_{k=1}^{3} N_k(x) w_{(k)}$$

where (k) means equation number (dof number) associated with node k, and therefore $w_{(k)}$ means the deflection at node k. For instance, $(1)=3$, and $w_{(1)}=w_3$. Note however that it may also be written as

$$w(x)=\sum_{k=1}^{3} N_{\langle k\rangle}(x) w_k$$

where k means equation member, which is associated with node $\langle k\rangle$, and therefore w_k means the deflection at node $\langle k\rangle$. In the first form of the trial function the sum was over all the nodes, whereas in the second form the sum is over all the degrees of freedom.

As given in the BVP (2.14), (2.15), the trial function must satisfy the essential boundary condition. Thus we a require

$$w(0) = \sum_{k=1}^{3} N_k(0) w_{(k)} = 0$$

Since we have $N_1(0) = 1$ and $N_2(0) = N_3(0) = 0$ this condition is equivalent to

$$w(0) = w_{(1)} = w_3 = 0$$

We see that the essential boundary condition determines the value of the prescribed degree of freedom at the pin $w_3 = 0$.

We are using the finite element Galerkin method, hence the test functions are taken to be the hat functions N_j on the mesh. The test functions must satisfy the condition $\eta_j(x = 0) = 0$, which means that only N_2 and N_3 are allowed since they are both zero at the left-hand side end of the wire. Thus we take $\eta_1 = N_2$ and $\eta_2 = N_3$. Because the test functions are from now on always going to be the finite element basis functions, we may just as well start writing the weighted residual equation as

$$-\int_0^L N_{\langle j \rangle}{}' P w' \, \mathrm{d}x + \int_0^L N_{\langle j \rangle} q \, \mathrm{d}x = 0, \;\; j = 1, ..., N_\mathrm{f} \, ,$$

where $N_{\langle j \rangle}$ is the basis function associated with node number $\langle j \rangle$ which carries the degree of freedom j.

The elements of the load vector L_j are now computed for $j = 1, 2$. We begin with L_1: first we see that the integral should be split into integrals over each element, since the test function $N_{\langle 1 \rangle} = N_2$ has different definitions in different elements.

$$L_1 = \int_0^L N_{\langle 1 \rangle} q \, \mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 1 \rangle} q \, \mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 1 \rangle} q \, \mathrm{d}x$$

For element 1 we compute the contribution to L_1 as

$$\int_{x_1}^{x_2} N_{\langle 1 \rangle} q \, \mathrm{d}x = \int_{x_1}^{x_2} N_2 q \, \mathrm{d}x = qL/4$$

and for element 2 we compute the contribution to L_1 as

$$\int_{x_2}^{x_3} N_{\langle 1 \rangle} q \, \mathrm{d}x = \int_{x_2}^{x_3} N_2 q \, \mathrm{d}x = qL/4$$

yielding $L_1 = qL/2$.

The load vector component L_2 is computed as

$$L_2 = \int_0^L N_{\langle 2 \rangle} q \, \mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 2 \rangle} q \, \mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 2 \rangle} q \, \mathrm{d}x$$

were the contribution from element 1 is zero, since $N_{\langle 2 \rangle} = N_3 = 0$ in element 1, and the contribution to L_2 from element 2 is

$$\int_{x_2}^{x_3} N_{\langle 2 \rangle} q \, \mathrm{d}x = qL/4$$

The load vector is therefore

$$[L] = \begin{bmatrix} qL/2 \\ qL/4 \end{bmatrix}$$

The components of the stiffness matrix are computed next. Substituting the second form of the trial function, $w(x) = \sum_{k=1}^{3} N_{\langle k \rangle}(x) w_k$, into the weighted residual equation we obtain for the stiffness term

$$\int_0^L N_{\langle j\rangle}{}'Pw'\,\mathrm{d}x = \int_0^L N_{\langle j\rangle}{}'P\left(\sum_{k=1}^{3} N_{\langle k\rangle}w_k\right)'\,\mathrm{d}x = \sum_{k=1}^{3}\left(\int_0^L N_{\langle j\rangle}{}'PN_{\langle k\rangle}{}'\,\mathrm{d}x\right)w_k$$

The component K_{11} $(j=1,k=1)$ is

$$K_{11} = \int_0^L N_{\langle 1\rangle}{}'PN_{\langle 1\rangle}{}'\mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 1\rangle}{}'PN_{\langle 1\rangle}{}'\mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 1\rangle}{}'PN_{\langle 1\rangle}{}'\mathrm{d}x = \frac{1}{\frac{L}{2}}P\frac{1}{\frac{L}{2}}\frac{L}{2} + \frac{-1}{\frac{L}{2}}P\frac{-1}{\frac{L}{2}}\frac{L}{2} = \frac{4P}{L}$$

The component K_{12} is

$$K_{12} = \int_0^L N_{\langle 1\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 1\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 1\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x = 0 + \frac{-1}{\frac{L}{2}}P\frac{1}{\frac{L}{2}}\frac{L}{2} = -\frac{2P}{L}$$

where the contribution of the first integral on the right-hand side is zero since $N_{\langle 2\rangle}$ is zero in element 1. The component K_{21} is the same as K_{12} since the functions in the integrals are the same. Finally, the component K_{22} is

$$K_{22} = \int_0^L N_{\langle 2\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 2\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 2\rangle}{}'PN_{\langle 2\rangle}{}'\,\mathrm{d}x = 0 + \frac{1}{\frac{L}{2}}P\frac{1}{\frac{L}{2}}\frac{L}{2} = \frac{2P}{L}$$

In matrix form the stiffness has the appearance

$$[K] = \begin{bmatrix} \frac{4P}{L}\,, & -\frac{2P}{L} \\ -\frac{2P}{L}\,, & \frac{2P}{L} \end{bmatrix} \tag{2.25}$$

The discrete equations of equilibrium are therefore written as

$$[K][w] = [L]$$

which may be rewritten in a clean way as

$$\frac{2P}{L}\begin{bmatrix} 2\,, & -1 \\ -1\,, & 1 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{qL}{4}\begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

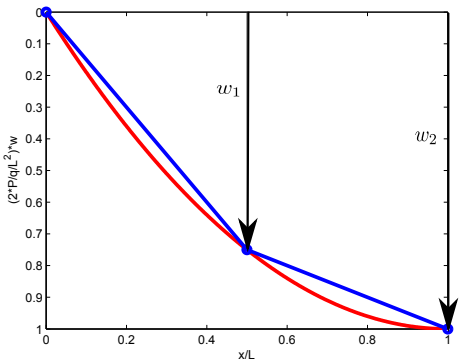
The solution is

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{L}{2P}\frac{qL}{4}\begin{bmatrix} 2\,, & -1 \\ -1\,, & 1 \end{bmatrix}^{-1}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \frac{qL^2}{8P}\begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

The approximate deflection function is written as

$$w(x) = \sum_{k=1}^{3} N_{\langle k\rangle}(x)w_k = N_2(x)w_1 + N_3(x)w_2$$

which is a piecewise linear function since the hat functions are piecewise linear. The figure below shows the deflection scaled by $2P/(qL^2)$, compared to the analytical solution curve.



2.11 Finite element Galerkin method

Henceforth we will consider only Galerkin methods that use finite element basis functions N_k. Therefore, we will use the following notation for the weighted residual equation:

$$N_{\langle j\rangle}(L)F_L - \sum_{i=1}^{N}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x = 0, \quad j = 1, ..., N_\mathrm{f} \ . \tag{2.26}$$

Note that the stiffness matrix defined by the integrals $\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x$ is effectively rectangular: N is the number of columns, and N_f is the number of rows, and these two are in general not the same (if there are any essential boundary conditions, the number of free degrees of freedom is less than the total number of degrees of freedom). The stiffness matrix is wider than it is tall: there are less equations than there are degrees of freedom multiplying the columns of the matrix. That is okay, since some of these degrees of freedom are actually known. The number of unknown degrees of freedom is exactly equal to the number of rows (equations). Therefore, we can split the stiffness matrix into a square matrix and a rectangular matrix (narrower than it is tall):

$$N_{\langle j\rangle}(L)F_L - \sum_{i=1}^{N_\mathrm{f}}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i - \sum_{i=N_\mathrm{f}+1}^{N}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x = 0, \quad j = 1, ..., N_\mathrm{f} \ . \tag{2.27}$$

The term

$$\sum_{i=1}^{N_\mathrm{f}}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i \ , \quad j = 1, ..., N_\mathrm{f}$$

represents the product of a square matrix with a vector consisting of the free degrees of freedom; and the term

$$\sum_{i=N_\mathrm{f}+1}^{N}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i \ , \quad j = 1, ..., N_\mathrm{f}$$

represents the product of a rectangular matrix with a vector consisting of the prescribed degrees of freedom. We can rewrite (2.27) in an explicit way that makes clear the distinction between known and unknown

$$\sum_{i=1}^{N_\mathrm{f}}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i = N_{\langle j\rangle}(L)F_L - \sum_{i=N_\mathrm{f}+1}^{N}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x, \quad j = 1, ..., N_\mathrm{f} \ . \tag{2.28}$$

On the left we have a square matrix multiplied by the vector of unknowns, on the right we have only known quantities. Later we will discuss under which conditions a solution exists, but at least the coefficient matrix on the left-hand side is square which is the first condition of its invertibility.

The square matrix on the left-hand side is sometimes called the ***free-free stiffness matrix***, while the rectangular stiffness matrix on the right is sometimes called the free-prescribed stiffness. Both matrices are sometimes given the epithet "global" .

It is worthwhile to stop and think about the meaning of equations (2.28). Is quite clear that on the right the components of the vector are forces (for instance: integrals over length of a non-dimensional function times force per unit length=force). Therefore, on the left we also must have forces. These forces are obtained as the products of w_i (the coefficients that represent the degrees of freedom

with physical unit [length]) and K_{ji} which must have physical units of [force/length]. The forces on the left represent the resistance of the taut wire to deformation, and (2.28) therefore expresses the equilibrium of the applied forces with the forces that resist deformation. Equation (2.28) is an ***equation of equilibrium***.

When the finite element functions as defined in Section 2.9 are chosen as the basis functions N_j, and the unknowns are deflections at the nodes, $w(x) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) w_i$, the so-called Galerkin Finite Element Methods (GFEM) result; otherwise we get Galerkin methods that are *not* finite element methods. On the other hand, finite element methods that are not Galerkin methods are quite common in certain applications (fluid mechanics, for instance). This classification is illustrated in Fig. 2.9.

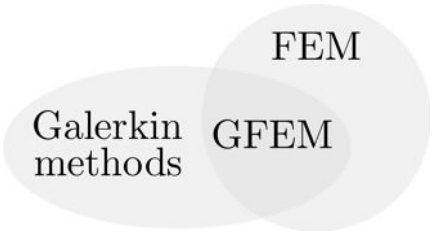


Fig. 2.9. The relationship of Galerkin and finite element methods (FEM). The intersection of the two sets are the Galerkin Finite Element Methods (GFEM).

2.12 Element-by-element computations

An especially powerful organizing principle in the finite element method that makes it easy to automate is that the computation of the necessary matrices and vectors can be carried out element-by-element.

First an example of such a calculation. We shall repeat the computation of Section 2.10 and identify opportunities for element-by-element computations. We begin with L_1:

$$L_1 = \int_0^L N_{\langle 1 \rangle} q \, \mathrm{d}x = \int_{x_1}^{x_2} N_{\langle 1 \rangle} q \, \mathrm{d}x + \int_{x_2}^{x_3} N_{\langle 1 \rangle} q \, \mathrm{d}x$$

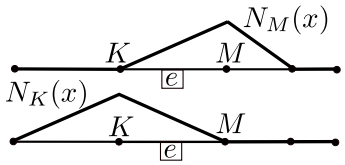
We see that the operation that would do this for all load vector components could be described by

```
Loop over all load vector components j
    Loop over all finite elements e
        Add contribution from element e to load vector component j
    end
end
```

However, by inspection we see that for the first component only elements within which the test function $N_{\langle 1 \rangle}$ is different from zero will contribute. There are at most two such elements, and therefore it makes sense not to loop over all the elements and rather reverse the order of the above loops:

```
Loop over all finite elements e (note: the nodes of the element e are K, M)
    Add contribution of element e to load vector component (K)
    Add contribution of element e to load vector component (M)
end
```

As shown in this figure from element e we compute contributions to $L_{(K)}$ and $L_{(M)}$.



For our particular mesh we start the computation of the load vector with the zero vector

$$[L] = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

For element 1 we compute the contribution to L_1 because the test function $N_{\langle 1 \rangle}$ is nonzero over this element. Thus we compute

$$\int_{x_1}^{x_2} N_{\langle 1 \rangle} q \, \mathrm{d}x = qL/4$$

which is added to the load vector

$$[L] = \begin{bmatrix} 0 + qL/4 \\ 0 \end{bmatrix}$$

There is no contribution to L_2 because $N_{\langle 2 \rangle}$ is zero over element 1. For element 2 we compute the contribution to L_1

$$\int_{x_2}^{x_3} N_{\langle 1 \rangle} q \, \mathrm{d}x = qL/4$$

and the contribution to L_2

$$\int_{x_2}^{x_3} N_{\langle 2 \rangle} q \, \mathrm{d}x = qL/4$$

which are added as

$$[L] = \begin{bmatrix} qL/4 + qL/4 \\ 0 + qL/4 \end{bmatrix}$$

The contributions to the stiffness matrix are computed in similar manner. Instead of computing the stiffness matrix component by component by looping over all the elements repeatedly, we loop over the elements just once, computing contributions to the integrals that define the stiffness matrix and adding them to the global stiffness matrix. That operation is called ***assembly***.

```
Loop over all finite elements e (note: the nodes of the element e are K, M)
    Add contribution of element e to stiffness matrix component (K)(K)
    Add contribution of element e to stiffness matrix component (K)(M)
    Add contribution of element e to stiffness matrix component (M)(K)
    Add contribution of element e to stiffness matrix component (M)(M)
end
```

For our particular mesh the global stiffness matrix starts out as a 2×2 zero matrix

$$[K] = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

From element 1 we compute the contribution to K_{11} because the test function $N_{\langle 1 \rangle}$ is nonzero over this element. Thus we compute

$$\int_{x_1}^{x_2} N_{\langle 1 \rangle}{}' P N_{\langle 1 \rangle}{}' \, \mathrm{d}x = \int_{x_1}^{x_2} N_2{}' P N_2{}' \, \mathrm{d}x = P \frac{1}{L/2} \frac{1}{L/2} L/2 = \frac{P}{L/2} = \frac{2P}{L}$$

which is added to the stiffness matrix as

$$[K] = \begin{bmatrix} \frac{2P}{L} & 0 \\ 0 & 0 \end{bmatrix}$$

From element 2 we compute the contribution to K_{11} as

$$\int_{x_1}^{x_2} N_{\langle 1 \rangle}{}' P N_{\langle 1 \rangle}{}' \, \mathrm{d}x = \int_{x_1}^{x_2} N_2{}' P N_2{}' \, \mathrm{d}x = P \frac{-1}{L/2} \frac{-1}{L/2} L/2 = \frac{P}{L/2} = \frac{2P}{L}$$

which is added to the stiffness matrix as

$$[K] = \begin{bmatrix} \frac{2P}{L} + \frac{2P}{L} & 0 \\ 0 & 0 \end{bmatrix}$$

Next from element 2 we compute the contribution to K_{12} (which is also the same as the contribution to K_{21}) as

$$\int_{x_1}^{x_2} N_{\langle 1 \rangle}{}' P N_{\langle 2 \rangle}{}' \, \mathrm{d}x = \int_{x_1}^{x_2} N_2{}' P N_3{}' \, \mathrm{d}x = P \frac{-1}{L/2} \frac{1}{L/2} L/2 = -\frac{P}{L/2} = -\frac{2P}{L}$$

which is added to the stiffness matrix for both components as

$$[K] = \begin{bmatrix} \frac{2P}{L} + \frac{2P}{L} \, , & 0 - \frac{2P}{L} \\ 0 - \frac{2P}{L} \, , & 0 \end{bmatrix}$$

Finally from element 2 we compute the contribution to K_{22} as

$$\int_{x_1}^{x_2} N_{\langle 2 \rangle}{}' P N_{\langle 2 \rangle}{}' \, \mathrm{d}x = \int_{x_1}^{x_2} N_3{}' P N_3{}' \, \mathrm{d}x = P \frac{1}{L/2} \frac{1}{L/2} L/2 = \frac{P}{L/2} = \frac{2P}{L}$$

which is added to the stiffness matrix to yield

$$[K] = \begin{bmatrix} \frac{2P}{L} + \frac{2P}{L} \, , & -\frac{2P}{L} \\ -\frac{2P}{L} \, , & 0 + \frac{2P}{L} \end{bmatrix}$$

The discrete equations of equilibrium are identical to those derived in Section 2.10, and from that point on the two solutions are identical.

2.12.1 Elementwise quantities

Now that we see the potential for the organization based on elements, we shall try to firm up these ideas. We continue by working through an example of the prestressed cable from Figure 2.8. Again we use a mesh of two L2 finite elements of equal length, but this time we will formulate the technique of element-by-element assembly of elementwise quantities (load vector and stiffness matrix).

By definition, the elementwise quantities collect the contributions from the elements to the load vector or the stiffness matrix. These contributions are computed whether or not there are unknown degrees of freedom associated with either node of the element. The equation numbers are taken into account when assembling the elementwise quantities into the global load vector or global stiffness matrix.

As a visual mnemonic device imagine a particular element as a finite element mesh: a single element is the mesh. Observe Figure 2.10 where at the top we have a mesh consisting of four finite

elements, and for one such element e we show the basis functions that are nonzero within this element. At the bottom we show this same element e separated out from the mesh. There are still the same two basis functions nonzero within this element. This element is now all alone in the mesh, and this mesh has only two degrees of freedom which we number without any consideration of boundary conditions: degree of freedom 1 at node K, and degree of freedom 2 at node M. Note that this numbering of the degrees of freedom has nothing to do with the original mesh, it is valid only for our mnemonic device and it is purely imaginary.

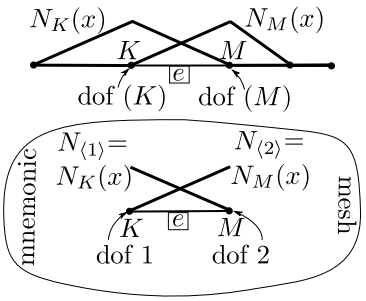


Fig. 2.10. Schema of the element e and the basis functions that are nonzero within the element.

With this setup we start computing. There are two test functions, $N_K = N_{\langle 1\rangle}$ and $N_M = N_{\langle 2\rangle}$. The trial function for the mnemonic mesh has the form

$$w(x) = N_{\langle 1\rangle} w_1 + N_{\langle 2\rangle} w_2$$

First the load vector: substituting into the weighted residual equation written for our single-element mesh we obtain

$$L_1^{(e)} = \int_{x_K}^{x_M} N_{\langle 1\rangle} q \, \mathrm{d}x = \int_{x_K}^{x_M} N_K q \, \mathrm{d}x$$

and

$$L_2^{(e)} = \int_{x_K}^{x_M} N_{\langle 2\rangle} q \, \mathrm{d}x = \int_{x_K}^{x_M} N_M q \, \mathrm{d}x$$

Therefore, we can write the so-called elementwise load vector as

$$[L]^{(e)} = \begin{bmatrix} L_1^{(e)} \\ L_2^{(e)} \end{bmatrix} = \begin{bmatrix} \displaystyle\int_{x_K}^{x_M} N_K(x) q \, \mathrm{d}x \\ \displaystyle\int_{x_K}^{x_M} N_M(x) q \, \mathrm{d}x \end{bmatrix}$$

For a uniform distributed load we get simply

$$[L]^{(e)} = \begin{bmatrix} q(x_M - x_K)/2 \\ q(x_M - x_K)/2 \end{bmatrix} \tag{2.29}$$

Now that we have computed the elementwise load vector what do we do with it? We take a dose of reality and note that the degrees of freedom at nodes K, M are really not 1 and 2, but (K) and (M). Therefore the components of the elementwise load vector are in fact contributions to $L_{(K)}$ and $L_{(M)}$. To assemble the elementwise load vectors we use the so-called ***element equation arrays***. They consist of the equation numbers associated to each of the nodes of the element. The elementwise load vector components are assembled to the global components according to the equation numbers. For element e the element equation array is

$$\begin{bmatrix} (K) \\ (M) \end{bmatrix}$$

The assembly operation simply adds the components of the elementwise load vector to the components of the global load vector as

$$L_{(K)} \leftarrow L_{(K)} + L_1^{(e)} \ , \quad L_{(M)} \leftarrow L_{(M)} + L_2^{(e)} \ ,$$

For the mesh used in our example in Section 2.10 and uniform q, for element 1 we compute the elementwise load vector as

$$[L]^{(1)} = \begin{bmatrix} qL/4 \\ qL/4 \end{bmatrix}$$

and for element 2 we compute the elementwise load vector as

$$[L]^{(2)} = \begin{bmatrix} qL/4 \\ qL/4 \end{bmatrix}$$

The two elementwise load vectors are the same because the load is the same and the lengths of the elements are the same.

For element 1 the element equation array is

$$\begin{bmatrix} (1) \\ (2) \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

Therefore the elementwise load vector for element 1 will be assembled as

$$[L]^{(1)} = \begin{bmatrix} L_1^{(1)} \\ L_2^{(1)} \end{bmatrix} \begin{matrix} \rightarrow \text{to row } 3 \\ \rightarrow \text{to row } 1, \text{that is to global component } L_1 \end{matrix}$$

Note that w_3 is known (prescribed at zero), and hence the contribution to equation 3 is ignored during the assembly.

For element 2 the element equation array is

$$\begin{bmatrix} (2) \\ (3) \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

The elementwise load vector for element 2 will be assembled as

$$[L]^{(2)} = \begin{bmatrix} L_1^{(2)} \\ L_2^{(2)} \end{bmatrix} \begin{matrix} \rightarrow \text{to row } 1, \text{that is to global component } L_1 \\ \rightarrow \text{to row } 2, \text{that is to global component } L_2 \end{matrix}$$

The global load vector therefore results as

$$[L] = \begin{bmatrix} qL/4 + qL/4 \\ qL/4 \end{bmatrix}$$

The contributions to the stiffness matrix are computed in similar manner. For instance

$$K_{11}^{(e)} = \int_{x_K}^{x_M} N'_{\langle 1 \rangle} P N'_{\langle 1 \rangle} \, \mathrm{d}x$$

so that we get for the single-element mesh the stiffness matrix

$$[K]^{(e)} = \begin{bmatrix} \displaystyle\int_{x_K}^{x_M} N'_{\langle 1 \rangle} P N'_{\langle 1 \rangle} \, \mathrm{d}x & \displaystyle\int_{x_K}^{x_M} N'_{\langle 1 \rangle} P N'_{\langle 2 \rangle} \, \mathrm{d}x \\ \displaystyle\int_{x_K}^{x_M} N'_{\langle 2 \rangle} P N'_{\langle 1 \rangle} \, \mathrm{d}x & \displaystyle\int_{x_K}^{x_M} N'_{\langle 2 \rangle} P N'_{\langle 2 \rangle} \, \mathrm{d}x \end{bmatrix}$$

which means explicitly

$$[K]^{(e)} = \begin{bmatrix} \int_{x_K}^{x_M} N_K' P N_K' \, \mathrm{d}x & \int_{x_K}^{x_M} N_K' P N_M' \, \mathrm{d}x \\ \int_{x_K}^{x_M} N_M' P N_K' \, \mathrm{d}x & \int_{x_K}^{x_M} N_M' P N_M' \, \mathrm{d}x \end{bmatrix}$$

Since we have

$$N_K' = -\frac{1}{x_M - x_K}\,, \quad N_M' = +\frac{1}{x_M - x_K}\,, \quad \int_{x_K}^{x_M} \mathrm{d}x = x_M - x_K$$

we obtain

$$\int_{x_K}^{x_M} N_K' P N_M' \, \mathrm{d}x = \left(-\frac{1}{x_M - x_K}\right) P \frac{1}{x_M - x_K} \int_{x_K}^{x_M} \mathrm{d}x = -\frac{P}{x_M - x_K}$$

and so on for the other elements of the stiffness matrix. Hence the ***elementwise stiffness matrix*** for the L2 finite element is

$$[K]^{(e)} = \frac{P}{x_M - x_K} \begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} \tag{2.30}$$

For the finite element 1 it is assembled using the element equation arrays as

$$[K]^{(1)} = \frac{P}{x_M - x_K} \begin{matrix} \text{to column 3} \\ \downarrow \\ \end{matrix} \begin{matrix} \text{to column 1} \\ \downarrow \\ \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} \rightarrow \text{to row 3} \\ \rightarrow \text{to row 1} \end{matrix}$$

where $x_M - x_K = L/2$. For the finite element 2 the elementwise stiffness matrix is assembled using the element equation arrays as

$$[K]^{(2)} = \frac{P}{x_M - x_K} \begin{matrix} \text{to column 1} \\ \downarrow \\ \end{matrix} \begin{matrix} \text{to column 2} \\ \downarrow \\ \end{matrix} \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix} \begin{matrix} \rightarrow \text{to row 1} \\ \rightarrow \text{to row 2} \end{matrix}$$

The global stiffness matrix therefore results as

$$[K] = \begin{bmatrix} \frac{2P}{L} + \frac{2P}{L}\,, & -\frac{2P}{L} \\ -\frac{2P}{L}\,, & \frac{2P}{L} \end{bmatrix}$$

The global load and stiffness matrices are the same as in Section 2.10, equation (2.25). The solution for the unknown degrees of freedom and the resulting approximation would be exactly the same here.

2.13 Prescribed displacements

In this section we consider a pin-roller prestressed cable from Figure 2.11. The boundary conditions consist of prescribed displacement $w(0)$ at the left-hand side support, zero natural boundary condition force at the right-hand side; the transverse load q is zero. The weighted residual statement (2.27) reduces for this case to

$$-\sum_{i=1}^{N_\mathrm{f}} \left(\int_0^L N_{\langle j \rangle}{}' P N_{\langle i \rangle}{}' \, \mathrm{d}x \right) w_i - \sum_{i=N_\mathrm{f}+1}^{N} \left(\int_0^L N_{\langle j \rangle}{}' P N_{\langle i \rangle}{}' \, \mathrm{d}x \right) w_i = 0, \qquad j = 1, ..., N_\mathrm{f}$$

where $N = 3$ and $N_\mathrm{f} = 2$. Given the mesh from Figure 2.11, the test functions are the basis functions at the nodes with the unknowns, $N_{\langle 1\rangle} = N_2$ and $N_{\langle 2\rangle} = N_3$. The trial function is

$$w(x) = N_{\langle 1\rangle}(x)w_1 + N_{\langle 2\rangle}(x)w_2 + N_{\langle 3\rangle}(x)w_3$$

as in Section 2.10, except that this time $w_3 \neq 0$ and it must be carried in the equation. Therefore, we can fully rewrite the above as

$$\begin{aligned}
-\int_0^L N_{\langle 1\rangle}{}' P N_{\langle 1\rangle}{}' \,\mathrm{d}x\, w_1 - \int_0^L N_{\langle 1\rangle}{}' P N_{\langle 2\rangle}{}' \,\mathrm{d}x\, w_2 - \int_0^L N_{\langle 1\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x\, w_3 = 0 \\
-\int_0^L N_{\langle 2\rangle}{}' P N_{\langle 1\rangle}{}' \,\mathrm{d}x\, w_1 - \int_0^L N_{\langle 2\rangle}{}' P N_{\langle 2\rangle}{}' \,\mathrm{d}x\, w_2 - \int_0^L N_{\langle 2\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x\, w_3 = 0
\end{aligned}$$

Realizing that w_3 is known, we may rearrange the terms so that everything on the right-hand side is known

$$\begin{aligned}
+\int_0^L N_{\langle 1\rangle}{}' P N_{\langle 1\rangle}{}' \,\mathrm{d}x\, w_1 + \int_0^L N_{\langle 1\rangle}{}' P N_{\langle 2\rangle}{}' \,\mathrm{d}x\, w_2 = -\int_0^L N_{\langle 1\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x\, w_3 \\
+\int_0^L N_{\langle 2\rangle}{}' P N_{\langle 1\rangle}{}' \,\mathrm{d}x\, w_1 + \int_0^L N_{\langle 2\rangle}{}' P N_{\langle 2\rangle}{}' \,\mathrm{d}x\, w_2 = -\int_0^L N_{\langle 2\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x\, w_3
\end{aligned}$$

On the left we have the stiffness matrix (see equation (2.25)) multiplied by the free displacements, as in Section 2.10, and on the right we have a load vector which accounts for the prescribed displacement at the pin. The coefficients on the right-hand side are easily evaluated as

$$K_{13} = \int_0^L N_{\langle 1\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x = \frac{-1}{\frac{L}{2}} P \frac{1}{\frac{L}{2}} \frac{L}{2} = -\frac{2P}{L}$$

and

$$K_{23} = \int_0^L N_{\langle 2\rangle}{}' P N_{\langle 3\rangle}{}' \,\mathrm{d}x = 0$$

The load vector is therefore

$$[L] = -\begin{bmatrix} K_{13} \\ K_{23} \end{bmatrix} w_3 = -\begin{bmatrix} -\dfrac{2P}{L} \\ 0 \end{bmatrix} w_3$$

We can see that the solution is $w_1 = w_3 = 0.2$ and $w_2 = w_3 = 0.2$: the entire cable just slides downwards by w_3 as a rigid body.

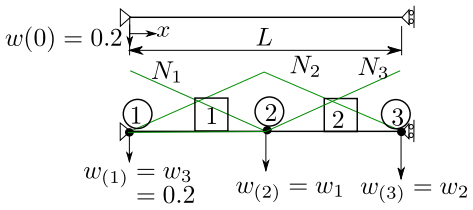


Fig. 2.11. Prestressed cable configuration with prescribed displacement

The prescribed-displacement load can be also treated by elementwise quantities: see Section 2.12.1. The elementwise prescribed-displacement load due to displacement w_K of node K

$$[L]^{(e)} = -\begin{bmatrix} \int_{x_K}^{x_M} N'_K P N'_K \, \mathrm{d}x \\ \int_{x_K}^{x_M} N'_M P N'_K \, \mathrm{d}x \end{bmatrix} w_K$$

which can be evaluated for the piecewise linear basis functions as

$$[L]^{(e)} = -\frac{P}{x_M - x_K} \begin{bmatrix} 1 \\ -1 \end{bmatrix} w_K$$

The elementwise prescribed-displacement load due to displacement w_M of node M

$$[L]^{(e)} = -\begin{bmatrix} \int_{x_K}^{x_M} N'_K P N'_M \, \mathrm{d}x \\ \int_{x_K}^{x_M} N'_M P N'_M \, \mathrm{d}x \end{bmatrix} w_M$$

which can be evaluated for the piecewise linear basis functions as

$$[L]^{(e)} = -\frac{P}{x_M - x_K} \begin{bmatrix} -1 \\ 1 \end{bmatrix} w_M$$

The elementwise load vector is assembled as discussed before:

$$[L]^{(e)} = \begin{bmatrix} L_1^{(e)} \\ L_2^{(e)} \end{bmatrix} \begin{matrix} \rightarrow \text{to row } (K) \\ \rightarrow \text{to row } (M) \end{matrix}$$

Exercise 15.

Solve for the deflection of the prestressed cable of Figure 2.11 again, this time using the elementwise stiffness matrices and load vectors. Use a mesh of two L2 finite elements of equal length.

Solution: The elementwise stiffness matrix

$$[K]^{(e)} = \frac{P}{x_M - x_K} \begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix}$$

is assembled from each of the two elements to yield the stiffness matrix (2.25). The elementwise prescribed-displacement load vector from element 1 is nonzero because on element 1 $w_K = w_3 \neq 0$ is prescribed.

$$[L]^{(1)} = -\frac{P}{L/2} \begin{bmatrix} 1 \\ -1 \end{bmatrix} w_3$$

For element 1 the element equation array is

$$\begin{bmatrix} (1) \\ (2) \end{bmatrix} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}$$

and therefore we assemble $-\frac{P}{L/2} \times (-1) \times w_3$ to L_1. For element 2 we don't get to assemble any load vector since at neither node of that element the deflections are known. As a result, the right-hand side load vector is identical to the one derived previously.

2.14 Partitioned form

We will change the rules of the game, temporarily, by allowing reactions to make their appearance in the weighted residual statement. Equation (2.13) will be therefore kept as is and we will not require that the test function becomes zero on the boundary with essential boundary conditions (i.e. $\eta_j(x=0)=0$ is not required)

$$-\eta_j(0)Pw'(0) - \int_0^L \eta_j{}'Pw' \,\mathrm{d}x + \int_0^L \eta_j q \,\mathrm{d}x + \eta_j(L)F_L = 0\,, \tag{2.31}$$

Specializing the above to the finite element setting we modify (2.26) to read

$$N_{\langle j\rangle}(0)R_0 - \sum_{i=1}^{N}\left(\int_0^L N_{\langle j\rangle}{}'PN_{\langle i\rangle}{}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x + N_{\langle j\rangle}(L)F_L = 0, \quad j=1,...,N, \tag{2.32}$$

where we have introduced the definition of the reaction $R_0 = -Pw'(0)$. Note that the test functions can be *all* the finite element basis functions ($j=1,...,N$). Since we haven't eliminated the reaction R_0 by the design of the test functions, it will be present in the resulting equations as an additional unknown. Here we illustrate this state of affairs by an example.

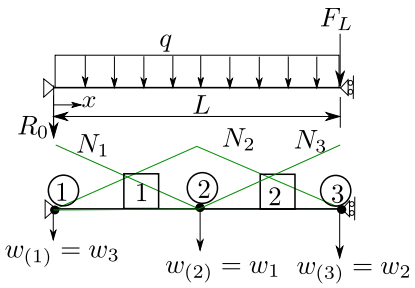


Fig. 2.12. Configuration of the cable for the partitioned analysis.

We will solve for the deflection of the prestressed cable from Figure 2.12 using the finite element Galerkin method with a mesh of two L2 finite elements of equal length. We will assume that the transverse load q, the transverse force F_L, the deflection that the pin $w(0) \neq 0$, and the pre-stressing force P are given, but we shall consider these quantities as variables.

The Galerkin weighted residual statement for the BVP for the static deflection of the pin-roller cable is written as shown in equation (2.31)

$$N_{\langle j\rangle}(0)R_0 - \sum_{i=1}^{N}\left(\int_0^L N_{\langle j\rangle}{}'PN_{\langle i\rangle}{}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x + N_{\langle j\rangle}(L)F_L = 0, \quad j=1,2,3\,,$$

where we require for the trial function

$$w(x) = \sum_{k=1}^{3} N_k(x) w_{(k)}$$

that the essential boundary condition be satisfied

$$w(x=0) = \bar{w}_0.$$

Also, we have used the definition

$$R_0 = -Pw'(0)$$

of the reaction at the pin. As before, the essential boundary condition determines the value of the prescribed degree of freedom at the pin $w_3 = \bar{w}_0$. The unknowns are w_1, w_2 but also the reaction R_0.

The mesh consists of two L2 elements

Element	Nodes
1	1,2
2	2,3

The degrees of freedom will be numbered starting with the unknowns (deflections at nodes 2, 3), and concluding with the deflection at node 1 which is known.

Node	Degree of freedom (equation) number
1	3
2	1
3	2

The elements of the load vector L_j are now computed for $j = 1, 2, 3$. We begin with the distributed transverse load. The elementwise load vector

$$[L]^{(e)} = \begin{bmatrix} qL/4 \\ qL/4 \end{bmatrix}$$

gets assembled according to the element equation arrays: element 1 (3,1), and element 2 (1,2). The partial result for the global load vector will be therefore

$$[L] = \begin{bmatrix} qL/4 + qL/4 \\ qL/4 \\ qL/4 \end{bmatrix}$$

Next the concentrated forces are assembled. Where do we put R_0? This is determined by which test function is nonzero at $x = 0$: we can see immediately that only $N_1(0) = 1$, all the other basis functions are zero at this location, and therefore R_0 is added to $L_{(1)} = L_3$ (recall that node 1 is associated with degree of freedom 3). Where do we put F_L? This is determined by which test function is nonzero at $x = L$: we can see immediately that only $N_3(L) = 1$, all the other basis functions are zero at this location, and therefore F_L is added to $L_{(3)} = L_2$. The partial result for the global load vector is therefore

$$[L] = \begin{bmatrix} qL/2 \\ qL/4 + F_L \\ qL/4 + R_0 \end{bmatrix}$$

The stiffness matrix is computed next. The elementwise stiffness matrix (2.30) is assembled twice using the above equation arrays resulting in

$$[K] = \begin{bmatrix} \frac{2P}{L} + \frac{2P}{L} , & -\frac{2P}{L} , & -\frac{2P}{L} \\ -\frac{2P}{L} , & \frac{2P}{L} , & 0 \\ -\frac{2P}{L} , & 0 , & \frac{2P}{L} \end{bmatrix} = \frac{2P}{L} \begin{bmatrix} 2 , & -1 , & -1 \\ -1 , & 1 , & 0 \\ -1 , & 0 , & 1 \end{bmatrix} \tag{2.33}$$

The discrete equations of equilibrium are therefore written as

$$[K][w] = [L]$$

or explicitly

$$\frac{2P}{L}\begin{bmatrix} 2\,, & -1\,, & -1 \\ -1\,, & 1\,, & 0 \\ -1\,, & 0\,, & 1 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} qL/2 \\ qL/4 + F_L \\ qL/4 + R_0 \end{bmatrix}$$

This is not a simple system of linear algebraic equations that can be solved by the inversion of the coefficient matrix yet. For one thing, w_3 on the left-hand side is known, while R_0 on the right-hand side is unknown. Because of the way in which we numbered the unknowns, we can solve this system by partitioning. We will split the matrices and vectors by cutting between the second and third column on the left and between the second and third row on both sides as

$$\frac{2P}{L}\begin{bmatrix} 2\,, & -1 \\ -1\,, & 1 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \frac{2P}{L}\begin{bmatrix} -1 \\ 0 \end{bmatrix} w_3 = \begin{bmatrix} qL/2 \\ qL/4 + F_L \end{bmatrix}$$

and

$$\frac{2P}{L}\begin{bmatrix} -1\,, & 0 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \frac{2P}{L}\begin{bmatrix} 1 \end{bmatrix} w_3 = \begin{bmatrix} qL/4 + R_0 \end{bmatrix}$$

Visually comparing these two equations with the unpartitioned original matrix equation we can see that nothing changed. However, the system is now unraveled by first solving for w_1, w_2 from the first partition

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{L}{2P}\begin{bmatrix} 2\,, & -1 \\ -1\,, & 1 \end{bmatrix}^{-1}\left\{-\frac{2P}{L}\begin{bmatrix} -1 \\ 0 \end{bmatrix} w_3 + \begin{bmatrix} qL/2 \\ qL/4 + F_L \end{bmatrix}\right\}$$

and with w_1, w_2 at hand we can solve from the second partition for R_0 as

$$\begin{bmatrix} R_0 \end{bmatrix} = \frac{2P}{L}\begin{bmatrix} -1\,, & 0 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + \frac{2P}{L}\begin{bmatrix} 1 \end{bmatrix} w_3 - \begin{bmatrix} qL/4 \end{bmatrix}$$

2.14.1 Derivation of the partitioned form

When the form of the weighted residual equation (2.31) is used, the resulting equations contained in addition to the unknown displacements at the nodes and the known displacements at the supports also the unknown reactions at the supports. Consequently, one needs as many equations as there are nodes, and all basis functions defined on the mesh are therefore used to derive the discrete equations. The resulting system of equations is written as

$$[K][w] = [L]$$

where the dimension of the stiffness matrix is $N \times N$, N being the total number of nodes. Let us call N_{d} the number of prescribed displacements, and N_{f} the number of unknown degrees of freedom. Evidently we have $N = N_{\mathrm{d}} + N_{\mathrm{f}}$. We shall use the convention of numbering first the unknown degrees of freedom, and only then the prescribed degrees of freedom. Therefore, we can write the above system of equations in the partitioned form

$$\begin{bmatrix} [K_{\mathrm{ff}}],\ [K_{\mathrm{fd}}] \\ [K_{\mathrm{df}}],\ [K_{\mathrm{dd}}] \end{bmatrix}\begin{bmatrix} [w_{\mathrm{f}}] \\ [w_{\mathrm{d}}] \end{bmatrix} = \begin{bmatrix} [L_{\mathrm{f}}] \\ [L_{\mathrm{d}}] + [R] \end{bmatrix}$$

Here $[K_{\mathrm{ff}}]$ is the stiffness matrix that links the displacements at the free degrees of freedom $[w_{\mathrm{f}}]$ to the forces acting on the nodes with the free degrees of freedom; $[K_{\mathrm{fd}}]$ is the stiffness matrix that links the displacements at the degrees of freedom that are prescribed $[w_{\mathrm{d}}]$ to the forces acting on the nodes with the free degrees of freedom; $[K_{\mathrm{df}}]$ is the stiffness matrix that links the displacements at

the free degrees of freedom $[w_\mathrm{f}]$ to the forces acting on the nodes with supports (i.e. with prescribed degrees of freedom); and $[K_\mathrm{dd}]$ is the stiffness matrix that links the prescribed displacements $[w_\mathrm{d}]$ to the forces acting on the nodes with supports. Further $[L_\mathrm{f}]$ are the applied loads acting on the nodes where displacement is unknown, and $[L_\mathrm{d}]$ are applied loads that are directly transferred into the supports. Finally, note that we have also included the reactions of the supports $[R]$. We can think of this system of equations as describing the equilibrium of a free body: all supports were replaced with corresponding reactions.

$$[K_\mathrm{ff}][w_\mathrm{f}] + [K_\mathrm{fd}][w_\mathrm{d}] = [L_f]$$
$$[K_\mathrm{df}][w_\mathrm{f}] + [K_\mathrm{dd}][w_\mathrm{d}] = [L_\mathrm{d}] + [R]$$

In the first step we solve for the free deflections

$$[K_\mathrm{ff}][w_\mathrm{f}] = [L_f] - [K_\mathrm{fd}][w_\mathrm{d}]$$

where the first term on the right-hand side collects the applied loads due to the transverse distributed load and any applied concentrated forces, and the second term are the prescribed-displacement loads.

The computed free displacements are then inserted into the expression

$$[R] = [K_\mathrm{df}][w_\mathrm{f}] + [K_\mathrm{dd}][w_\mathrm{d}] - [L_\mathrm{d}]$$

which solves for the reactions.

2.15 Principle of superposition

In this book we treat only the so-called linear problems. In the end they all lead to a solution of linear algebraic equations, or to a solution of linear ordinary differential equations with constant coefficient matrices. For simplicity the principle of superposition will be demonstrated first for the linear-algebraic equation case.

Consider one of the models treated previously. The equations to be solved could be summarized as

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$$

where $\boldsymbol{A}$ is the coefficient (stiffness) matrix, $\boldsymbol{b}$ is the load vector, and $\boldsymbol{x}$ is the solution. The principle of superposition essentially states that the load vector may be arbitrarily decomposed

$$\boldsymbol{b} = \boldsymbol{b}_1 + \boldsymbol{b}_2$$

and the solution will therefore be composed of

$$\boldsymbol{x} = \boldsymbol{x}_1 + \boldsymbol{x}_2$$

where

$$\boldsymbol{A}\boldsymbol{x}_1 = \boldsymbol{b}_1 \ , \quad \boldsymbol{A}\boldsymbol{x}_2 = \boldsymbol{b}_2$$

This is often used to advantage in static analyses of structures where the structure is analyzed for a few load types (live and dead load, wind load, temperature load, support motion,...) $\boldsymbol{b}_j$, resulting in the response vectors $\boldsymbol{x}_j$, and the responses are then combined in various proportions to evaluate the effect of multiple (typically many more than there are load types) combinations of loads. For instance, with λ_1, λ_2 being load multipliers we can obtain the response to loads $\boldsymbol{b} = \lambda_1\boldsymbol{b}_1 + \lambda_2\boldsymbol{b}_2$ as $\boldsymbol{x} = \lambda_1\boldsymbol{x}_1 + \lambda_2\boldsymbol{x}_2$.

The Galerkin method leads to a set of coupled ordinary differential equations (IVP), which can always be written in the first order form

$$\boldsymbol{B}\dot{\boldsymbol{y}} = \boldsymbol{L} \ , \quad \boldsymbol{y}(0) = \boldsymbol{y}_0$$

Similarly to the static case, in the dynamic case we can split the forcing and compute the responses separately

$$\boldsymbol{B}\dot{\boldsymbol{y}}_1 = \boldsymbol{L}_1 \ , \quad \boldsymbol{y}_1(0) = \boldsymbol{y}_{0,1} \ , \quad \boldsymbol{B}\dot{\boldsymbol{y}}_2 = \boldsymbol{L}_2 \ , \quad \boldsymbol{y}_2(0) = \boldsymbol{y}_{0,2}$$

so that the overall response may be obtained as the combination

$$\boldsymbol{B}(\dot{\boldsymbol{y}}_1 + \dot{\boldsymbol{y}}_2) = \boldsymbol{L}_1 + \boldsymbol{L}_2 \ , \quad \boldsymbol{y}_1(0) + \boldsymbol{y}_2(0) = \boldsymbol{y}_{0,1} + \boldsymbol{y}_{0,2}$$

3

Taut wire dynamics with the Galerkin method

We continue working with the prestressed cable model from Figure 1.1. In this chapter we will revise the results of the previous chapter to incorporate dynamics.

3.1 Residual of the balance equation

For the dynamical equations of motion the starting point is the balance equation (1.1)

$$Pw''(x,t) + q(x,t) = \mu\ddot{w}(x,t) \ , \tag{3.1}$$

where we have explicitly indicated the arguments. Furthermore we have the boundary conditions

$$w(0,t) = \bar{w}_0(t) \ . \tag{3.2}$$

where the deflection now depends on time. The force boundary condition may also be time dependent

$$-Pw'(L,t) + F_L(t) = 0 \ . \tag{3.3}$$

The balance equation (3.1) may be written in the residual form as

$$Pw''(x,t) + q(x,t) - \mu\ddot{w}(x,t) = r_B(x,t) \ . \tag{3.4}$$

The residual r_B is identically zero if w is the exact solution. For an approximate solution, the residual r_B varies from point to point and from time to time, and is in general nonzero.

3.2 Integral test of the residual

The same procedure that was adopted for the statics case in the previous chapter will now be applied here. Note that the test function $\eta(x)$ does not depend on time: it describes only shape that is unchanging in time

$$\int_0^L \eta(x) r_B(x,t) \ \mathrm{d}x \ . \tag{3.5}$$

Evidently, we will be taking the test function from among the basis functions defined on a finite element mesh as before

$$\int_0^L N_{\langle j \rangle}(x) r_B(x,t) \ \mathrm{d}x \ , \quad j = 1, ..., N_\mathrm{f} \ .$$

The trial function will be the finite element expansion in the form

$$w(x,t) = \sum_{k=1}^{N} N_{\langle k \rangle}(x) w_k(t) \tag{3.6}$$

where we explicitly indicate that the trial function accepts both x and t as arguments, but they get separated in the finite element expansion since the basis functions depend only x on whereas the degrees of freedom are only functions of time t. Consequently, when we carry out the derivatives in the trial function we obtain

$$w''(x,t) = \sum_{k=1}^{N} N''_{\langle k \rangle}(x) w_k(t)$$

and

$$\ddot{w}(x,t) = \sum_{k=1}^{N} N_{\langle k \rangle}(x) \ddot{w}_k(t)$$

3.3 Weighted residual manipulations

We will handle the essential boundary condition as for the statics case by designing the trial function to satisfy these conditions identically. For instance, if the boundary condition requires

$$w(0,t) = \bar{w}_0(t)$$

we obtain by substituting

$$w(0,t) = \sum_{k=1}^{N} N_{\langle k \rangle}(0) w_k(t) = \bar{w}_0(t)$$

Let us say node 1 is at $x = 0$, then this condition will mean

$$w(0,t) = \sum_{k=1}^{N} N_{\langle k \rangle}(0) w_k(t) = \sum_{k=1}^{N} N_k(0) w_{(k)}(t) = w_{(1)}(t) = \bar{w}_0(t)$$

so that the degree of freedom $w_{(1)}$ (deflection) is prescribed as a function of time.

Concerning the balance residual: As for the statics case, we have to do something about the term $w''(x,t)$ because the finite element basis functions cannot be differentiated twice. The solution in the previous chapter was integration by parts on the term Pw'', and such solution is certainly still applicable here. Also, the combination of the balance residual and the natural boundary condition residual still makes sense here. Therefore, carrying out both of these operations will yield the following slight modification of the weighted residual equation (2.13)

$$\begin{aligned} &\int_0^L \eta(x) r_B(x,t)\, \mathrm{d}x + \eta(L) r_F(t) = \\ &-\eta(0) P w'(0,t) - \int_0^L \eta'(x) P w'(x,t)\, \mathrm{d}x + \int_0^L \eta(x) q(x,t)\, \mathrm{d}x + \eta(L) F_L(t) \\ &- \int_0^L \eta(x) \mu \ddot{w}(x,t)\, \mathrm{d}x = 0\ , \end{aligned} \tag{3.7}$$

Here we will also use the trick of setting the test function to zero at the boundary where the reaction $(-Pw'(0))$ is unknown so that we get

$$\begin{aligned} &\int_0^L \eta(x) r_B(x,t)\, \mathrm{d}x + \eta(L) r_F(t) = \\ &-\int_0^L \eta'(x) P w'(x,t)\, \mathrm{d}x + \int_0^L \eta(x) q(x,t)\, \mathrm{d}x + \eta(L) F_L(t) - \int_0^L \eta(x) \mu \ddot{w}(x,t)\, \mathrm{d}x = 0\ , \end{aligned} \tag{3.8}$$

where we require $\eta(0) = 0$.

At this point the only changes with respect to the statics case are the presence of time in these expressions, and the additional term

$$\int_0^L \eta(x)\mu\ddot{w}(x,t)\,\mathrm{d}x$$

As we will show next, this term will lead to a mass matrix and an inertial force.

3.4 Mass matrix and load vector

The trial function will be the finite element expansion (3.6). The test functions will be the finite element basis functions that satisfy the condition

$$\eta(0) = N_{\langle k\rangle}(0) = 0$$

The trial and test functions give upon substitution into the weighted residual equation (3.8)

$$\begin{aligned} &-\int_0^L N_{\langle j\rangle}{}'(x)P\sum_{i=1}^{N} N_{\langle i\rangle}{}'(x)w_i(t)\,\mathrm{d}x + \int_0^L N_{\langle j\rangle}(x)q(x,t)\,\mathrm{d}x + N_{\langle j\rangle}(L)F_L(t) \\ &-\int_0^L N_{\langle j\rangle}(x)\mu\sum_{i=1}^{N} N_{\langle i\rangle}(x)\ddot{w}_i(t)\,\mathrm{d}x = 0\ , \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } j = 1,...,N_{\mathrm{f}}. \end{aligned} \tag{3.9}$$

which may be rewritten as

$$\begin{aligned} &-\sum_{i=1}^{N}\left(\int_0^L N_{\langle j\rangle}{}'(x)PN_{\langle i\rangle}{}'(x)\,\mathrm{d}x\right)w_i(t) + \int_0^L N_{\langle j\rangle}(x)q(x,t)\,\mathrm{d}x + N_{\langle j\rangle}(L)F_L(t) \\ &-\sum_{i=1}^{N}\left(\int_0^L N_{\langle j\rangle}(x)\mu N_{\langle i\rangle}(x)\,\mathrm{d}x\right)\ddot{w}_i(t) = 0\ , \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{for } j = 1,...,N_{\mathrm{f}}. \end{aligned} \tag{3.10}$$

With the definitions of the global stiffness matrix components

$$K_{ji} = \int_0^L N_{\langle j\rangle}{}'(x)PN_{\langle i\rangle}{}'(x)\,\mathrm{d}x\ \ ,\quad \text{for } j = 1,...,N_{\mathrm{f}},\ i = 1,...,N, \tag{3.11}$$

and the global load vector components

$$L_j(t) = \int_0^L N_{\langle j\rangle}(x)q(x,t)\,\mathrm{d}x + N_{\langle j\rangle}(L)F_L(t)\ \ ,\quad \text{for } j = 1,...,N_{\mathrm{f}}, \tag{3.12}$$

and the global mass matrix components

$$M_{ji} = \int_0^L N_{\langle j\rangle}(x)\mu N_{\langle i\rangle}(x)\,\mathrm{d}x\ \ ,\quad \text{for } j = 1,...,N_{\mathrm{f}},\ i = 1,...,N, \tag{3.13}$$

we may write (3.10) in the form

$$\sum_{i=1}^{N} M_{ji}\ddot{w}_i(t) + \sum_{i=1}^{N} K_{ji}w_i(t) = L_j(t),\quad j = 1,...,N_{\mathrm{f}} \tag{3.14}$$

that makes it clear we have converted the original BVP to a linear-algebra problem of a system of coupled linear differential equations. The matrices are rectangular, because the number of rows

is the number of free degrees of freedom, whereas the number of columns is the total number of degrees of freedom. We can switch to a slightly more common form with a square stiffness matrix and square mass matrix by writing

$$\sum_{i=1}^{N_\mathrm{f}} M_{ji}\ddot{w}_i(t) + \sum_{i=1}^{N_\mathrm{f}} K_{ji}w_i(t) = L_j(t) - \sum_{i=N_\mathrm{f}+1}^{N} M_{ji}\ddot{w}_i(t) - \sum_{i=N_\mathrm{f}+1}^{N} K_{ji}w_i(t), \quad j = 1, ..., N_\mathrm{f} \quad (3.15)$$

where the terms on the right-hand side account for the so-called prescribed-displacement loads (loads due to prescribed motion of the supports).

In order for the above system to be solvable, we must supply the so-called initial conditions. The system (3.14) will then be classified as an initial value problem (IVP). Before we discuss some solution methods, we will tie up a few loose ends. The first one concerns the mass matrix.

3.5 Elementwise mass matrix

Yet again we get to use our a visual mnemonic device of a single-element mesh (Figure 2.10). Note this mesh has only two degrees of freedom which we number without any consideration of boundary conditions: degree of freedom 1 at node K, and degree of freedom 2 at node M.

There are two test functions, $N_K = N_{\langle 1\rangle}$ and $N_M = N_{\langle 2\rangle}$. The trial function for the mnemonic mesh has the form

$$w(x,t) = N_{\langle 1\rangle}(x)w_1(t) + N_{\langle 2\rangle}(x)w_2(t)$$

which we substitute into the mass matrix definition (3.13). For instance we compute the elementwise mass matrix component

$$M_{11}^{(e)} = \int_{x_K}^{x_M} N_{\langle 1\rangle}\mu N_{\langle 1\rangle}\,\mathrm{d}x = \int_{x_K}^{x_M} N_K\mu N_K\,\mathrm{d}x$$

For simplicity, let us assume that the mass density (mass per unit length) is uniform along the cable. Then the above component evaluates to

$$M_{11}^{(e)} = \int_{x_K}^{x_M} N_K\mu N_K\,\mathrm{d}x = \mu\int_{x_K}^{x_M} N_K^2\,\mathrm{d}x = \mu\int_{x_K}^{x_M}\left(\frac{(x-x_M)}{(x_K-x_M)}\right)^2\,\mathrm{d}x = \frac{\mu(x_M-x_K)}{3}$$

Similarly we obtain the general expression

$$M_{12}^{(e)} = \int_{x_K}^{x_M} N_{\langle 1\rangle}\mu N_{\langle 2\rangle}\,\mathrm{d}x = \int_{x_K}^{x_M} N_K\mu N_M\,\mathrm{d}x = M_{21}^{(e)}$$

which for uniform mass density simplifies to

$$M_{12}^{(e)} = \int_{x_K}^{x_M} N_K\mu N_M\,\mathrm{d}x = \mu\int_{x_K}^{x_M} N_K N_M\,\mathrm{d}x = \mu\int_{x_K}^{x_M}\frac{(x-x_M)}{(x_K-x_M)}\frac{(x-x_K)}{(x_M-x_K)}\,\mathrm{d}x = \frac{\mu(x_M-x_K)}{6}$$

To summarize, we get for the single-element mesh the mass matrix

$$[M]^{(e)} = \begin{bmatrix} \displaystyle\int_{x_K}^{x_M} N_{\langle 1\rangle}\mu N_{\langle 1\rangle}\,\mathrm{d}x & \displaystyle\int_{x_K}^{x_M} N_{\langle 1\rangle}\mu N_{\langle 2\rangle}\,\mathrm{d}x \\ \displaystyle\int_{x_K}^{x_M} N_{\langle 2\rangle}\mu N_{\langle 1\rangle}\,\mathrm{d}x & \displaystyle\int_{x_K}^{x_M} N_{\langle 2\rangle}\mu N_{\langle 2\rangle}\,\mathrm{d}x \end{bmatrix}$$

which means explicitly

$$[M]^{(e)} = \begin{bmatrix} \int_{x_K}^{x_M} N_K \mu N_K \, \mathrm{d}x & \int_{x_K}^{x_M} N_K \mu N_M \, \mathrm{d}x \\ \int_{x_K}^{x_M} N_M \mu N_K \, \mathrm{d}x & \int_{x_K}^{x_M} N_M \mu N_M \, \mathrm{d}x \end{bmatrix}$$

For uniform mass density the elementwise mass matrix for the L2 finite element is

$$[M]^{(e)} = \frac{\mu(x_M - x_K)}{6} \begin{bmatrix} 2, 1 \\ 1, 2 \end{bmatrix} \tag{3.16}$$

This is commonly referred to as the ***consistent mass matrix***. Later on we will see examples of other mass matrices that results from inaccurate numerical quadrature rules.

3.6 Initial conditions

As pointed out in Section 1.5, the IBVP of the taut wire motion is rounded off with initial conditions (IC) (1.4). That means that the trial function at the time $t = 0$ should somehow approximate the prescribed initial shape and initial velocity of the called taut wire

$$w(x, t = 0) = \bar{W}(x), \quad \frac{\partial w}{\partial t}(x, t = 0) = \bar{V}(x) \quad \text{(trial function satisfies I.C.)} \tag{3.17}$$

In general we will not be able to satisfy the initial conditions exactly by the chosen trial function. For instance, let us say we wanted to set the initial deflection to a sinusoidal curve. If we use the piecewise linear test function of the form we have discussed so far, we cannot match a smooth sinusoidal curve. We then usually *approximate* the initial conditions by interpolation (see exercises 12ff.). So, for instance we can set the initial deflection in the trial function $w(x, 0) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) w_i(0)$ by computing the degrees of freedom from the interpolation condition as

$$w(x_j, 0) = \bar{W}(x_j) = \sum_{i=1}^{N} N_{\langle i \rangle}(x_j) w_i(0) = \sum_{i=1}^{N} N_i(x_j) w_{(i)}(0) = \sum_{i=1}^{N} \delta_{ji} w_{(i)}(0) = w_{(j)}(0),$$

In words, the degree of freedom at node j is set to the value of the prescribed deflection at node j.

So, we can wrap up the setup of the finite element model for the dynamics of the taut wire. By introducing the finite element basis functions we could work out that instead of a linear algebra problem as for the statics we end up with a system of coupled second order ordinary differential equations (with an initial condition), i.e. with an initial value problem (IVP)

$$\sum_{i=1}^{N_\mathrm{f}} M_{ji} \ddot{w}_i(t) + \sum_{i=1}^{N_\mathrm{f}} K_{ji} w_i(t) = L_j(t) - \sum_{i=N_\mathrm{f}+1}^{N} M_{ji} \ddot{w}_i(t) - \sum_{i=N_\mathrm{f}+1}^{N} K_{ji} w_i(t), \quad j = 1, ..., N_\mathrm{f} \tag{3.18}$$

$$\text{where} \qquad w_{(i)}(0) = \bar{W}(x_i) \, , \quad \dot{w}_{(i)}(0) = \bar{V}(x_i) \, , \quad i = 1, ..., N$$

The above IVP may be integrated for instance by converting them to first order form and using an off-the-shelf Matlab integrator. However, because of their special form, there are excellent custom-tailored algorithms for this purpose: for example the Newmark explicit algorithm discussed later in the book, or the trapezoidal rule. An important class of solution methods is based on the so-called free vibration form of this IVP, and that is what we take up next. We begin with a few exercises that solve the free vibration of the taut wire analytically.

Exercise 16.

Find analytically the natural frequencies and the normal modes of the simply-supported pre-stressed cable with a uniform mass density.

Solution: The initial boundary value problem is written as

$$Pw'' = \mu\ddot{w} \ , \quad w(0,t) = w(L,t) = 0$$

where the initial deflection and initial velocity is some smooth function of x. We are interested in finding solutions which are harmonic in time: sine or cosine function in the t variable.

We will use the technique of separation of variables: the displacement function will be sought as a product that separates space and time

$$w(x,t) = \phi(x)\psi(t)$$

The first function describes the shape of the cable, the second function gives it a variation in time.

Substituting into the equation of motion we obtain

$$P\phi''\psi = \mu\phi\ddot{\psi} \ .$$

When this is rewritten by grouping functions in x and functions in t

$$\frac{P}{\mu}\frac{\phi''(x)}{\phi(x)} = \frac{\ddot{\psi}(t)}{\psi(t)} \ ,$$

we may conclude that the ratios of the functions on either side must be constant. Otherwise the equality couldn't be true for all x's and for all times.

Functions whose second derivatives are proportional to themselves are the exponentials. Therefore we shall assume

$$\phi(x) = A\exp(\lambda x) \ , \quad \psi(t) = B\exp(\beta t) \ ,$$

where all the constants may be in general complex. However, both $\phi(x)$ and $\psi(t)$ must be real functions so that we better write

$$\phi(x) = \mathrm{Re}\left\{A\exp(\lambda x)\right\} \ , \quad \psi(t) = \mathrm{Re}\left\{B\exp(\beta t)\right\} \ ,$$

We multiply through and use the Euler identity

$$\exp(\lambda x) = \exp(\mathrm{Re}\lambda\, x)\left[\cos(\mathrm{Im}\lambda\, x) + \mathrm{i}\sin(\mathrm{Im}\lambda\, x)\right]$$

where $\mathrm{Re}\lambda$ and $\mathrm{Im}\lambda$ are the real and imaginary parts of λ. As a result we obtain

$$\phi(x) = \exp(\mathrm{Re}\lambda\, x)\left[\mathrm{Re}A\ \cos(\mathrm{Im}\lambda\, x) - \mathrm{Im}A\ \sin(\mathrm{Im}\lambda\, x)\right] \ ,$$

Now we can try to determine some of the constants by introducing the boundary conditions: Firstly

$$w(0,t) = 0 \quad \Rightarrow \quad \phi(0) = 0 \quad \Rightarrow \quad \mathrm{Re}A\ = 0$$

Secondly

$$w(L,t) = 0 \quad \Rightarrow \quad \phi(L) = \exp(\mathrm{Re}\lambda\, L)\left[0 - \mathrm{Im}A\ \sin(\mathrm{Im}\lambda\, L)\right] = 0 \quad \Rightarrow \quad \sin(\mathrm{Im}\lambda\, L) = 0$$

Therefore, we must have

$$\mathrm{Im}\lambda\, L = k\pi \ , \quad k = 1, 2, 3...$$

or in other words

$$\mathrm{Im}\lambda\ = \frac{k\pi}{L} \ , \quad k = 1, 2, 3...$$

The value $k = 0$ would also satisfy the boundary conditions, but it would yield a totally uninteresting deflection: zero everywhere. That is why we do not consider it.

The time function should describe steady-state oscillation: harmonic motion, no decay or growth of the amplitude. Therefore, in

$$\exp(\beta t) = \exp(\mathrm{Re}\beta\, t)\,[\cos(\mathrm{Im}\beta\, t) + \mathrm{i}\sin(\mathrm{Im}\beta\, t)]$$

we must have $\mathrm{Re}\beta = 0$. Upon substitution of the exponentials into the balance equation we obtain the relationship between the exponents

$$\frac{P}{\mu}\lambda^2 = \beta^2$$

or, explicitly,

$$\frac{P}{\mu}\left[(\mathrm{Re}\lambda)^2 - (\mathrm{Im}\lambda)^2 + \mathrm{i}2(\mathrm{Re}\lambda)(\mathrm{Im}\lambda)\right] = \left[(\mathrm{Re}\beta)^2 - (\mathrm{Im}\beta)^2 + \mathrm{i}2(\mathrm{Re}\beta)(\mathrm{Im}\beta)\right]$$

which holds separately for the real and imaginary part

$$\frac{P}{\mu}\left[(\mathrm{Re}\lambda)^2 - (\mathrm{Im}\lambda)^2\right] = \left[(\mathrm{Re}\beta)^2 - (\mathrm{Im}\beta)^2\right]\ , \qquad \frac{P}{\mu}\left[2(\mathrm{Re}\lambda)(\mathrm{Im}\lambda)\right] = \left[2(\mathrm{Re}\beta)(\mathrm{Im}\beta)\right]$$

and because $\mathrm{Re}\beta = 0$ we must have $\mathrm{Re}\lambda = 0$, and

$$\frac{P}{\mu}(\mathrm{Im}\lambda)^2 = (\mathrm{Im}\beta)^2$$

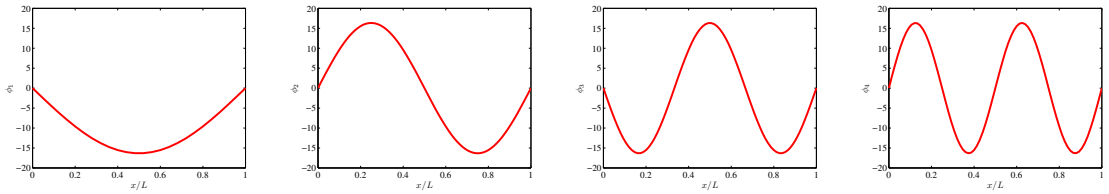
By convention we call $\omega = \mathrm{Im}\beta$ the angular frequency of oscillation. The above equation therefore determines the frequency of oscillation of the cable

$$\omega = \sqrt{\frac{P}{\mu}}\frac{k\pi}{L}\ , \quad k = 1, 2, 3...$$

In fact the above equation determines an infinite number of frequencies,

$$\omega_k = \sqrt{\frac{P}{\mu}}\frac{k\pi}{L}\ , \quad k = 1, 2, 3...$$

which are tied together with a particular shape (normal mode) of the prestressed cable that the cable executes while it is moving with this frequency. Here are the first four modes. Note that the amplitude of the modes is essentially arbitrary.



The significance of the above development is that it tells us that the cable will only vibrate freely (i.e. without forcing) with particular frequencies (natural frequencies) and it will assume only the associated shapes (normal modes) while it vibrates. The steady-state response in the most general case will be a superposition several or all of the normal modes.

To determine the solution to the free-vibration IBVP uniquely one needs to incorporate that the initial conditions. For instance, consider this set of initial conditions: the cable starts from deflection corresponding to the first normal mode

$$w(x, 0) = C\phi_1(x) = C\sin(\pi x/L)$$

and with zero initial velocity

$$\dot{w}(x,0) = 0$$

The cable will therefore vibrate only in the first normal mode. That can be proven by the following argument: when a particular normal mode is present at any time during the motion, it must be present for all times (its time dependence is a sine or cosine), including time $t = 0$ when the initial condition held. Therefore, normal modes will be present only when they are present in the initial condition.

The coefficients are found from the initial conditions as

$$\mathrm{Im}A\ \mathrm{Re}B = -C\ , \quad \mathrm{Im}B = 0$$

and the complete solution is

$$w(x,t) = C\sin(\pi x/L)\cos(\omega_1 t)$$

3.7 Free vibration

In this section we address a particular form of the system (3.18): the free vibration. In this situation there are no applied loads, and the supports are immovable. Therefore, the terms

$$L_j(t) - \sum_{i=N_\mathrm{f}+1}^{N} M_{ji}\ddot{w}_i(t) - \sum_{i=N_\mathrm{f}+1}^{N} K_{ji}w_i(t),$$

are all identically zero, and the IVP reduces to

$$\sum_{i=1}^{N_\mathrm{f}} M_{ji}\ddot{w}_i(t) + \sum_{i=1}^{N_\mathrm{f}} K_{ji}w_i(t) = 0, \quad j = 1,...,N_\mathrm{f} \tag{3.19}$$

$$\text{where} \qquad w_{(i)}(0) = \bar{W}(x_i)\ , \quad \dot{w}_{(i)}(0) = \bar{V}(x_i)\ , \quad i = 1,...,N$$

The above is written in terms of components. The matrix notation allows us to write the tidy expression

$$\boldsymbol{M}\ddot{\boldsymbol{w}} + \boldsymbol{K}\boldsymbol{w} = \boldsymbol{0}\ , \tag{3.20}$$

where $\boldsymbol{M}$ and $\boldsymbol{K}$ are square $N_\mathrm{f} \times N_\mathrm{f}$ matrices. The column matrix $\boldsymbol{w}$ collects the degrees of freedom $w_i(t)$. The solution is sought in the form $\boldsymbol{w}(t) = \boldsymbol{\phi}\cos(\omega t)$, where ω is the circular frequency of free vibration (also called natural frequency), and $\boldsymbol{\phi}$ is the eigenmode (also called normal mode, or free vibration shape).

Differentiating $\boldsymbol{w}(t) = \boldsymbol{\phi}\cos(\omega t)$ twice with respect to time yields $\ddot{\boldsymbol{w}}(t) = -\omega^2\boldsymbol{\phi}\cos(\omega t)$, and this leads to the ***generalized eigenvalue problem***

$$\left(\boldsymbol{K}\boldsymbol{\phi} - \omega^2\boldsymbol{M}\boldsymbol{\phi}\right)\cos(\omega t) = \boldsymbol{0}\ , \Rightarrow \boldsymbol{K}\boldsymbol{\phi} = \omega^2\boldsymbol{M}\boldsymbol{\phi}\ , \tag{3.21}$$

where the eigenvalue is ω^2 and the eigenvector is $\boldsymbol{\phi}$. Incidentally, assuming $\boldsymbol{w}(t) = \boldsymbol{\phi}\sin(\omega t)$ would work just as well, and the natural frequency and the mode shape would have been the same; on the other hand, the solution would evidently be able to produce a motion phase shifted by 90^o with respect to the cosine time-dependence. Therefore, in general we consider a mixture of sine and cosine and assume $\boldsymbol{w}(t) = (A\sin(\omega t) + B\cos(\omega t))\boldsymbol{\phi}$.

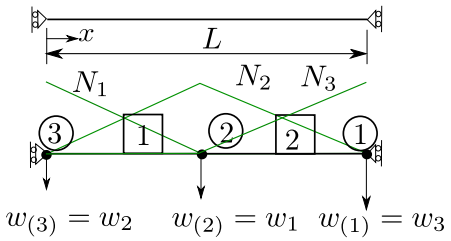


Fig. 3.1. Roller-roller prestressed cable, free vibration.

Exercise 17.

Solve the free vibration IBVP of a roller-supported prestressed cable with uniform mass density using Galerkin finite element method with 2 equal-length finite elements of the L2 type.

Solution: Note that for variety we have have changed the numbering of the nodes. The number of free degrees of freedom is $N_{\rm f} = 3$. The element equation arrays are

$$\text{Element 1: } \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \quad \text{Element 2: } \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

and $x_M - x_K = L/2$. Therefore, the elementwise stiffness matrix (2.30) and the elementwise mass matrix (3.16) are assembled into the global stiffness and mass matrix as

$$[K] = \frac{2P}{L}\begin{bmatrix} 1+1\,, & -1\,, & -1 \\ -1\,, & 1\,, & 0 \\ -1\,, & 0\,, & 1 \end{bmatrix}$$

$$[M] = \frac{\mu L}{12}\begin{bmatrix} 2+2\,, & 1\,, & 1 \\ 1\,, & 2\,, & 0 \\ 1\,, & 0\,, & 2 \end{bmatrix}$$

The free vibration eigenvalue problem is written therefore as

$$\frac{2P}{L}\begin{bmatrix} 2\,, & -1\,, & -1 \\ -1\,, & 1\,, & 0 \\ -1\,, & 0\,, & 1 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix} = \omega_j^2 \frac{\mu L}{12}\begin{bmatrix} 4\,, & 1\,, & 1 \\ 1\,, & 2\,, & 0 \\ 1\,, & 0\,, & 2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix}$$

where we expect three eigenvalues $\omega_j^2, j = 1, 2, 3$ and three eigenvectors. We write ϕ_{kj} for the $k-$th component of the $j-$th eigenvector.

How do we compute the eigenvalues without knowing P, μ, L? By considering them as parameters. For instance we may write

$$\begin{bmatrix} 2\,, & -1\,, & -1 \\ -1\,, & 1\,, & 0 \\ -1\,, & 0\,, & 1 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix} = \frac{L}{2P}\omega_j^2 \frac{\mu L}{12}\begin{bmatrix} 4\,, & 1\,, & 1 \\ 1\,, & 2\,, & 0 \\ 1\,, & 0\,, & 2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix}$$

which means that if we define

$$\Lambda_j = \frac{L}{2P}\omega_j^2 \frac{\mu L}{12}$$

we can solve for the eigenvalues and eigenvectors from

$$\begin{bmatrix} 2\,, & -1\,, & -1 \\ -1\,, & 1\,, & 0 \\ -1\,, & 0\,, & 1 \end{bmatrix} \begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix} = \Lambda_j \begin{bmatrix} 4\,, & 1\,, & 1 \\ 1\,, & 2\,, & 0 \\ 1\,, & 0\,, & 2 \end{bmatrix} \begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \end{bmatrix}$$

This eigenvalue problem can be solved with a calculator. Or with Matlab:

```
>> [Phi,Lambda]=eig([2,-1,-1;-1,1,0;-1,0,1], [4,1,1;1,2,0;1,0,2])
Phi =
   -0.2887   -0.0000   -0.5000
   -0.2887   -0.5000    0.5000
   -0.2887    0.5000    0.5000
Lambda =
   -0.0000         0         0
         0    0.5000         0
         0         0    2.0000
```

The eigenvalues are on the diagonal of the matrix `Lambda`. Which means that we can solve for the natural frequencies from

$$\omega_j^2 = \Lambda_j \frac{2P}{L}\frac{12}{\mu L}, \quad \Lambda_j = 0, 0.5, 2.0$$

The eigenvectors are the columns of the matrix `Phi`. We can see that in correspondence with the zero eigenvalue $\Lambda_1 = 0$ we have zero natural frequency, and the shape of vibration is

$$w(x,t) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) w_i(t),$$

where

$$w_i(t) = \phi_{i1} \cos(0 \times t) = \phi_{i1}$$

so that

$$w(x,t) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) w_i(t) = \phi_{i1}$$

because the basis functions sum to one, $\sum_{i=1}^{N} N_{\langle i \rangle}(x) = 1$.

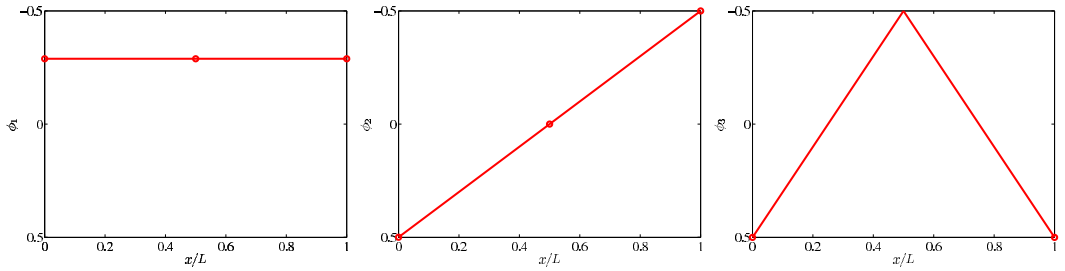


Fig. 3.2. Computed mode shapes

4

Further refinements of the Galerkin finite element method

So far we have been evaluating the needed integrals analytically. It is simple enough for the wire model and the L2 finite element, but it becomes more difficult for heat conduction or elasticity and finite elements of other shapes, triangles, tetrahedra, and so on. Numerical quadrature is an essential tool in practical finite element calculations.

4.1 Numerical quadrature

In the preceding discussion we have described how the integrals are calculated element-by-element. To evaluate the elementwise quantities (load factors, stiffness and mass matrices) we need to integrate over the length of the finite element, that is over a general interval $x_K \leq x \leq x_M$. When numerical quadrature rules are used, the integrand needs to be evaluated at selected points, and the values are given specific weights. In order for these selected points and there weights not to depend on the particular interval over which the integration needs to be performed, it is common practice to develop numerical integration rules on *standard intervals*. For us that will be $-1 \leq \xi \leq +1$ in this section (and in general for the so-called line elements in one dimension; quadrilaterals in two dimensions, and bricks in three dimensions all use this interval definition in the so-called tensor-product forms).

As an example, Simpson's 1/3 rule is given on the standard interval as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \frac{1}{3}f(\xi=-1) + \frac{4}{3}f(\xi=0) + \frac{1}{3}f(\xi=+1) \ .$$

In general, a numerical quadrature rule would be written on the standard interval $-1 \leq \xi \leq +1$ as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \sum_{k=1}^{M} f(\xi_k)W_k \ , \tag{4.1}$$

where ξ_k are the locations of the integration points, and W_k are their weights. Integrating numerically arbitrary functions over arbitrary intervals is then made possible by a map from the standard interval $-1 \leq \xi \leq +1$ to the arbitrary interval $a \leq x \leq b$

$$x = \frac{1}{2}(a+b) + \frac{1}{2}(b-a)\xi \ , \tag{4.2}$$

(the first part is the midpoint of the interval $a \leq x \leq b$, the second part is the departure from the midpoint to either side). Because this map is linear, the relationship between the differentials is constant,

$$\mathrm{d}x = \frac{1}{2}(b-a)\mathrm{d}\xi \ , \tag{4.3}$$

where the factor $\frac{1}{2}(b-a)$ is called the ***Jacobian determinant*** (or Jacobian for short). Hence, the integral over the interval $a \leq x \leq b$ is written in terms of the integral over the standard interval as

$$\int_a^b f(x)\mathrm{d}x = \frac{1}{2}(b-a)\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \tag{4.4}$$

Using such definitions we can rewrite the Simpson's 1/3 rule for an arbitrary interval $a \leq x \leq b$ as

$$\int_a^b f(x)\mathrm{d}x \approx \frac{1}{2}(b-a)\left[\frac{1}{3}f(a) + \frac{4}{3}f(\frac{1}{2}(a+b)) + \frac{1}{3}f(b)\right] .$$

This is true for the linear map (4.2). In general, when the map between the standard interval and the given interval in x is

$$x = g(\xi), \quad a = g(-1)\ , b = g(+1)\ , \tag{4.5}$$

differentiating both sides with respect to x yields

$$\frac{\mathrm{d}}{\mathrm{d}x}x = 1 = \frac{\mathrm{d}}{\mathrm{d}x}g(\xi) = \frac{\mathrm{d}g(\xi)}{\mathrm{d}\xi}\frac{\mathrm{d}\xi}{\mathrm{d}x}\ ,$$

resulting in

$$\mathrm{d}x = \frac{\mathrm{d}g(\xi)}{\mathrm{d}\xi}\mathrm{d}\xi\ ,$$

and the numerical quadrature over an arbitrary interval may be written as

$$\int_a^b f(x)\mathrm{d}x = \int_{-1}^{+1} f(\xi)\frac{\mathrm{d}g(\xi)}{\mathrm{d}\xi}\mathrm{d}\xi \approx \sum_{k=1}^{M} f(\xi_k)\frac{\partial g}{\partial \xi}(\xi_k)W_k\ , \tag{4.6}$$

where $\dfrac{\partial g}{\partial \xi}(\xi_k)$ is the Jacobian determinant evaluated at the quadrature point ξ_k.

Exercise 18. Integrate the function $f(x) = 2x^2 + \frac{x^3}{3}$ from -1 to 0 using (1) the trapezoidal rule, and (2) Simpson's rule. Compare with the analytical solution.

Solution: The analytical solution is

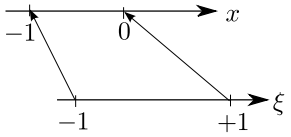
$$\int_0^{-1} 2x^2 + \frac{x^3}{3}\ dx = 7/12 = 0.58\overline{3}$$

The trapezoidal rule uses the points and weights of Table 4.1

k	ξ_k	W_k
1	-1	1
2	$+1$	1

Table 4.1. Trapezoidal quadrature rule

We consider the linear map (4.2) from the standard interval to x.



The Jacobian is therefore $\frac{0-(-1)}{2} = 1/2$. Here we define the integrand as a Matlab anonymous function:

```
f=@(x)2*x^2+(x^3)/3;% integrated function
```

The two quadrature points map to the ends of the interval in x. Therefore, with a trapezoidal rule quadrature the integral is approximated as

```
>> I=f(-1)*W_1*J +f(0)*W_2*J
I =
   0.833333333333333
```

The Simpson's rule is based on the table

k	ξ_k	W_k
1	-1	1/3
2	0	4/3
3	$+1$	1/3

Table 4.2. Simpson's quadrature rule

This time we will proceed in a systematic way by also defining a function for the map (4.2)

```
g=@(xi)1/2*(0+-1) +1/2*(-1-0)*xi;% the map from the standard interval to x
```

The Simpson's weights and quadrature point locations:

```
W_1=1/3; W_2=4/3; W_3=1/3;
xi_1=-1; xi_2=0; xi_3=1;
```

Note that writing `g(xi_1)` we obtain the location of the first quadrature point in the $-1 \leq x \leq 0$ interval. The value of the integral follows as

```
>> I=f(g(xi_1))*W_1*J +f(g(xi_2))*W_2*J +f(g(xi_3))*W_3*J
I =
   0.583333333333333
```

Note that the Simpson's quadrature rule is exact for the present integrand.

In order to be able to perform integration over the standard interval it will be useful to have the basis functions expressed in terms of the coordinate ξ. Basis functions expressed on the standard interval (4.7) are sometimes referred to as being expressed in the ***parametric coordinates***. The procedure of Section 2.9 is easily applied to construct linear functions over this interval (Figure 4.1): just imagine that the mesh consists of a single element, with node i at $\xi = -1$ and node j at $\xi = +1$. The basis functions will then be

$$N_i(\xi) = \frac{\xi - 1}{-1 - 1} = \frac{\xi - 1}{-2}, \quad N_j(\xi) = \frac{\xi - (-1)}{+1 - (-1)} = \frac{\xi + 1}{+2}\,. \tag{4.7}$$

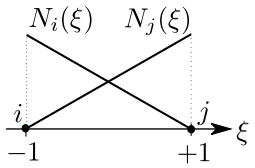


Fig. 4.1. L2 finite element basis functions on the standard interval.

The expressions (4.7) are just the Lagrange interpolation polynomials on the standard interval. As we shall see later, writing down the basis functions over a *standard shape* – a square for general quadrilaterals, a cube for general brick elements, standards triangles or standard tetrahedra for general triangles or general tetrahedra, and so on– as opposed to the general shape (general distorted quadrilaterals, bricks or tetrahedra), is not only convenient, but also highly advisable from the point of view of computer implementation: most of the code for different element shapes and types is then shared, and does not have to be repeated.

The coordinate x is related for the L2 element to the parametric coordinate ξ through the map (4.2). When we introduce other element shapes, we will refine the concept of the map between the parametric coordinate and the physical coordinates using the concept of ***isoparametric formulation***.

Exercise 19. How are the basis functions expressed in the x coordinate related to the basis functions (4.7)?

Solution: Take the element connecting nodes i at x_i and j at x_j and construct its basis functions

$$N_i(x) = \frac{x - x_j}{x_i - x_j} \ , \quad N_j(x) = \frac{x - x_i}{x_j - x_i} \ , \tag{4.8}$$

and substitute for x from (4.2), which for the present situation is specialized as

$$x = \frac{1}{2}(x_i + x_j) + \frac{1}{2}(x_j - x_i)\xi \ . \tag{4.9}$$

We obtain

$$N_i(x) = \frac{\frac{1}{2}(x_i + x_j) + \frac{1}{2}(x_j - x_i)\xi - x_j}{x_i - x_j} = \frac{\frac{1}{2}(x_i - x_j) - \frac{1}{2}(x_i - x_j)\xi}{x_i - x_j} = \frac{1}{2}(1 - \xi)$$

and

$$N_j(x) = \frac{\frac{1}{2}(x_i + x_j) + \frac{1}{2}(x_j - x_i)\xi - x_i}{x_j - x_i} = \frac{\frac{1}{2}(x_j - x_i) + \frac{1}{2}(x_j - x_i)\xi}{x_j - x_i} = \frac{1}{2}(1 + \xi)$$

These expressions are exactly (4.7). So, provided the coordinates are related by (4.9) the two ways of writing the basis functions for an element are one and the same. We can write down the basis functions in terms of x and then substitute from (4.9) to get (4.7). Or, we can start with (4.7), substitute the inverse of (4.9)

$$\xi = \frac{x - \frac{1}{2}(x_i + x_j)}{\frac{1}{2}(x_j - x_i)} \ .$$

and we obtain (4.8).

Now we can use a numerical rule to evaluate some finite element quantity. For instance, let us use numerical integration to evaluate the elementwise load vector for a load with linear variation along the element.

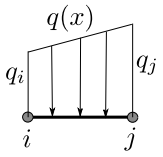


Fig. 4.2. Linearly varying distributed load

Exercise 20. Use the Simpson quadrature rule to evaluate the elementwise load vector for linearly varying distributed load.

Solution: The distributed load varies linearly from $q(x_i) = q_i$ at node i to $q(x_j) = q_j$ at node j. We can describe such variation using Lagrange interpolation polynomials. As we have discussed above, those are exactly the basis functions (4.8), so that we write

$$q(x) = q_i N_i(x) + q_j N_j(x)$$

Since we would like to use the numerical rule in the parametric coordinates, we shall plug-in x from (4.9). This takes no work at all, since as explained in Exercise 19 when we substitute for x into the basis functions we get the equivalent basis function expressions in ξ, and therefore distributed load varies in the parametric coordinates as

$$q(\xi) = q_i N_i(\xi) + q_j N_j(\xi)$$

We are now ready to use the Simpson's rule from Table 4.2 to evaluate the integral (4.4) where the integrand is

$$f(\xi) = N_i(\xi) q(\xi)$$

to evaluate the component at node i, and

$$f(\xi) = N_j(\xi) q(\xi)$$

to evaluate the component at node j. So we have

$$L_1^{(e)} = \int_{x_i}^{x_j} N_i(x) q(x) \mathrm{d}x = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_i(\xi) q(\xi) \mathrm{d}\xi$$

which we can in fact evaluate analytically as a check

$$L_1^{(e)} = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_i(\xi) q(\xi) \mathrm{d}\xi = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_i(\xi) \left(q_i N_i(\xi) + q_j N_j(\xi) \right) \mathrm{d}\xi = (x_j - x_i) \left(\frac{q_i}{3} + \frac{q_j}{6} \right)$$

For the Simpson's rule we have the values of the integrand at the quadrature points

$$\xi_1 = -1: \quad N_i(\xi_1) \left(q_i N_i(\xi_1) + q_j N_j(\xi_1) \right) = 1 \times (q_i \times 1 + q_j \times 0) = q_i$$

$$\xi_2 = 0: \quad N_i(\xi_2) \left(q_i N_i(\xi_2) + q_j N_j(\xi_2) \right) = 1/2 \times (q_i \times 1/2 + q_j \times 1/2) = (1/4) q_i + (1/4) q_j$$

$$\xi_3 = +1: \quad N_i(\xi_3) \left(q_i N_i(\xi_3) + q_j N_j(\xi_3) \right) = 0 \times (q_i \times 0 + q_j \times 1) = 0$$

so that the integral is approximated as

$$\begin{aligned} L_1^{(e)} &= \frac{x_j - x_i}{2} \int_{-1}^{+1} N_i(\xi) q(\xi) \mathrm{d}\xi \approx \frac{x_j - x_i}{2} \left\{ \frac{1}{3} q_i + \frac{4}{3} [(1/4) q_i + (1/4) q_j] + \frac{1}{3} 0 \right\} \\ &= (x_j - x_i) \left(\frac{q_i}{3} + \frac{q_j}{6} \right) \end{aligned}$$

Note that we have gotten in fact the exact same results as from the analytical integration. In this case, Simpson's rule was not approximate but exact. Similarly we would have arrived at

$$L_2^{(e)} = \int_{x_i}^{x_j} N_j(x) q(x) \mathrm{d}x = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_j(\xi) q(\xi) \mathrm{d}\xi$$

and

$$L_2^{(e)} = (x_j - x_i) \left(\frac{q_i}{6} + \frac{q_j}{3} \right)$$

4.2 Gauss quadrature

The numerical quadratures that are in common use with polynomial finite elements are the ***Gauss rules***. They are by now standard fare described in any number of textbooks (e.g. [H00]). For completeness we present a simple version of the logic of Gaussian quadrature rules below.

We will start by explaining the basic idea on the example of a one-point integration rule. The idea is in fact common to a variety of quadrature rules: instead of the original function $f(\xi)$ we integrate an interpolation polynomial $L(\xi)$ that passes through selected points on the function curve.

For our selected example, a one-point rule, we will consider an interpolating polynomial that passes through a single point. Such a polynomial is a constant, $L(\xi) = \text{const} = f(\xi_1)$. The coordinate ξ_1 is the location of the integration point, and at this point it is unknown. We are now going to formulate a condition from which to compute the optimal location of this integration point. The function $f(\xi)$ is integrated on the standard interval $-1 \leq \xi \leq +1$ numerically using the rule, with $M = 1$, and ξ_1 and W_1 to be determined, as

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi \approx \sum_{k=1}^{M} L(\xi_k)W_k = L(\xi_1)W_1$$

where the polynomial L interpolates at the integration point, $L(\xi_1) = f(\xi_1)$. Since for the case the given function is a constant, $f(\xi) = C$, we must have that

$$\int_{-1}^{+1} f(\xi)\mathrm{d}\xi = 2C = L(\xi_1)W_1 = f(\xi_1)W_1 = CW_1 \tag{4.10}$$

the weight of the integration point must be $W_1 = 2$.

Next we will consider the location of the integration point. Our criterion will be to make the rule as accurate as possible. In particular, if the given function is a polynomial of a certain degree the integration rule should integrate it exactly. That means that the integral of the difference $F(\xi) = f(\xi) - L(\xi)$ must vanish

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = 0 \tag{4.11}$$

Because L interpolates at the integration point, the difference $F(\xi)$ assume value zero at the integration point, $F(\xi_1) = 0$. This can be accomplished by writing $F(\xi)$ as the product of two polynomials

$$F(\xi) = (\xi - \xi_1)q(\xi) \tag{4.12}$$

The first term, $(\xi - \xi_1)$, ensures that $F(\xi_1) = 0$. What could the second term look like? Consider a linear polynomial as an example

$$q(\xi) = A\xi + B$$

Substituting for $F(\xi)$ we obtain for the integral (4.11)

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = \int_{-1}^{+1} (\xi - \xi_1)(A\xi + B)\mathrm{d}\xi$$

If the above integral should become zero for arbitrary A and B

$$\int_{-1}^{+1} (\xi - \xi_1)(A\xi + B)\mathrm{d}\xi = 0 \quad \Rightarrow \quad A\int_{-1}^{+1} (\xi - \xi_1)\xi\mathrm{d}\xi = 0 \quad \text{and} \quad B\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = 0$$

However since we have only ξ_1 to play with, we cannot possibly make both $\int_{-1}^{+1}(\xi - \xi_1)\xi\mathrm{d}\xi = 0$ and $\int_{-1}^{+1}(\xi - \xi_1)\mathrm{d}\xi = 0$. Our guess of $q(\xi)$ as a linear polynomial was too ambitious. We should have

picked a polynomial with a single unknown coefficient, $q(\xi) = D$ (a constant polynomial). Then if we substitute for $F(\xi)$ in the integral (4.11) we obtain

$$\int_{-1}^{+1} F(\xi)\mathrm{d}\xi = \int_{-1}^{+1} (\xi - \xi_1)D\mathrm{d}\xi$$

which means the condition for ξ_1 reads

$$\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = 0$$

Therefore

$$\int_{-1}^{+1} (\xi - \xi_1)\mathrm{d}\xi = \left[\xi^2/2 - \xi\xi_1\right]_{-1}^{+1} = -2\xi_1 = 0 \quad \Rightarrow \quad \xi_1 = 0$$

To summarize, the one-point Gauss quadrature rule is defined by the parameters $\xi_1 = 0$ and $W_1 = 2$. It can exactly integrate a linear polynomial, as evident from the fact that it can exactly integrate the polynomial (4.12). Using this route leads to Gauss rules with an arbitrary number of points.

Rule	Coordinates ξ_j	Weights W_j	Integrates exactly
1-point	0	2	linear polynomial
2-point	$-\sqrt{1/3}$	1	cubic polynomial
	$+\sqrt{1/3}$	1	
3-point	$-\sqrt{3/5}$	5/9	quintic polynomial
	0	8/9	
	$+\sqrt{3/5}$	5/9	

Table 4.3. Gauss quadrature rules

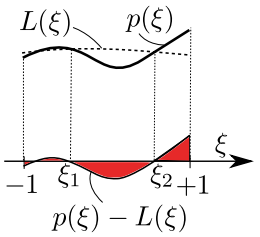
Table 4.3 lists three most common ones. For completeness: these are not the only Gauss rules that are in use in the finite element methods. The rules we discussed here are the so-called open quadrature rules since the endpoints of the interval are not included amongst the integration points.

Exercise 21. Determine the parameters of a two-point Gauss quadrature rule.

Solution: The goal is to integrate the function $p(\xi)$ by integrating its interpolating polynomial $L(\xi)$. Interpolation conditions are satisfied at the quadrature points. We have to find the location of the quadrature points ξ_1, ξ_2 so that the polynomial $p(\xi) - L(\xi)$ becomes zero at the quadrature points, and at the same time integrates to zero

$$\int_{-1}^{+1} p(\xi) - L(\xi)\, dx = 0$$

Consult the figure below– the red-filled areas must cancel:



We will tailor the Gauss rule to work well for polynomials. We will assume therefore that both $p(\xi)$ and $L(\xi)$ are polynomials and therefore their difference is also going to be a polynomial

$$F(\xi) = p(\xi) - L(\xi)$$

It is important to realize that all such polynomials may be written as

$$F(\xi) = (\xi - \xi_1)(\xi - \xi_2)q(\xi)$$

The term $(\xi - \xi_1)$ makes sure that the interpolating condition holds at ξ_1 since it guarantees $F(\xi_1) = 0$. Similarly the term $(\xi - \xi_2)$ makes sure that the interpolating condition holds at ξ_2. The remaining term, $q(\xi)$, is at this point arbitrary, but must satisfy

$$\int_{-1}^{+1} F(\xi)\, dx = 0$$

In order to make this happen we can choose the locations of the quadrature points. Since we have two quadrature points, we need exactly two equations. Such equations will materialize if we choose $q(\xi) = A\xi + B$ as

$$\int_{-1}^{+1} F(\xi)\, dx = \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)q(\xi)\, dx = \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)(A\xi + B)\, dx = 0$$

means that because A and B are arbitrary we obtain two equations

$$\int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)\xi\, dx = 0\,, \quad \int_{-1}^{+1} (\xi - \xi_1)(\xi - \xi_2)\, dx = 0$$

that have to be satisfied at the same time. These two equations can be solved for the locations of the quadrature points. The two integrals are evaluated as

```
syms xi xi1 xi2 real
I1 =int((xi-xi1)*(xi-xi2),xi,-1,+1);
I2 =int(xi*(xi-xi1)*(xi-xi2),xi,-1,+1);
```

and `I1,I2` become expressions in the unknowns `xi1,xi2`. The integrals are set equal to zero and the resulting system of equations is solved for the locations of the quadrature points:

```
solution =solve(I1,I2,'xi1','xi2');
solution.xi1
solution.xi2
ans =
  1/3*3^(1/2)
 -1/3*3^(1/2)
ans =
 -1/3*3^(1/2)
  1/3*3^(1/2)
```

Evidently the locations of the quadrature points are $\xi_1 = -\frac{1}{\sqrt{3}}$ and $\xi_2 = +\frac{1}{\sqrt{3}}$.

The weights of the two-point Gauss rule can now be determined so that when the original function $p(\xi)$ and the interpolating function $L(\xi)$ are one and the same function the numerical integral comes out exact. Since the interpolating function is determined by two points, $p(\xi)$ must be a linear polynomial $(A\xi + B)$. Therefore we require that the exact integral of such a function

$$\int_{-1}^{+1} (A\xi + B)\, dx = 2B$$

is identically reproduced by the numerical quadrature rule

$$\int_{-1}^{+1} (A\xi + B)\, dx \approx (A\xi_1 + B)W_1 + (A\xi_2 + B)W_2 = 2B$$

Therefore we need

$$A\xi_1 W_1 + A\xi_2 W_2 = 0 \Longrightarrow W_1 = W_2$$

and

$$BW_1 + BW_2 = 2B \Longrightarrow W_1 + W_2 = 2$$

Hence we conclude $W_1 = W_2 = 1$. In summary, we obtain the following table for the two-point Gauss rule:

k	ξ_k	W_k
1	$-\frac{1}{\sqrt{3}}$	1
2	$+\frac{1}{\sqrt{3}}$	1

Since we have shown that the integration is exact for the difference of the interpolated and interpolating function being $F(\xi) = (\xi - \xi_1)(\xi - \xi_2)q(\xi) = (\xi - \xi_1)(\xi - \xi_2)(A\xi + B)$, which is a cubic polynomial, the two-point Gauss rule is *exact* for constant, linear, quadratic, and cubic polynomials.

4.3 Derivatives of basis functions

Integrals are in the finite element method carried out over standard-shape domains, such as the standard interval for the L2 finite elements. However, this means that if we wish to evaluate for instance the stiffness matrix, we have to express the derivative of the basis functions using a chain rule. For the stiffness matrix the integrand consists of products that look like

$$N'_{\langle p\rangle}(x) P N'_{\langle m\rangle}(x)$$

or, when expressed in the parametric coordinate,

$$N'_{\langle p\rangle}(\xi) P N'_{\langle m\rangle}(\xi)$$

While the arguments changed to ξ, we still need to take derivatives of the basis functions with respect to x, as indicated by the prime symbol. As we will see later, we will commonly know expressions for the basis functions only in the parametric coordinate (as opposed to the L2 element where it isn't difficult to work out the expressions in x). Therefore, we will need to compute the derivative using the chain rule:

$$\frac{\partial N_i(\xi)}{\partial x} = \frac{\partial N_i(\xi)}{\partial \xi}\frac{\partial \xi}{\partial x}\,. \tag{4.13}$$

The partial derivative $\dfrac{\partial \xi}{\partial x}$ is readily obtainable from (4.2), which we invert

$$\xi = \frac{x - \frac{1}{2}(a+b)}{\frac{1}{2}(b-a)}\,,$$

and then differentiated

$$\frac{\partial \xi}{\partial x} = \frac{1}{\frac{1}{2}(b-a)} \tag{4.14}$$

and may be identified as the inverse of the Jacobian determinant (4.3).

Exercise 22. Compute the derivatives of the basis functions with respect to x starting from the expression for the basis functions in the parametric coordinates (4.7).

Solution: We already have $\frac{\partial \xi}{\partial x}$ from (4.14):

$$\frac{\partial \xi}{\partial x} = \frac{2}{(x_j - x_i)}$$

Now we only need the partial derivatives of (4.7)

$$\frac{\partial N_i(\xi)}{\partial x} = -\frac{1}{2}\,, \quad \frac{\partial N_j(\xi)}{\partial x} = +\frac{1}{2}\,,$$

in order to evaluate

$$N_i' = \frac{\partial N_i(\xi)}{\partial x} = \frac{\partial N_i(\xi)}{\partial \xi}\frac{\partial \xi}{\partial x} = -\frac{1}{2}\frac{2}{(x_j - x_i)} = -\frac{1}{(x_j - x_i)}\,.$$

and

$$N_j' = \frac{\partial N_j(\xi)}{\partial x} = \frac{\partial N_j(\xi)}{\partial \xi}\frac{\partial \xi}{\partial x} = \frac{1}{2}\frac{2}{(x_j - x_i)} = \frac{1}{(x_j - x_i)}\,.$$

Let us consider for instance the expression for the mass matrix. The integral would be approximated with a numerical quadrature rule as

$$\int_{x_i}^{x_j} N_p(x)\mu(x)N_m(x)\,\mathrm{d}x \approx \frac{1}{2}(x_j - x_i)\sum_{k=1}^{M} N_p(\xi_k)\mu(\xi_k)N_m(\xi_k)W_k\,,$$

where the integration rule is as yet undetermined: we could choose any one of the dozens of numerical rules available in the literature. What would the rationale for these choices be? At first sight, of the integrals for the stiffness matrix (2.19) and for the mass matrix (3.13), the latter will require a more accurate numerical quadrature rule: the stiffness matrix involves products of the derivatives of the basis functions, which for linear basis functions are constants; the mass matrix, on the other hand, requires products of the basis functions themselves, which are linear functions of x. Therefore, the stiffness matrix consists of integrals of constants, while the mass matrix elements are integrals of quadratic functions. Interestingly, increased efficiency and even higher accuracy may be occasionally achieved if the mass matrix is *not* integrated exactly. In particular, diagonal (lumped) mass matrices are often used to achieve both benefits in wave propagation problems. We will take up this topic in detail later in the book.

5

More about Boundary Conditions

In this chapter we will explore the effect the boundary conditions have on the solution, both its existence and its computability with the Galerkin model.

We will consider only statics, so that the balance equation (1.1) drops the inertial term

$$Pw'' + q = 0 \ , \tag{5.1}$$

and furthermore we will assume that the transverse load q is a constant. With the definition $k = -q/P$, the task is to integrate

$$w'' = k = \text{constant} \ , \tag{5.2}$$

which is easily accomplished as

$$w(x) = k\frac{x^2}{2} + Cx + D \ , \tag{5.3}$$

where C and D are integration constants to be determined from the boundary condition.

For the model of the wire the boundary of the domain is composed of two disjoint sets, each consisting of a single point at either end. As already mentioned (Section 1.4), one condition specified on the entire boundary can be used to determine the solution. Since the balance equation is of second order, and the unknown is a single function, a single boundary condition is all that is needed. In one dimension, a simple explanation is that a second order equation needs two integration constants; we will talk about this issue in higher dimensional domains when we deal with the heat conduction equation and also in the part dedicated to the elasticity model.

For taut wire model the boundary condition may be of two distinct types: either prescribed deflection (an essential boundary condition), or prescribed slope (derivative of the deflection, which is a natural boundary condition). Since we may prescribe only one boundary condition, but the boundary consists of two disjoint sets, we may in fact prescribe one or the other type at either of the two endpoints. Both existence and uniqueness of the solution depend on which type of boundary condition is applied, and the various possibilities are discussed in this chapter.

5.1 Mixed essential and natural boundary conditions

This case of boundary condition was treated in the previous few chapters. The natural condition may be expressed in terms of the end-point transverse forces. At $x = L$ the relation is Eq. (1.3), and it may be derived in the same way at $x = 0$ as

$$Pw'(0) + F_0 = 0 \ . \tag{5.4}$$

The Galerkin algebraic equations for an essential boundary condition at $x = 0$ and a natural boundary condition at $x = L$ are given by Eq. (2.28) (statics). When the points of application

of these boundary conditions are switched, the changes are limited to the boundary term and the conditions for the test and trial functions only

$$N_{\langle j\rangle}(0)F_0 - \sum_{i=1}^{N_\mathrm{f}}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i - \sum_{i=N_\mathrm{f}+1}^{N}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i + \int_0^L N_{\langle j\rangle} q \,\mathrm{d}x = 0, \quad j = 1, ..., N_\mathrm{f} \ . \tag{5.5}$$

After the solution for the deflection is obtained, the slope at the end with the essential boundary condition may be computed, yielding the associated reaction force. For instance, the analytical solution of (5.1), with the boundary conditions

$$w'(0) = -\frac{F_0}{P} = \bar{w}_0', \quad w(L) = \bar{w}_L \ ,$$

is

$$w(x) = \frac{k}{2}(x^2 - L^2) + \bar{w}_0'(x - L) + \bar{w}_L \ .$$

This yields the reaction force $F_L = P(kL + \bar{w}_0')$. Therefore, we may conclude that the solution exists and it is unique. This rests on our ability to (a) determine the integration constants, and to (b) determine them uniquely.

5.2 Essential boundary conditions only

The boundary conditions are in this case the deflections given at both ends,

$$w(0) = \bar{w}_0, \quad w(L) = \bar{w}_L \ ,$$

which may be substituted into (5.3) for $x = 0$ and $x = L$ to yield two equations for C and D. Clearly, the solution exists and is unique. (We have been able to determine the integration constants, and determine them uniquely.)

Upon substitution, the slopes at the end points are available as

$$w'(0) = \frac{\bar{w}_L - \bar{w}_0}{L} - k\frac{L}{2}, \quad w'(L) = \frac{\bar{w}_L - \bar{w}_0}{L} + k\frac{L}{2} \ .$$

From the slopes, the forces F_0 and F_L are available from (5.4) and (1.3). The physical meaning of these two forces is that of reactions at the supports. The algebraic equation for this type of boundary conditions are obtained from either (5.5) or (2.28) by omitting the term with the boundary force.

5.3 Natural boundary conditions only

The boundary conditions are in this case the slopes given at both ends,

$$w'(0) = \bar{w}_0', \quad w'(L) = \bar{w}_L',$$

Integrating (5.2) once yields for the slope

$$w'(x) = kx + C \ ,$$

which may be used in conjunction with the boundary conditions to express the constant C as either of the two expressions

$$C = \bar{w}_0' \ , \quad \text{or} \quad C = \bar{w}_L' - kL \ .$$

Clearly, this is only possible if

$$\bar{w}_0' = \bar{w}_L' - kL \ ,$$

which may be interpreted as a condition under which a solution exists: if the two slopes are linked by the previous equation, solution exists; otherwise no solution exists, because the boundary conditions are contradictory. So this situation is quite different from the one encountered for the mixed or essential-only boundary condition cases. Now it is possible that no solution exists. Furthermore, if the solution exists, the two given slopes $\bar{w}_0', \bar{w}_L'$ are not independent as they are linked by the above condition. Therefore, we have not supplied enough information to determine *two* constants of integration, but only *one*. Correspondingly, the deflection of the wire is then

$$w(x) = k\frac{x^2}{2} + \bar{w}_0' x + D \ ,$$

where the constant D remains undetermined.

An initial boundary value problem with natural boundary conditions only is called a ***pure-traction problem***, or Neumann problem. We could see that the solution then exists only under certain conditions. In this case, the condition is one of *static equilibrium*: the end-point forces must balance the transverse load. Provided equilibrium may be established, the solution still remains *non-unique*, as it is possible to translate the wire perpendicularly to its axis without affecting the equilibrium.

5.4 Concentrated forces in the interior

In this section we will consider the implications of having concentrated forces acting on the taut wire in between the supports at the ends. Figure 5.1 illustrates the situation in question. In order to make progress we need to consider what happens to the balance equation

$$Pw'' + q = 0$$

in the vicinity of the concentrated force. We can explicitly solve for the curvature w'' as

$$w'' = -\frac{q}{P}$$

Since the prestress force P is uniform and given as data, the curvature of the wire changes from point to point in dependence on the transverse distributed load. In particular, we see that the wire has zero curvature when it isn't loaded ($q = 0$). Conversely, the larger the distributed load q the larger the curvature.

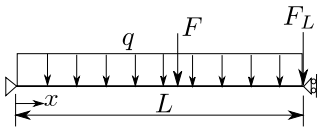


Fig. 5.1. Taught wire with an interior concentrated force

We will introduce the concentrated force as the limit of uniformly distributed load acting on a shrinking length of the wire. Figure 5.2 shows how to apply a total force of magnitude F by distributing uniform load q on a segment of length d. In order to apply the total force, we require for the three quantities satisfy

$$F = qd$$

Therefore, if we make the length of the segment shrink towards zero, $d \to 0$, the distributed load magnitude will at the same time approach infinity because the total force should be preserved

$$q = \lim_{d \to 0} \frac{F}{d} = \infty$$

Consequently the curvature of the wire will also acquire infinite magnitude

$$\lim_{d \to 0} w'' = \lim_{d \to 0} (-\frac{q}{P}) = -\infty$$

underneath the distributed load as the width shrinks to zero. The problem is what to make of the balance equation in this situation. The presence of the infinite quantities draws the validity of the equation at such points into question. Therefore, it makes sense to avoid the need to write the balance equation underneath a concentrated force. In this sense, any point of application of a concentrated force would be as special as the boundary of the wire: Note that the balance equation is written in the interior of the wire, not on the boundary. The solution then seems to be to split the wire domain into intervals. What kind of equation do we use to replace the balance equation at the point of application of concentrated force though?

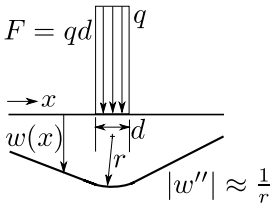


Fig. 5.2. Schematic of distributed load which converges to a concentrated force

The answer is continuity equations. First let us look at the deflection curve itself. That is certainly continuous everywhere along the length of the wire, including underneath the force. For definiteness, let us consider a single interior force at the location $x = x_F$. The first continuity condition is therefore written as

$$w(x_F^-) = w(x_F^+) \tag{5.6}$$

where by x_F^- (and x_F^+) we mean immediately to the left (immediately to the right) of the point x_F.

Next we may consider the slope of the wire. But that is not continuous underneath the concentrated force: Since the magnitude of the curvature and the radius of curvature r are inversely related

$$|w''| \approx \frac{1}{r}$$

we may conclude

$$\lim_{d \to 0} r = 0$$

underneath the distributed load as the width shrinks to zero. Another way of describing this phenomenon is to say that a sharp corner (a kink) will develop in the deflection curve as the distributed load approaches the limit of a concentrated force. A kink is a point where the slope changes discontinuously.

The slope is discontinuous, but we can still use the slope: we can write the equilibrium of forces that act on a small neighborhood of the wire in the vicinity of the concentrated force, and the transverse forces produced by the wire are proportional to the slopes

$$-Pw'(x_F^-) + Pw'(x_F^+) + F = 0 \tag{5.7}$$

The first term is the transverse force immediately to the left of the concentrated force, pulling upwards. The second term is the transverse force immediately to the right of the concentrated force, pulling downwards, and finally we account for the concentrated force F itself (positive downwards).

To summarize we will write down the BVP of taut wire statics which includes a concentrated force in the interior of the wire (Figure 5.1). This will be the analogy of (2.1)– (2.3). The domain of the wire is split into two subintervals: $0 < x < x_F$ and $x_F < x < L$. The balance equation

$$Pw''(x) + q(x) = 0 \ ,$$

is valid in these intervals. The essential boundary condition

$$w(0) = \bar{w}_0 \ .$$

and the natural boundary condition

$$-Pw'(L) + F_L = 0$$

are unchanged. Finally, to compensate for the point x_F where the balance equation is not applicable we add the deflection continuity (5.6)

$$w(x_F^-) = w(x_F^+) \ ,$$

and the force equilibrium (5.7)

$$-Pw'(x_F^-) + Pw'(x_F^+) + F = 0 \ . \tag{5.8}$$

To derive the weighted residual equation we apply the same reasoning as in Section 2.5 to shift derivative from the trial function to the test function. This must be done interval by interval, since we must leave out x_F, and therefore we write

$$\begin{aligned}
&\int_0^{x_F^-} \eta_j P w'' \, \mathrm{d}x + \int_{x_F^+}^{L} \eta_j P w'' \, \mathrm{d}x = \\
&\eta_j(x_F^-) P w'(x_F^-) - \eta_j(0) P w'(0) - \int_0^{x_F^+} {\eta_j}' P w' \, \mathrm{d}x \\
&+\eta_j(L) P w'(L) - \eta_j(x_F^+) P w'(x_F^+) - \int_{x_F^+}^{L} {\eta_j}' P w' \, \mathrm{d}x \ ,
\end{aligned}$$

This is then substituted into the weighted residual balance equation, and combined with the natural boundary condition and with (5.8) in weighted residual form as in Section 2.7. The term

$$\eta_j(0) P w'(0)$$

drops out because we must require $\eta_j(0) = 0$ because of the essential boundary condition, and the term $Pw'(L)$ is replaced with F_L, and finally the two terms

$$\eta_j(x_F^-) P w'(x_F^-) - \eta_j(x_F^+) P w'(x_F^+)$$

are replaced with

$$\eta_j(x_F) F$$

using (5.8). Therefore, the only change with respect to our previous final expression for the statics with the Galerkin method (2.15) is that we picked up one more term that involves a concentrated force

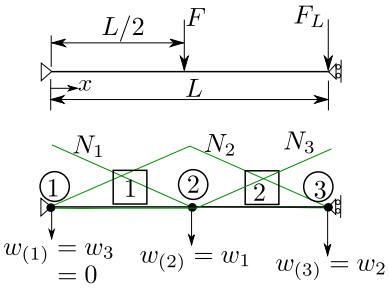


Fig. 5.3. Taut wire with two concentrated forces

$$\eta_j(L)F_L + \eta_j(x_F)F - \int_0^L \eta_j{}' P w' \, \mathrm{d}x + \int_0^L \eta_j q \, \mathrm{d}x = 0, \quad j = 1, ..., N \, , \tag{5.9}$$

We shall practice this now on an example.

Exercise 23.

Consider the taut wire with two concentrated forces in Figure 5.3. Solve with a finite element model shown (two finite elements, three nodes, two degrees of freedom).

Solution: We will modify the finite element weighted residual equation (2.28) based on (5.9) by deleting the term with the distributed load and the term with the prescribed displacements (since the prescribed displacements are zero), and adding the term due to the concentrated interior force F

$$\sum_{i=1}^{N_\mathrm{f}} \left(\int_0^L N_{\langle j \rangle}{}' P N_{\langle i \rangle}{}' \, \mathrm{d}x \right) w_i = N_{\langle j \rangle}(L)F_L + N_{\langle j \rangle}(x_F)F, \qquad j = 1, ..., N_\mathrm{f} \, . \tag{5.10}$$

On the left-hand side we will obtain the same stiffness matrix (2.25) as in Section 2.10. The load vector is assembled as follows: For degree of freedom 1 we write the right-hand side as

$$N_{\langle 1 \rangle}(L)F_L + N_{\langle 1 \rangle}(x_F)F = N_2(L)F_L + N_2(x_F)F = 0 \times F_L + 1 \times F = F$$

which readily follows from inspection of Figure 5.3. This means that F is assembled to the global load vector component L_1. For degree of freedom 2 we write the right-hand side as

$$N_{\langle 2 \rangle}(L)F_L + N_{\langle 2 \rangle}(x_F)F = N_3(L)F_L + N_3(x_F)F = 1 \times F_L + 0 \times F = F_L$$

which means that F_L is assembled to the global load vector component L_2. So now we have the stiffness matrix on the right-hand side vector assembled and we can solve for the unknown degrees of freedom. For practice we will use the principle of superposition. We will split the right-hand side into two load vectors, one for F and one for F_L so that we will solve these two systems of coupled linear equations

$$\frac{2P}{L} \begin{bmatrix} 2 \, , & -1 \\ -1 \, , & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} F \\ 0 \end{bmatrix} \, , \quad \frac{2P}{L} \begin{bmatrix} 2 \, , & -1 \\ -1 \, , & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0 \\ F_L \end{bmatrix}$$

whose solutions are

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{L}{2P} \begin{bmatrix} 1 \, , & 1 \\ 1 \, , & 2 \end{bmatrix} \begin{bmatrix} F \\ 0 \end{bmatrix} = \frac{LF}{2P} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \, , \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \frac{L}{2P} \begin{bmatrix} 1 \, , & 1 \\ 1 \, , & 2 \end{bmatrix} \begin{bmatrix} 0 \\ F_L \end{bmatrix} = \frac{LF_L}{2P} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

A cool way of presenting these results is through deflections due to single unit loads. Assuming in turn that $F = 1$ and $F_L = 1$ yields

$$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \underbrace{\frac{L}{2P}\begin{bmatrix} 1 \\ 1 \end{bmatrix}}_{\text{Deflection due to unit load at degree of freedom 1}} , \quad \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \underbrace{\frac{L}{2P}\begin{bmatrix} 1 \\ 2 \end{bmatrix}}_{\text{Deflection due to unit load at degree of freedom 2}}$$

which are the displacement vectors as deflection patterns due to unit loads at the individual degrees of freedom (Figure 5.4). Then, if we are given actual values for F and F_L we multiply these solution vectors with the force magnitudes and add them together (that is we form a linear combination of the solutions).

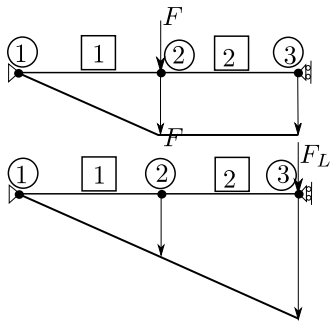


Fig. 5.4. Taut wire with two concentrated forces, solutions

5.5 Elementwise stiffness matrix properties

In the preceding sections we have seen how the boundary conditions affect the existence and uniqueness of the analytical solution. Now we are going to have a look at the finite element solution.

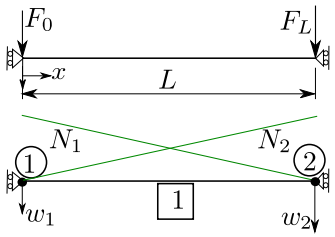


Fig. 5.5. Free-free taut wire and the single-element model

The elementwise stiffness matrix (2.30) is going to help us understand the questions of existence and uniqueness of solutions to the equilibrium equations (2.28).

Consider first the natural boundary condition-only (free-free taut wire) case of Section 5.3, but without distributed load q, with the mesh consisting of a single element (Figure 5.5). There are two degrees of freedom, w_1 and w_2 and the finite element equations are modified from (5.5) by including both forces applied at the rollers as

$$N_{\langle j\rangle}(0)F_0 + N_{\langle j\rangle}(L)F_L - \sum_{i=1}^{2}\left(\int_0^L {N_{\langle j\rangle}}' P {N_{\langle i\rangle}}' \,\mathrm{d}x\right) w_i = 0, \quad j = 1, 2 \,. \tag{5.11}$$

Using the elementwise stiffness matrix expression (2.30) we easily work out that these equations are equivalent to

$$\frac{P}{L}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} F_0 \\ F_L \end{bmatrix}$$

or using bold math symbols

$$\boldsymbol{K}\boldsymbol{w} = \boldsymbol{L} \tag{5.12}$$

The solution requires that we invert the matrix

$$\boldsymbol{K} = \frac{P}{L}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} \tag{5.13}$$

but in order for the matrix to be invertible, its determinant must be different from zero. This is not verified for (5.13), we get zero determinant and therefore we may be tempted to say that no solution w_1 and w_2 exists. But this is not necessarily true.

Consider that to say that the matrix $\boldsymbol{K}$ is not invertible is equivalent to saying it has a zero determinant, or that it is singular, or that it has a zero eigenvalue. The last statement is written as

$$\boldsymbol{K}\boldsymbol{v} = \lambda\boldsymbol{v}$$

where λ is the eigenvalue, and $\boldsymbol{v}$ is the eigenvector. It is known that a 2×2 matrix has two eigenvalues. In the present case the first eigenvalue is zero since the matrix is not invertible, $\lambda_1 = 0$, and we have

$$\boldsymbol{K}\boldsymbol{v}_1 = \boldsymbol{0}$$

The second eigenvalue is $\lambda_2 = 2\frac{P}{L}$, and we have

$$\boldsymbol{K}\boldsymbol{v}_2 = 2\frac{P}{L}\boldsymbol{v}_2$$

For the zero eigenvalue we substitute

$$\frac{P}{L}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix}\begin{bmatrix} v_{11} \\ v_{21} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

to find that the corresponding eigenvector is $v_{11} = v_{21}$. For instance the vector

$$\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

can be taken as the first eigenvector. Note that this eigenvector represents deflection which is uniform along the length of the wire (both the left-hand side node and the right hand side node moved by the same amount, which by the way in the eigenvector above is 1, but it could be just as well any other real number, including zero). Since the cable in this situation moves without deformation, as a rigid body, we called this displacement pattern the ***rigid body mode***.

For the second eigenvalue we obtain

$$\frac{P}{L}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix}\begin{bmatrix} v_{12} \\ v_{22} \end{bmatrix} = 2\frac{P}{L}\begin{bmatrix} v_{12} \\ v_{22} \end{bmatrix}$$

to find that the corresponding eigenvector is $v_{12} = -v_{22}$. For instance the vector

$$\boldsymbol{v}_2 = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

can be taken as the second eigenvector (again, any other number than 1 would do just as well). Note that this eigenvector represents deflection which is linear along the length of the wire. The left-hand side node moves by the same amount as the right hand side node, but they move in opposite directions.

Now that we have the eigenvectors and eigenvalues at hand, we can return to the question of whether there is any solution even when $\boldsymbol{K}$ is not invertible. The answer is yes, since a solution would exist if the right-hand side was proportional to one of the eigenvectors. For instance taking

$$\boldsymbol{L} = m\lambda_1 \boldsymbol{v}_1 \ ,$$

with m being an arbitrary number, means that (5.12) for this right-hand side

$$\boldsymbol{K}\boldsymbol{w} = \boldsymbol{L} = m\lambda_1 \boldsymbol{v}_1$$

has a solution $\boldsymbol{w} = m\boldsymbol{v}_1$. However, this is equivalent to saying

$$\boldsymbol{K}\boldsymbol{w} = \boldsymbol{0}$$

has a solution $\boldsymbol{w} = m\boldsymbol{v}_1$ since $\lambda_1 = 0$. The mechanics interpretation of this is that the taut wire model can assume equilibrium if there are no transverse forces applied as loads, but at the same time the deflection of the cable may be nonzero provided the node at the left moves by the same amount as the node at the right (that is the first eigenvector shape).

Further, taking

$$\boldsymbol{L} = p\lambda_2 \boldsymbol{v}_2 \ ,$$

with p being an arbitrary number, means that (5.12) for this right-hand side

$$\boldsymbol{K}\boldsymbol{w} = \boldsymbol{L} = p\lambda_2 \boldsymbol{v}_2$$

has a solution $\boldsymbol{w} = p\boldsymbol{v}_2$. The mechanical interpretation is that the forces applied at the ends of the wire are of equal magnitude but opposite direction. They self-equilibrate, and the shape of the wire "follows" the forces. This means that the midpoint of the wire stays put.

The preceding also means that a linear combination of eigenvectors is also a possible right-hand side which results in the solution $\boldsymbol{w} = m\boldsymbol{v}_1 + p\boldsymbol{v}_2$ for the right-hand side vector

$$\boldsymbol{L} = m\lambda_1 \boldsymbol{v}_1 + p\lambda_2 \boldsymbol{v}_2 = p\lambda_2 \boldsymbol{v}_2 \ .$$

In words, a solution exists (some multiple of the second eigenvector, $p\boldsymbol{v}_2$) provided the applied forces are of equal magnitude but opposite orientation (i.e. proportional to the second eigenvector), but the solution is not unique since we can add on an arbitrary multiple of the first eigenvector ($m\boldsymbol{v}_1$). The interpretation from the viewpoint of mechanics is that forces applied at the ends of the wire that are of equal magnitude but opposite direction can be equilibrated as the wire deflects, and that this equilibrium is not going to be perturbed if we shift the cable on its rollers up or down by an arbitrary amount.

In summary, we find that the elementwise stiffness matrix (5.13) is singular. Its zero-eigenvalue eigenvector represents equal displacement at both nodes. A deflection solution exists, provided the right-hand side load vector corresponds to forces of equal magnitude but opposite direction, but this deflection is not unique since we can add to it an arbitrary shift of equal magnitude at both nodes.

5.6 Removing rigid body modes

What about finite element model for the situation in Figure 5.5 with more than one finite element? Consider for instance a mesh with six nodes, numbered so that the equation numbers are identical to the node numbers, and with the five elements of unequal length, also being numbered left to right (Figure 5.6). Will the global stiffness matrix be singular?

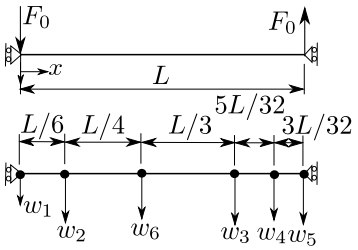


Fig. 5.6. Free-free taut wire and the five-element model

Here we have the global equilibrium relations

$$\frac{P}{15L}\begin{bmatrix} 90, & -90, & 0, & 0, & 0, & 0 \\ -90, & 150, & 0, & 0, & 0, & -60 \\ 0, & 0, & 141, & -96, & 0, & -45 \\ 0, & 0, & -96, & 256, & -160, & 0 \\ 0, & 0, & 0, & -160, & 160, & 0 \\ 0, & -60, & -45, & 0, & 0, & 105 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} F_0 \\ 0 \\ 0 \\ 0 \\ -F_0 \\ 0 \end{bmatrix}$$

We can easily convince ourselves that when we multiply the stiffness matrix

$$\boldsymbol{K} = \frac{P}{15L}\begin{bmatrix} 90, & -90, & 0, & 0, & 0, & 0 \\ -90, & 150, & 0, & 0, & 0, & -60 \\ 0, & 0, & 141, & -96, & 0, & -45 \\ 0, & 0, & -96, & 256, & -160, & 0 \\ 0, & 0, & 0, & -160, & 160, & 0 \\ 0, & -60, & -45, & 0, & 0, & 105 \end{bmatrix} \tag{5.14}$$

with the vector

$$[w_{\mathrm{RBM}}] = m\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

we get a zero vector as a result. An easy check is to notice that multiplying a matrix by a vector of ones means adding together all the columns, and in our case adding together the columns of the stiffness matrix results in a zero vector. Therefore we conclude that the above vector is an eigenvector of the stiffness matrix with a zero associated eigenvalue (i.e. it is a rigid body mode). In other words, the global stiffness matrix for this mesh is singular. It does not matter how many elements are assembled, as long as it is possible to deflect all nodes by the same amount (i.e. by the zero-eigenvalue eigenvector) the stiffness matrix is going to be singular. If the whole wire can move as a rigid body, then each of the finite elements can move as a rigid body. The converse is also true.

How do we make the problem solvable? We may try to remove the singularity of the stiffness matrix, and there are several ways in which this may be achieved. We will be working *directly* with the finite element model. In other words, we are not going to go back to derive a different weighted residual statement to correct the situation, but we will work directly with the discrete equations of the finite element model.

5.6.1 Adding pin support

We could consider adding a pin support at one of the nodes. Under which conditions would this be admissible? We must not change the loads under which the structure operates, that is our basic

concern. Therefore we cannot afford to add an additional force (reaction of the pin). Consequently, we may add a pin support *provided* the associated *reaction* is guaranteed to be *zero*. Then the original conditions are preserved. In the present situation the two applied forces balance each other,

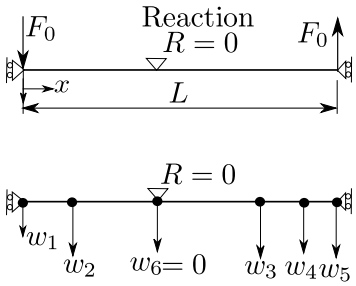


Fig. 5.7. Free-free taut wire and the five-element model with an additional pin support

they are self-equilibrated. Therefore adding a pin support for instance at node 6 is going to be admissible since the reaction will be zero (Figure 5.7). This leads to the partitioning of the stiffness matrix (5.14). Degree of freedom 6 is now prescribed at zero value, which means that the free-free stiffness matrix will be the 5×5 submatrix

$$\boldsymbol{K}_{5\times 5} = \frac{P}{15L}\begin{bmatrix} 90, & -90, & 0, & 0, & 0 \\ -90, & 150, & 0, & 0, & 0 \\ 0, & 0, & 141, & -96, & 0 \\ 0, & 0, & -96, & 256, & -160 \\ 0, & 0, & 0, & -160, & 160 \end{bmatrix}$$

The columns of this matrix no longer sum to a zero vector (since the sixth column is now missing), and the vector

$$[w] = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = m \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

does not give a zero force as it would if it was a zero-eigenvalue eigenvector (rigid body mode). The matrix $\boldsymbol{K}_{5\times 5}$ is no longer singular.

5.6.2 Adding spring support

We might also consider adding a grounded spring (Figure 5.8). Again, we must conclude that provided no force is generated in the spring this would be admissible: no force would be added on the structure. We see that no force will be generated in the spring provided node 6 does not displace and hence the spring length does not change. Therefore the effect on the solution would be exactly as if we added a pin support as in the previous section, but we will get there in a different way: Adding the spring to the finite element model means that degree of freedom 6 is connected to ground by a stiffness coefficient k. The stiffness matrix (5.14) will therefore be changed by this addition to read

$$\boldsymbol{K} = \frac{P}{15L}\begin{bmatrix} 90, & -90, & 0, & 0, & 0, & 0 \\ -90, & 150, & 0, & 0, & 0, & -60 \\ 0, & 0, & 141, & -96, & 0, & -45 \\ 0, & 0, & -96, & 256, & -160, & 0 \\ 0, & 0, & 0, & -160, & 160, & 0 \\ 0, & -60, & -45, & 0, & 0, & 105 + k\frac{15L}{P} \end{bmatrix}$$

which clearly is not singular since $[w_{\mathrm{RBM}}]$ is not an eigenvector. The equilibrium equations

$$\frac{P}{15L}\begin{bmatrix} 90, & -90, & 0, & 0, & 0, & 0 \\ -90, & 150, & 0, & 0, & 0, & -60 \\ 0, & 0, & 141, & -96, & 0, & -45 \\ 0, & 0, & -96, & 256, & -160, & 0 \\ 0, & 0, & 0, & -160, & 160, & 0 \\ 0, & -60, & -45, & 0, & 0, & 105 + k\frac{15L}{P} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} F_0 \\ 0 \\ 0 \\ 0 \\ -F_0 \\ 0 \end{bmatrix}$$

have the solution

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \frac{LF_0}{96P} \begin{bmatrix} 40 \\ 24 \\ -32 \\ -47 \\ -56 \\ 0 \end{bmatrix}$$

Note that $w_6 = 0$ as desired. Evidently here the solution does not depend on the numerical value of the stiffness of the spring k but that is not always the case. Next we show an example which demonstrates that the solution may depend on the spring stiffness.

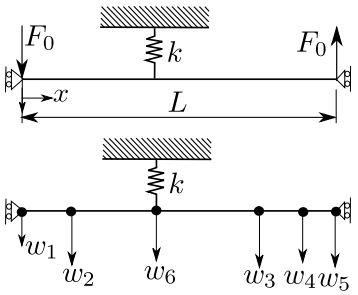


Fig. 5.8. Free-free taut wire and the five-element model with an additional spring

5.7 Using springs to enforce essential boundary conditions

The spring support may be useful for approximate enforcement of deflections at supports. This approach is sometimes referred to as a penalty technique. First we will explore this idea on an example. We will solve for the deflection of the pin-roller cable of Figure 5.9 (situations similar to that of Figure 2.11, but with an added force at the roller).

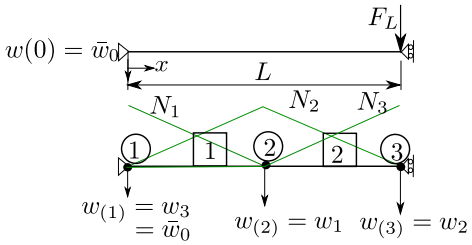


Fig. 5.9. Prestressed wire with prescribed displacement at the pin and force at the roller

Consider Figure 5.10. On the right we're showing the structure with a spring to ground attached at the node at the left-hand side end. For the moment, we shall separate the spring from the structure. The spring will thus connect node 1 to the ground. How do we make the node 1 move by the prescribed amount $w_3 = \bar{w}_0$ downward? In order for the node to move the spring must be stretched, therefore the force in the spring must be

$$C = kw_3 = k\bar{w}_0$$

Since w_3 is given, this *defines* the force C that we need to apply in order to make node 1 deflect. However, node 1 is in fact connected also to the wire. The wire contributes stiffness to node 1, and therefore it is possible that the force needed to deflect node 1 is in fact not equal to $k\bar{w}_0$!

The stiffness matrix without the addition of the spring constant is assembled as shown in (2.33). Adding the spring stiffness to connect node 3 to the ground yields

$$[K] = \frac{2P}{L}\begin{bmatrix} 2\ , & -1\ , & -1 \\ -1\ , & 1\ , & 0 \\ -1\ , & 0\ , & 1+\dfrac{kL}{2P} \end{bmatrix}$$

The loads will be all the forces acting on the structure, which means the force at the roller F_L and C. Here we will assume that we can approximate C as $C = kw_3 = k\bar{w}_0$ even though we know this not to be entirely correct

$$[L] = \begin{bmatrix} 0 \\ F_L \\ C \end{bmatrix} = \begin{bmatrix} 0 \\ F_L \\ k\bar{w}_0 \end{bmatrix}$$

The symbolic inverse of the stiffness matrix is

$$[K]^{-1} = \frac{2P}{L}\begin{bmatrix} \dfrac{L}{2P}+1/k\ , & \dfrac{L}{2P}+1/k\ , & 1/k \\ \dfrac{L}{2P}+1/k\ , & \dfrac{L}{P}+1/k\ , & 1/k \\ 1/k\ , & 1/k\ , & 1/k \end{bmatrix}$$

and the solution therefore reads

$$[w] = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} (\dfrac{L}{2P}+1/k)F_L + \bar{w}_0 \\ (\dfrac{L}{P}+1/k)F_L + \bar{w}_0 \\ (1/k)F_L + \bar{w}_0 \end{bmatrix}$$

We can see that the deflections are not quite right. For instance, we should have $w_3 = \bar{w}_0$ but we got $w_3 = (1/k)F_L + \bar{w}_0$. However, this can be made into a rather good solution, if only we make the stiffness of the spring very large, since

$$\lim_{k\to\infty} w_3 = \lim_{k\to\infty} (1/k)F_L + \bar{w}_0 = \bar{w}_0\ .$$

We say that k is a penalty parameter. The technique using stiff springs to enforce approximately boundary conditions is therefore called the penalty approach. The penalty approach may be used to enforce pin support conditions, with either zero or nonzero prescribed deflections. Figure 5.11 shows

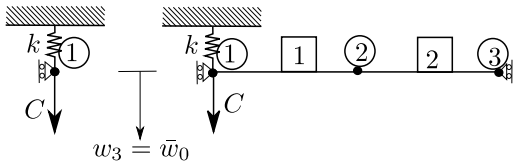


Fig. 5.10. Spring to ground disconnected from the structure (left), and connected to the structure (right).

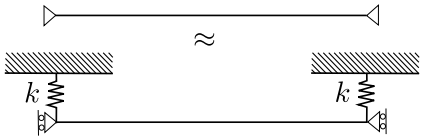


Fig. 5.11. Enforcing essential boundary conditions with a penalty formulation (stiff springs)

the approximate equivalents of the pin-pin supported wire and wire supported by stiff grounded springs (k being very large). The penalty parameter k theoretically can approach infinity, practically there are limits. If we make this parameter too large we will run into computer arithmetic errors so large as to destroy the solution.

The grounded-spring support is in fact a third kind of boundary condition that we can add to our list of pin support and roller support. We envision the spring-support conditions shown in Figure 5.12. The ground connector of the spring is allowed to displace (by $w_{a,0}$ at $x = 0$, and by $w_{a,L}$ at $x = L$) and the end of the spring connected to the wire can displace by $w(0)$ at $x = 0$, and by $w(L)$ at $x = L$. The difference between the displacements can generate a force in the spring, which needs to be in equilibrium with the transverse component of the prestress force at $x = 0$

$$Pw'(0) - k_0(w(0) - w_{a,0}) = 0;$$

and

$$-Pw'(L) - k_L(w(L) - w_{a,L}) = 0;$$

at $x = L$.

These boundary conditions are treated in the context of the weighted residual approach exactly as the natural boundary condition. Let us for definiteness consider static loading of the structure in Figure 5.12 where at either end we have this new boundary condition. We can form the weighted residuals

$$\eta_j(L) r_F(L) = \eta_j(L) \left[-Pw'(L) - k_L(w(L) - w_{a,L})\right] \ ,$$

and

$$\eta_j(0) r_F(0) = \eta_j(0) \left[Pw'(0) - k_0(w(0) - w_{a,0})\right] \ ,$$

and combine them with the balance residual. Similarly to the cancellation in equation (2.12), the terms $-\eta_j(L)Pw'(L)$ and $\eta_j(0)Pw'(0)$ cancel and we obtain the weighted residual statement

$$-\eta_j(0)k_0(w(0) - w_{a,0}) - \eta_j(L)k_L(w(L) - w_{a,L}) - \int_0^L {\eta_j}' Pw' \, \mathrm{d}x + \int_0^L \eta_j q \, \mathrm{d}x = 0 \ , \tag{5.15}$$

or, flipping the sign,

$$+\eta_j(0)k_0(w(0) - w_{a,0}) + \eta_j(L)k_L(w(L) - w_{a,L}) + \int_0^L {\eta_j}' Pw' \, \mathrm{d}x - \int_0^L \eta_j q \, \mathrm{d}x = 0 \ , \tag{5.16}$$

Next we will practice this with a finite element model that incorporates the spring supports.

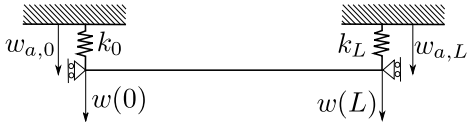


Fig. 5.12. Spring-support boundary conditions

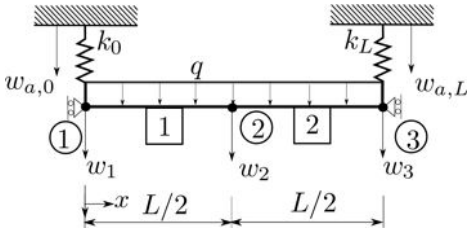


Fig. 5.13. Spring-support boundary conditions

Exercise 24.

Solve for the deflections of the prestressed wire with uniform transverse load and spring supports at either end (Figure 5.13) using a mesh of two finite elements.

Solution: The finite element test $\eta_j = N_{\langle j \rangle}$ and trial function $w = \sum_{i=1}^{3} N_{\langle i \rangle} u_i$ are introduced into the weighted residual equation (5.16) as usual

$$\begin{aligned} &N_{\langle j \rangle}(0) k_0 (\sum_{i=1}^{3} N_{\langle i \rangle}(0) u_i - w_{a,0}) + N_{\langle j \rangle}(L) k_L (\sum_{i=1}^{3} N_{\langle i \rangle}(L) u_i - w_{a,L}) \\ &\quad + \int_0^L N_{\langle j \rangle}{}' P \sum_{i=1}^{N} N'_{\langle i \rangle} u_i \, \mathrm{d}x - \int_0^L N_{\langle j \rangle} q \, \mathrm{d}x = 0 \,, \quad j = 1, 2, 3 \end{aligned} \tag{5.17}$$

The term

$$N_{\langle j \rangle}(0) k_0 (\sum_{i=1}^{3} N_{\langle i \rangle}(0) u_i - w_{a,0})$$

is nonzero only for $j = 1, i = 1$ because $N_{\langle 1 \rangle}(0) = 1$ and $N_{\langle j=2,3 \rangle}(0) = 0$ and we get

$$N_{\langle 1 \rangle}(0) k_0 (\sum_{i=1}^{3} N_{\langle i \rangle}(0) u_i - w_{a,0}) = k_0 (w_1 - w_{a,0}) \,.$$

The term $k_0 w_1$ will contribute to the stiffness matrix, and $-k_0 w_{a,0}$ will contribute to the load vector. Similarly, the term

$$N_{\langle j \rangle}(L) k_L (\sum_{i=1}^{3} N_{\langle i \rangle}(L) u_i - w_{a,L})$$

is nonzero only for $j = 3, i = 3$ because $N_{\langle 3 \rangle}(L) = 1$ and $N_{\langle j=1,2 \rangle}(L) = 0$ and we get

$$N_{\langle j \rangle}(L) k_L (\sum_{i=1}^{3} N_{\langle i \rangle}(L) u_i - w_{a,L}) = k_L (w_3 - w_{a,L}) \,.$$

The term $k_0 w_3$ will contribute to the stiffness matrix, and $-k_L w_{a,L}$ will contribute to the load vector.

The remaining two terms are standard and are treated with the assembly of elementwise quantities. In this way we get the stiffness matrix

$$[K] = \frac{2P}{L}\begin{bmatrix} 1+\dfrac{k_0 L}{2P}\,, & -1\,, & 0 \\ -1\,, & 2\,, & -1 \\ 0\,, & -1\,, & 1+\dfrac{k_L L}{2P} \end{bmatrix}$$

where we note the presence of the spring stiffnesses, and the load vector

$$[F] = \begin{bmatrix} \dfrac{qL}{4} + k_0 w_{a,0} \\ \dfrac{qL}{2} \\ \dfrac{qL}{4} + k_L w_{a,L} \end{bmatrix} .$$

To make the symbolic algebra more manageable, we will assign the springs the same properties, and we will express the stiffness of the springs as a multiple of the basic stiffness of the wire

$$k_0 = \alpha\frac{P}{L}\,, \quad k_L = \alpha\frac{P}{L}\,.$$

Here the multiplier α is nondimensional. Hence the stiffness matrix neatly simplifies to

$$[K] = \frac{2P}{L}\begin{bmatrix} 1+\dfrac{\alpha}{2}\,, & -1\,, & 0 \\ -1\,, & 2\,, & -1 \\ 0\,, & -1\,, & 1+\dfrac{\alpha}{2} \end{bmatrix}$$

and the load vector may be put as

$$[F] = \begin{bmatrix} \dfrac{qL}{4} + \alpha\dfrac{P}{L} w_{a,0} \\ \dfrac{qL}{2} \\ \dfrac{qL}{4} + \alpha\dfrac{P}{L} w_{a,L} \end{bmatrix} .$$

Symbolic algebra allows us to express the inverse of the stiffness matrix as

$$[K]^{-1} = \frac{L}{2P}\begin{bmatrix} 1/(\alpha+2)+1/\alpha, & 1/\alpha, & 1/\alpha - 1/(\alpha+2) \\ 1/\alpha, & 1/\alpha + 1/2, & 1/\alpha \\ 1/\alpha - 1/(\alpha+2), & 1/\alpha, & 1/(\alpha+2)+1/\alpha \end{bmatrix} .$$

For clarity we will split the load vector into two load cases. The first load case is due to the transverse load q

$$[F_q] = \begin{bmatrix} \dfrac{qL}{4} \\ \dfrac{qL}{2} \\ \dfrac{qL}{4} \end{bmatrix}$$

and the second load case brings in the prescribed deflections of the grounded ends of the springs

$$[F_w] = \begin{bmatrix} \alpha \dfrac{P}{L} w_{a,0} \\ 0 \\ \alpha \dfrac{P}{L} w_{a,L} \end{bmatrix} .$$

The nodal deflections for the first load case are obtained as

$$[w_q] = [K]^{-1}[F_q] = \begin{bmatrix} \dfrac{qL^2}{2P\alpha} \\ \dfrac{qL^2}{8P} + \dfrac{qL^2}{2P\alpha} \\ \dfrac{qL^2}{2P\alpha} \end{bmatrix} .$$

It is instructive to split this solution into two parts

$$[w_q] = \begin{bmatrix} 0 \\ \dfrac{qL^2}{8P} \\ 0 \end{bmatrix} + \frac{qL^2}{2P\alpha} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} .$$

The first part corresponds to the deflection of a pin-pin supported wire, and the second part is the rigid body motion of the wire due to the extension of the springs under the total force qL

$$\frac{qL}{k_0 + k_L} = \frac{qL}{\alpha \frac{P}{L} + \alpha \frac{P}{L}} = \frac{qL^2}{2P\alpha} .$$

The deflection for the second case is

$$[w_w] = [K]^{-1}[F_w] = \begin{bmatrix} w_0 - \dfrac{w_0 - w_L}{\alpha + 2} \\ \dfrac{w_0 + w_L}{2} \\ w_L + \dfrac{w_0 - w_L}{\alpha + 2} \end{bmatrix} .$$

We can see that we could use the spring supports as an approximate way of enforcing pin-support conditions. If we make the springs very stiff, $k_0 \to \infty$ and $k_L \to \infty$, which in our case is achieved by $\alpha \to \infty$, we get the correct behavior of the solutions

$$\lim_{\alpha \to \infty} [w_q] = \lim_{\alpha \to \infty} \begin{bmatrix} 0 \\ \dfrac{qL^2}{8P} \\ 0 \end{bmatrix} + \frac{qL^2}{2P\alpha} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ \dfrac{qL^2}{8P} \\ 0 \end{bmatrix} ,$$

and

$$\lim_{\alpha \to \infty} [w_w] = \lim_{\alpha \to \infty} \begin{bmatrix} w_0 - \dfrac{w_0 - w_L}{\alpha + 2} \\ \dfrac{w_0 + w_L}{2} \\ w_L + \dfrac{w_0 - w_L}{\alpha + 2} \end{bmatrix} = \begin{bmatrix} w_0 \\ \dfrac{w_0 + w_L}{2} \\ w_L \end{bmatrix} .$$

Practically we would choose α just large, not infinite. Then the solutions would be slightly in error. For instance choosing $\alpha = 10^7$ would produce approximately six correct digits in the resulting deflections.

6

Statics and Dynamics of Taut Wire with the FEM toolbox

In this chapter we will introduce a finite element library (toolbox), FAESOR[1]. It will be used to produce finite element solutions using the results of the previous chapters for the Galerkin finite element method.

FAESOR is written for Matlab, and its design is based on the object oriented support in Matlab. In particular, all the methods and algorithms are present in the library as methods defined for classes.

The toolbox is available for download at

http://hogwarts.ucsd.edu/~pkrysl/FAESOR .

Expand the zip archive anywhere you like. The result will be a hierarchy of folders starting with one called FAESOR. Then start Matlab. Make FAESOR your working folder within the Matlab environment, and once in there run the initialization function FAESOR_init. This will inform Matlab where to find classes, utilities, and examples from the toolbox. For those using a Windows environment: one may start Matlab and run FAESOR_init at the same time by double-clicking the start.bat batch file. For details and other information, please refer to the README file.

Matlab is adequately described in the online documentation [Matlab]. The "Getting started" tutorial is especially helpful for beginners. Also, Moler has published an accessible book about numerical analysis and Matlab [M04], including a brief tutorial on the use of Matlab. The book can also serve as reference for any numerical analysis issues that are touched upon in this book.

The Matlab online documentation [Matlab] has also a well-written discussion of the Matlab object-oriented features (under "Matlab"/"Programming"/"Classes and Objects"). The FAESOR toolbox employs only the most basic class-based facilities of the language, and the design of the toolbox is quite simple.

6.1 Statics: uniform load

In this example, we assume the transverse load q is uniform, and the transverse force is absent, $F_L = 0$. The Matlab script implementing the solution is w1 [2]. It starts with the definition of the variables.

```
0001 disp('Taut wire: example 1-- statics, uniform load');
0002 L=6; % length of span
0003 P=4; % pre-stress force
0004 q =-0.1;% distributed load
```

Next, the mesh is defined: an array of nodes is created, with node 1 at $x = 0$ and so on. The function fenodeset is the constructor of the class fenodeset, and the attributes are being passed as fields of a struct, as pairs "name, value" (for instance, 'id',(1:n+1)'): this approach is uniformly adopted

[1] FAESOR is © 2005-2010, Petr Krysl

[2] Folder: FAESOR/examples/taut_wire

for all constructors in FAESOR. The object gcells collects information about the finite elements, which are here of the type "line element with two nodes" (L2). The object gcells is a set of elements of class gcellset_L2. The attribute conn is the connectivity array: the numbers of nodes that are connected by the element, one row per element. The finite elements are referred to as ***geometric cells***. The main reason is that the finite elements by themselves have rather limited responsibilities in FAESOR, namely calculation of the basis functions (and their derivatives), and drawing of the shape of the cell are essentially all that is required. The finite elements represent individual pieces of the computational domain, each a little bit of the geometric extent of the domain: a geometric cell. Here we create the set of nodes and the set of elements, numbered from left to right.

```
0005 n=2; % number of elements
0006 % Mesh
0007 % finite element node set
0008 fens=fenodeset(struct ('id',(1:n+1)','xyz',linspace(0, L, n+1)'));
0009 % set of geometric cells
0010 gcells = gcellset_L2(struct('id',1,'conn',[(1:n)', (2:n+1)']));
```

The operations on the mesh that reflect the particular problem that is being solved are encapsulated in a class descended from the so-called finite element block class, feblock. In particular, the prestressed wire stiffness and mass matrix, the effect of the distributed load, q, and the effect of the nonzero displacement at $x = 0$, are computed by the methods of the class feblock_defor_taut_wire. Note that Simpson's 1/3 rule is being used for the numerical quadrature. The finite element block consists of all the finite elements in the array gcells.

```
0011 % Finite element block
0012 feb = feblock_defor_taut_wire(struct ('mater',mater,...
0013     'gcells',gcells,...
0014     'integration_rule',simpson_1_3_rule,...
0015     'P',P));
```

The quantities that are interpolated on the finite element mesh, such as the transverse displacement of the wire, w, or the geometry of the mesh, are represented in FAESOR as instances of the class field. The field geom records the geometry of the mesh. The constructor on line 0017 retrieves the nodal coordinates from the array of the nodes. The dimension of the field is 1 because each degree of freedom is just a single displacement. On line 0019 we define the field of the transverse displacements, w. For convenience, it is defined by cloning the geom field, and then zeroing out all the degrees of freedom.

```
0016 % Geometry
0017 geom = field(struct ('name',['geom'], 'dim', 1, 'fens',fens));
0018 % Define the displacement field
0019 w   = 0*clone(geom,'w');
```

Next, the displacement (essential) boundary conditions are defined, and applied to the displacement field. The method set_ebc only makes a note in the field which components of the degrees of freedom, at which node, are being prescribed (or released), and to which value they are being prescribed. The method apply_ebc is then used to transfer this information to the actual degrees of freedom. Finally, the method numbereqns numbers the degrees of freedom that are not being prescribed, effectively assigning each one a global equation number.

```
0020 % Apply EBC's
0021 fenids=[1]; prescribed=[1]; component=[1]; val=0;
0022 w   = set_ebc(w, fenids, prescribed, component, val);
0023 w   = apply_ebc (w);
0024 % Number equations
0025 w   = numbereqns (w);
```

An important remark should be made here: Lines 0022, 0023, and 0025 illustrate a design feature of Matlab, where all arguments are passed by value. Therefore, no matter what we do with the arguments inside the functions, the variables that were passed by the caller into those functions do not change at all. The functions work with copies, not the actual variables that the caller passed. If the caller wishes to change the variables, the method must return the changed value, and the caller must assign this value. Example: on line 0031 the method `numbereqns` will number the equations in a copy of the field `w`, and then will return the copy. Since we assign back to the field `w`, the computed numbering of the equations will be now available in `w`; if we did not assign back to `w`, all the work done by the method `numbereqns` would be forgotten.

Next we create the global system of equations. The stiffness matrix is an object, `K`, which gets created in line 0027, and initialized to represent a dense `neqns`×`neqns` matrix. In line 0028, the stiffness matrices calculated for each finite element by the block `feb` using the method `stiffness` are assembled into the global matrix `K` (class `dense_sysmat`).

```
0027 K = start (dense_sysmat, get(w, 'neqns'));
0028 K = assemble (K, stiffness(feb, geom, w));
```

The number of equations is retrieved from the displacement field (the global equation numbers have been placed there above) using the `get` method. The `get` method is available for all `FAESOR` objects, and just typing `w` at the command line produces a list of all the properties that can be obtained from the object. A great way to explore `FAESOR` objects is the graphical user interface of the object browser, `OBgui` (part of `FAESOR`). Its operation is supported by the output of `get(w)` (notice that only the object itself is passed as argument): using `get` in this way returns a cell array, with the name and the description of each attribute.

```
>> w
w =
   field object:
=== Gettable properties: ========================
name   -   name of the field, character array
dim   -   dimension of the nodal parameters, scalar
nfens   -   number of nodes associated with the field, scalar
eqnums   -   equation numbers, array [nfens,dim]
neqns   -   number of equations (i.e. total number of free components), scalar
values   -   values of the nodal parameters, array [nfens,dim]
is_prescribed   -   is the component of a nodal parameter prescribed or is it
             free? (0=free, 1=prescribed), array [nfens,dim]
prescribed_values   -   prescribed values of the nodal
             parameters, array [nfens,dim]
=== Settable properties: ========================
   -
```

The load q is in this case represented by the `body_load` class. The global load object `sysvec` is assembled from element load vectors computed by the finite element block.

```
0030 fi = force_intensity(struct ('magn',q));
0031 F = start (sysvec, get(w, 'neqns'));
0032 F = assemble (F, body_loads(feb, geom, w, fi));
```

Finally, the global stiffness object is asked to produce the actual stiffness array (plain two-dimensional Matlab matrix), and the global load object is asked to supply the actual load vector array (Matlab column matrix). The standard backslash Matlab operator then computes the solution, which is stored in the proper places in the displacement field `w`. The method `scatter_sysvec` distributes the system vector (the solution of the system of linear equations) to the proper degrees of freedom, and we should note that again the result is assigned to `w`.

```
0034 w = scatter_sysvec(w, get(K,'mat')\get(F,'vec'));
```

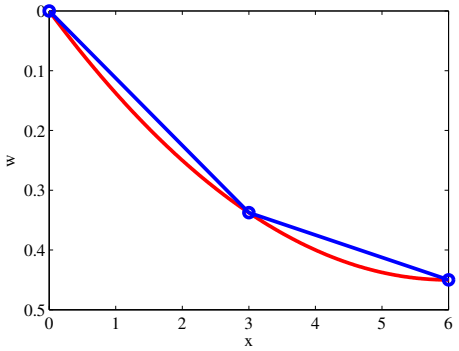


Fig. 6.1. Deflection curve of the taut wire; the exact solution curve, and the approximation by the broken line

A graphical presentation is generated by plotting the linear coordinate (`gather`ed from the geometry field `geom`) versus the linear interpolation of the approximate displacements (`gather`ed from `w`). The analytical solution is also plotted (line `0037`). It is noteworthy that the approximate solution interpolates the analytical solution, but such behavior is peculiar to the one-dimensional problem and does not occur for the multi-dimensional models discussed later in the book.

```
0036 xs= (0:0.01:L);
0037 plot (xs, -q/P*xs.*(xs/2-L),'r-','linewidth', 3);
0038 hold on
0039 plot (gather(geom, (1:n+1),'values'), ...
0040     gather(w, (1:n+1),'values'),'bo-','linewidth', 3);
0041 left_handed_axes;% Plotting is consistent with the textbook sign convention
```

6.2 Sparse matrices

Structural engineers nowadays deal almost daily with results produced by models which are much larger than the ones encountered so far in this book. Structural analysis programs, or more generally finite element analysis programs, work on a regular basis with models where one million unknowns is not uncommon. In recent years there have been reports of successful analyses with billions of unknowns (simulation of seismic events).

The first prerequisite of a successful algorithm for such substantial linear algebra problems is ***sparse matrix*** technology. Consider this toy problem: a taut wire finite element model with seven elements and eight nodes, numbered from left to right. The stiffness matrix (before partitioning) looks like this

```
K =
[  e1,     -e1,       0,       0,       0,       0,       0,   0]
[ -e1, e1 + e2,     -e2,       0,       0,       0,       0,   0]
[   0,     -e2, e2 + e3,     -e3,       0,       0,       0,   0]
[   0,       0,     -e3, e3 + e4,     -e4,       0,       0,   0]
[   0,       0,       0,     -e4, e4 + e5,     -e5,       0,   0]
[   0,       0,       0,       0,     -e5, e5 + e6,     -e6,   0]
[   0,       0,       0,       0,       0,     -e6, e6 + e7, -e7]
[   0,       0,       0,       0,       0,       0,     -e7,  e7]
```

when the degrees of freedom are also numbered left to right. (The element matrices are indicated with `e1`, ...,`e7`.) This is a sparse matrix: there are 64 numbers in the square array, and only 22 of

them are nonzero. This matrix would be classified as ***banded*** as the nonzero elements are to be found in a band around the main diagonal.

The matrix may be treated as if all numbers in it were actually nonzero (the so-called ***dense matrix***). Then one would have to store 64 numbers: all the numbers are stored in a two-dimensional table. Or, if it is treated as sparse, only 22 numbers need to be stored. Considerable savings may be achieved in practice by storing the matrix as sparse.

Figure 6.2 displays a finite element model with over 2000 unknowns. A small model, it can be handled comfortably on a reasonably equipped laptop, yet it will serve us well to illustrate some of the aspects of the so-called large-scale computing algorithms of which we need to be aware. If the stiffness matrix for this model was stored as dense, the memory required would be $2000^2 \times 8 \times 2^{-20} \approx 30.5$ MB (we are assuming double precision, eight bytes per number). Since there are only approximately 131,000 non-zeros in the matrix, storing it as a sparse matrix requires only $131000 \times 8 \times 2^{-20} \approx$ 1.0 MB. Considerable saving. Sometimes storing a dense matrix would be actually impossible. Let us say 1 million rows and columns: the required memory is $(10^6)^2 \times 8 \times 2^{-30} \approx 7450$ GB. Can your desktop handle this?

Furthermore we may note that in many analyses we work with symmetric matrices. Considerable savings are possible then. Take the LU factorization of a symmetric matrix

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U}$$

Now it is possible to factor $\boldsymbol{U}$ by dividing the rows with its diagonal elements, so that we can write $\boldsymbol{U}$ as the product of the diagonal $\boldsymbol{D} = \texttt{diag}(\texttt{diag}(\boldsymbol{U}))$ (expressed in MATLAB notation) with the matrix $\widehat{\boldsymbol{U}}$

$$\boldsymbol{U} = \boldsymbol{D}\widehat{\boldsymbol{U}}$$

Since we must have $\boldsymbol{A} = \boldsymbol{A}^T$, substituting

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{U} = \boldsymbol{L}\boldsymbol{D}\widehat{\boldsymbol{U}}$$

implies that $\widehat{\boldsymbol{U}} = \boldsymbol{L}^T$. Therefore for symmetric $\boldsymbol{A}$ we can make one more step from the LU factorization to the LDLT factorization

$$\boldsymbol{A} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$$

This saves both time (we don't have to compute $\boldsymbol{U}$) and space (we don't have to store $\boldsymbol{U}$).

The figure shows a tuning fork. This one sounds approximately the note of A (440 Hz, international "concert pitch"). To find this vibration frequency, we need to solve an eigenvalue problem (in our terminology, the free vibration problem).

The dynamic matrix (which couples together the stiffness and the mass matrix) is of dimension of roughly 2000×2000. However, not all 4 million numbers are nonzero. Figure 6.3 illustrates this by displaying the nonzeros as black dots (the zeros are not shown). The code to get an image like this for the matrix `A` is as simple as

```
spy(A)
```

Where do the unknowns come from? The vibration model describes the motion of each node (that would be the corners and the midsides of the edges of the tetrahedral T6 finite elements which constitute the mesh of the tuning fork). At each node we have three displacements. Through the stiffness and mass of each of the tetrahedra the nodes which are connected by the tetrahedra are dynamically coupled (in the sense that the motion of one node creates forces on another node). All these coupling interactions are recorded in the dynamic matrix $\boldsymbol{A}$. If an unknown displacement j at node K is coupled to an unknown displacement k at node M, there will be a nonzero element A_{jk} in the dynamic matrix. If we do not care how we number the individual unknowns, the dynamic matrix may look for instance as shown in Figure 6.3: there are some interesting patterns in the matrix, but otherwise the distribution of the connections seems to be pretty much random.

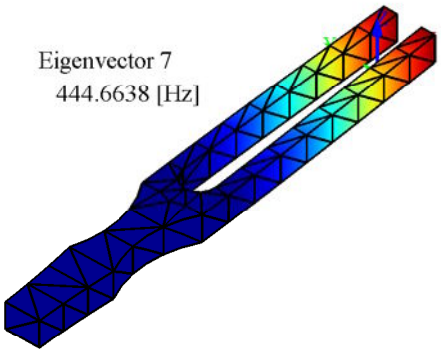


Fig. 6.2. Tuning fork finite element mesh.

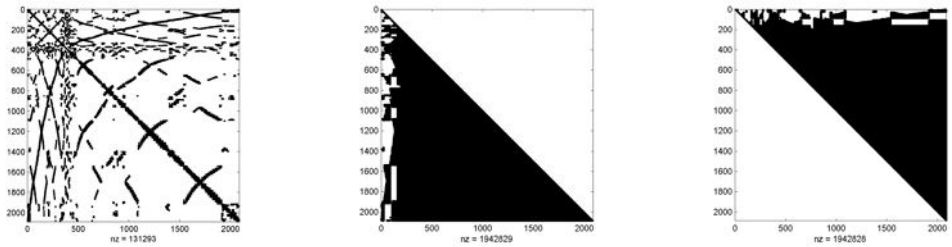


Fig. 6.3. The structure of the tuning fork dynamic matrix. Left to right: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{L}$, $\boldsymbol{U}$. Original numbering of the unknowns. The black dots represent non-zeros, zeros are not shown.

An important aspect of working with large matrices is that as a rule only the non-zeros in matrices will be stored. The matrices will be stored as ***sparse***. A sparse matrix has a more complicated storage, since only the non-zeros are kept, and all the zeros are implied (not stored, but when we ask for an element of the matrix that is not in storage, we will get back a zero). This may mean considerable savings for matrixes that hold only a very small number of non-zeros.

The reason we might want to worry about how the unknowns are numbered lies in the way the LU factorization works. Remember, we are removing non-zeros below the diagonal by combining rows. That means that if we are eliminating element km, we are adding a multiple of the row k and the row m. If the row m happens to have non-zeros to the right of the column m, all those non-zeros will now appear in row k. In this way, some of the zeros in a certain envelope around the diagonal will become non-zeros during the elimination. This is clearly evident in Figure 6.3, where we can see almost entirely black (non-zero) matrices $\boldsymbol{L}$ and $\boldsymbol{U}$. Why is this a problem? Because there are a lot more non-zeros in the LU factors than in the original matrix $\boldsymbol{A}$. The more numbers we have to operate on, the more it costs to factorize the matrix, and the longer it takes. Also, all the non-zeros need to be stored, and to update a sparse matrix with additional non-zeros is very expensive.

The appearance of additional non-zeros in the matrix during the elimination is called ***fill-in***. Fortunately, there are ways in which the fill-in may be minimized by carefully numbering coupled unknowns. Figure 6.4 and Figure 6.5 visualize the dynamic matrix and its factors for two different renumbering schemes: the reverse Cuthill-McKee and symmetric approximate minimum degree permutation. The matrix $\boldsymbol{A}$ holds the same number of non-zeros in all three figures (original numbering, and the two renumbered cases). However the factors in the renumbered cases hold about 10 times less non-zeros than in the original factors. This may be significant. Recall that for a dense matrix the cost of factorization scales as $O(N^3)$. For a sparse matrix with a nice numbering which will limit the fill-in to say 100 elements per row, the cost will scale as $O(100 \times N^2)$. For $N = 10^6$ this will be the difference between having to wait for the factors for one minute or for a full week.

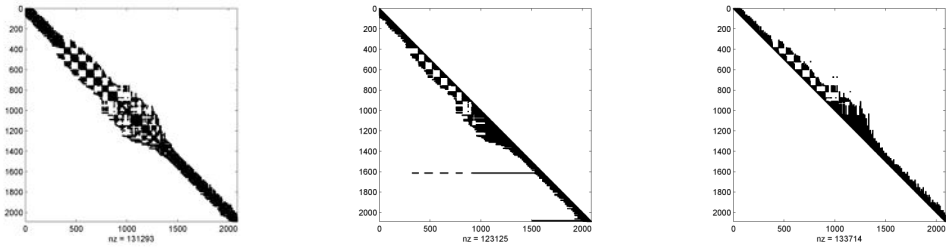


Fig. 6.4. The structure of the tuning fork dynamic matrix. Left to right: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{L}$, $\boldsymbol{U}$. Renumbering of the unknowns with `symrcm`. The black dots represent non-zeros, zeros are not shown.

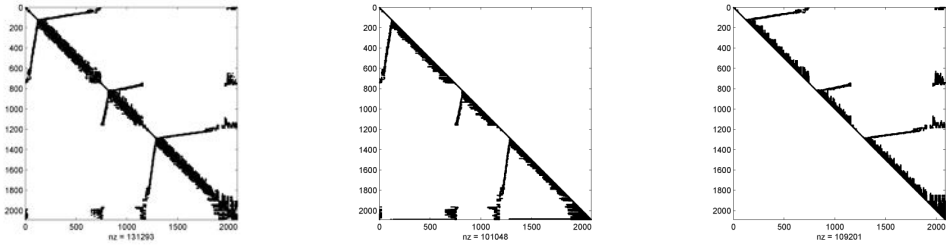


Fig. 6.5. The structure of the tuning fork dynamic matrix. Left to right: $\boldsymbol{A} = \boldsymbol{LU}$, $\boldsymbol{L}$, $\boldsymbol{U}$. Renumbering of the unknowns with `symamd`. The black dots represent non-zeros, zeros are not shown.

As a last note on the subject we may take into account other techniques of solving systems of linear algebraic equations than factorization. There is a large class of iterative algorithms, a line up starting with Jacobi and Gauss-Seidel solvers and currently ending with the so-called multi-grid solvers. These algorithms are much less sensitive to the numbering of the unknowns. In this book we do not discuss these techniques, but refer for instance to Trefethen, Bau [TB97] for an interesting story on current iterative solvers. They are becoming ubiquitous in commercial softwares, hence we better know something about them.

6.3 Free vibration

Numerical quadrature rules can be used for various, sometimes surprising, gains. Either efficiency, or accuracy, or, amazingly, both efficiency and accuracy, may be obtained by deliberately using inaccurate numerical quadrature rule.

Exercise 25. Compute the elementwise mass matrix of the L2 element with uniform mass density using the trapezoidal numerical integration rule (see Table 4.1).

Solution: To compute the component

$$M_{11}^{(e)} = \int_{x_K}^{x_M} N_{\langle 1\rangle} \mu N_{\langle 1\rangle} \, \mathrm{d}x = \int_{x_K}^{x_M} N_K \mu N_K \, \mathrm{d}x$$

of the elementwise mass matrix (see Figure 2.10) using the trapezoidal quadrature rule we first convert the integral from the x domain to the standard interval

$$M_{11}^{(e)} = \int_{x_K}^{x_M} N_K \mu N_K \, \mathrm{d}x = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_K(\xi) \mu N_K(\xi) \mathrm{d}\xi$$

and then we apply the numerical integration formula with the data of Table 4.1. The integrand is

$$\xi_1 = -1 : \quad N_K(\xi_1) \mu N_K(\xi_1) = 1 \times \mu \times 1 = \mu$$

$$\xi_2 = +1: \quad N_K(\xi_2)\mu N_K(\xi_2) = 0 \times \mu \times 0 = 0$$

and we get

$$M_{11}^{(e)} = \frac{x_j - x_i}{2}(\mu \times W_1 + 0 \times W_2) = \frac{(x_j - x_i)\mu}{2}$$

Similarly we obtain

$$M_{12}^{(e)} = \int_{x_K}^{x_M} N_K \mu N_M \, \mathrm{d}x = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_K(\xi)\mu N_M(\xi)\mathrm{d}\xi$$

The integrand is

$$\xi_1 = -1: \quad N_K(\xi_1)\mu N_M(\xi_1) = 0 \times \mu \times 1 = 0$$

$$\xi_2 = +1: \quad N_K(\xi_2)\mu N_M(\xi_2) = 0 \times \mu \times 0 = 0$$

and we get

$$M_{12}^{(e)} = \frac{x_j - x_i}{2}(0 \times W_1 + 0 \times W_2) = 0$$

(and therefore also $M_{21}^{(e)} = M_{12}^{(e)} = 0$). Finally

$$M_{22}^{(e)} = \int_{x_K}^{x_M} N_M \mu N_M \, \mathrm{d}x = \frac{x_j - x_i}{2} \int_{-1}^{+1} N_M(\xi)\mu N_M(\xi)\mathrm{d}\xi$$

and

$$\xi_1 = -1: \quad N_M(\xi_1)\mu N_M(\xi_1) = 0 \times \mu \times 0 = 0$$

$$\xi_2 = +1: \quad N_M(\xi_2)\mu N_M(\xi_2) = 1 \times \mu \times 1 = \mu$$

and we get

$$M_{22}^{(e)} = \frac{x_j - x_i}{2}(0 \times W_1 + \mu \times W_2) = \frac{(x_j - x_i)\mu}{2}$$

To summarize, we get the elementwise mass matrix

$$[M]^{(e)} = \begin{bmatrix} \int_{x_K}^{x_M} N_K \mu N_K \, \mathrm{d}x & \int_{x_K}^{x_M} N_K \mu N_M \, \mathrm{d}x \\ \int_{x_K}^{x_M} N_M \mu N_K \, \mathrm{d}x & \int_{x_K}^{x_M} N_M \mu N_M \, \mathrm{d}x \end{bmatrix} = \frac{\mu(x_M - x_K)}{2} \begin{bmatrix} 1, 0 \\ 0, 1 \end{bmatrix} \tag{6.1}$$

This is commonly referred to as the ***lumped mass matrix***. This terminology reflects that each of the nodes is getting one half of the total mass of the element $\frac{\mu(x_M - x_K)}{2}$, and practically this results in the mass of the wire being concentrated (lumped) into massive particles at the nodes.

We will look at the properties of solutions that obtain with different mass matrices now. Compare the consistent elementwise mass matrix of (3.16) with the lumped mass matrix (6.1). The main distinguishing feature is that the lumped mass matrix is *diagonal*. Diagonal matrices result in the most efficient operations. There are also implications as far as accuracy is concerned, and it is not bad news.

The free-vibration problem introduced in Section 3.7 will be solved for a simply-supported taut wire of uniform mass density. The Matlab script `wvib`[3] obtains the solution for a series of progressively finer and finer meshes (from 8 elements to 1024 elements). The mass matrix is either assembled

[3]Folder: `FAESOR/examples/taut_wire`

from the formula (3.16) (this is the so-called ***consistent mass matrix***), or it is the lumped mass matrix of (6.1). Note that we use a method of the finite element block `feblock`, `lumped_hrz` to compute the lumped mass matrices from the consistent ones.

The analytical formula for the natural frequency [G91]

$$\omega^2 = \frac{P}{\mu}\left(\frac{n\pi}{L}\right)^2 \quad , \quad n = 1, 2, 3, ...$$

produces reference values which are compared with the frequencies solved for from (3.21). The results are summarized in Fig. 6.6. The progressive reduction of the normalized error of the angular natural frequencies

$$e_\omega = \frac{\omega_h - \omega}{\omega}$$

(ω_h is the finite element result, and ω is the reference analytical result) due to the use of more and more elements is called ***convergence***. It may be observed that the two mass matrix formulations lead to convergence from different sides: consistent mass matrix overestimates the natural frequencies, while the lumped mass matrix underestimates them. As to the accuracy: clearly, for quite reasonable engineering tolerance of 5% error, it takes 32 elements for either formulation of the mass matrix to compute all five natural frequencies within the tolerance.

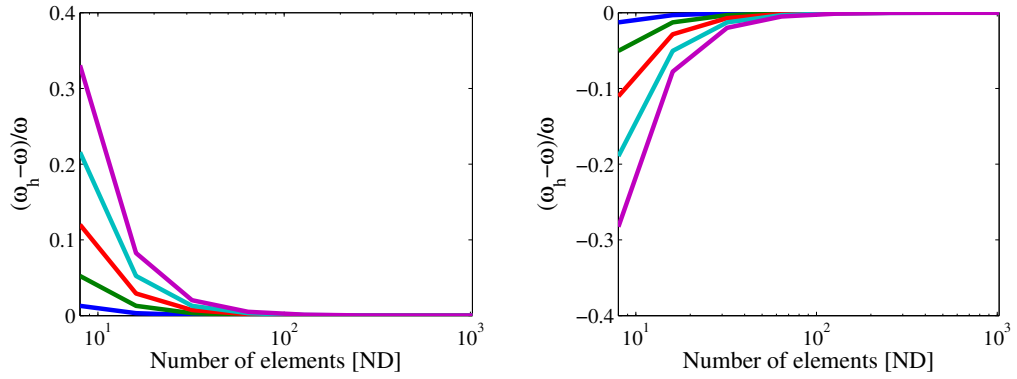


Fig. 6.6. Convergence of the first five natural frequencies of vibration; left: consistent mass matrix, right: lumped mass matrix. Vertical axis: normalized error $(\omega_h - \omega)/\omega$.

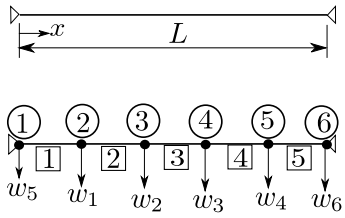


Fig. 6.7. Simply-supported prestressed cable, free vibration.

Exercise 26.

Solve the free vibration IBVP of a pin-pin supported prestressed cable with uniform mass density using Galerkin finite element method with 5 equal-length finite elements of the L2 type. Use the lumped mass matrix formulation.

Solution: Note numbering of the nodes, left to right, has nothing to do with which degrees of freedom are known or unknown. This is typically the state of affairs in finite element computations. The number of free degrees of freedom is $N_{\mathrm{f}} = 4$, and the distribution of degrees of freedom is indicated in Figure 6.7. The length of each element is $L/5$.

For variety we will assemble the global stiffness and mass matrix for all degrees of freedom, and then we will partition it to obtain the free-free matrices. Therefore, the elementwise stiffness matrix (2.30) and the elementwise mass matrix (6.1) are assembled into the global stiffness and mass matrix as

$$[K] = \frac{5P}{L}\begin{bmatrix} 2, -1, & 0, & 0, -1, & 0 \\ -1, & 2, -1, & 0, & 0, & 0 \\ 0, -1, & 2, -1, & 0, & 0, & 0 \\ 0, & 0, -1, & 2, & 0, & -1 \\ -1, & 0, & 0, & 0, & 1, & 0 \\ 0, & 0, & 0, -1, & 0, & 1 \end{bmatrix}, \quad \text{and} \quad [M] = \frac{\mu L}{10}\begin{bmatrix} 2,0,0,0,0,0 \\ 0,2,0,0,0,0 \\ 0,0,2,0,0,0 \\ 0,0,0,2,0,0 \\ 0,0,0,0,1,0 \\ 0,0,0,0,0,1 \end{bmatrix}$$

The free-free stiffness and mass matrices are the 4×4 submatrices in the upper left corner, and the free vibration eigenvalue problem is therefore written as

$$\frac{5P}{L}\begin{bmatrix} 2, -1, & 0, & 0 \\ -1, & 2, -1, & 0 \\ 0, -1, & 2, & -1 \\ 0, & 0, -1, & 2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \\ \phi_{4j} \end{bmatrix} = \omega_j^2 \frac{\mu L}{10}\begin{bmatrix} 2,0,0,0 \\ 0,2,0,0 \\ 0,0,2,0 \\ 0,0,0,2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \\ \phi_{4j} \end{bmatrix}$$

where we expect four eigenvalues $\omega_j^2, j = 1, 2, 3, 4$ and four eigenvectors. We write ϕ_{kj} for the $k-$th component of the $j-$th eigenvector.

We use the same trick as in Exercise 17, and write

$$\begin{bmatrix} 2, -1, & 0, & 0 \\ -1, & 2, -1, & 0 \\ 0, -1, & 2, & -1 \\ 0, & 0, -1, & 2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \\ \phi_{4j} \end{bmatrix} = \Lambda_j \begin{bmatrix} 2,0,0,0 \\ 0,2,0,0 \\ 0,0,2,0 \\ 0,0,0,2 \end{bmatrix}\begin{bmatrix} \phi_{1j} \\ \phi_{2j} \\ \phi_{3j} \\ \phi_{4j} \end{bmatrix}$$

were we define

$$\Lambda_j = \frac{L}{5P}\omega_j^2 \frac{\mu L}{10}$$

This eigenvalue problem can be solved with Matlab:

```
>> [V,Lambda] =eig([ 2    -1     0     0;
                    -1     2    -1     0;
                     0    -1     2    -1;
                     0     0    -1     2],2*eye(4))
V =
    0.2629   -0.4253   -0.4253   -0.2629
    0.4253   -0.2629    0.2629    0.4253
    0.4253    0.2629    0.2629   -0.4253
    0.2629    0.4253   -0.4253    0.2629
Lambda =
    0.1910         0         0         0
         0    0.6910         0         0
         0         0    1.3090         0
         0         0         0    1.8090
```

The eigenvalues are on the diagonal of the matrix `Lambda`. Which means that we can solve for the natural frequencies from

$$\omega_j^2 = \Lambda_j \frac{5P}{L}\frac{10}{\mu L}, \quad \Lambda_j = 0.1910, 0.6910, 1.3090, 1.8090$$

The eigenvectors are the columns of the matrix `V`. However, note well that these are only the components of the eigenvectors for the free degrees of freedom. To this we need to attach the prescribed degrees of freedom which are all zero. For instance, the first eigenvector has components

$$\begin{bmatrix} \phi_{11} \\ \phi_{21} \\ \phi_{31} \\ \phi_{41} \\ \phi_{51} \\ \phi_{61} \end{bmatrix} = \begin{bmatrix} 0.2629 \\ 0.4253 \\ 0.4253 \\ 0.2629 \\ 0 \\ 0 \end{bmatrix}$$

For each shape of vibration j we can write

$$w(x,t) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) w_i(t) = \sum_{i=1}^{N} N_{\langle i \rangle}(x) \left(\phi_{ij} \cos(\omega_j t)\right) = \left(\sum_{i=1}^{N} N_{\langle i \rangle}(x)\phi_{ij}\right) \cos(\omega_j t),$$

Inside the parenthesis we have the so-called mode shape $\sum_{i=1}^{N} N_{\langle i \rangle}(x)\phi_{ij}$. These do indeed resemble the sine half-waves that we expect from the analytical solution.

We can also compare the computed angular frequencies to the analytical predictions. For instance, the lowest natural angular frequency is computed by the finite element model as

$$\omega_1 \doteq \frac{\sqrt{0.1910 \times 50}}{L}\sqrt{\frac{P}{\mu}} \doteq \frac{3.090}{L}\sqrt{\frac{P}{\mu}},$$

which compares favorably (in error by less than 2%) with

$$\omega = \frac{\pi}{L}\sqrt{\frac{P}{\mu}}$$

as given by the analytical formula for the lowest frequency.

6.4 Integration of transient motion

In addition to analytical approaches, the system of ordinary differential equations (3.18) may be integrated numerically. In this section, we will apply an off-the-shelf Matlab integrator.

Matlab integrators work with a system of first order differential equations. Therefore, (3.18) will be first converted to this form. For convenience, Eq. (3.18) will be first cast in matrix form

$$\boldsymbol{M}\ddot{\boldsymbol{w}} + \boldsymbol{K}\boldsymbol{w} = \boldsymbol{F} \ . \tag{6.2}$$

where we recognize the mass matrix, the stiffness matrix, and where the loads on the right-hand side are all grouped together in one vector $\boldsymbol{F}$. The vector $\boldsymbol{w}$ collects only the free degrees of freedom.

Defining the velocity vector $\boldsymbol{v} = \dot{\boldsymbol{w}}$, the first order system may be written as

$$\begin{bmatrix} \boldsymbol{1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{M} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{w}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{v} \\ -\boldsymbol{K}\boldsymbol{w} + \boldsymbol{F} \end{bmatrix} \ , \tag{6.3}$$

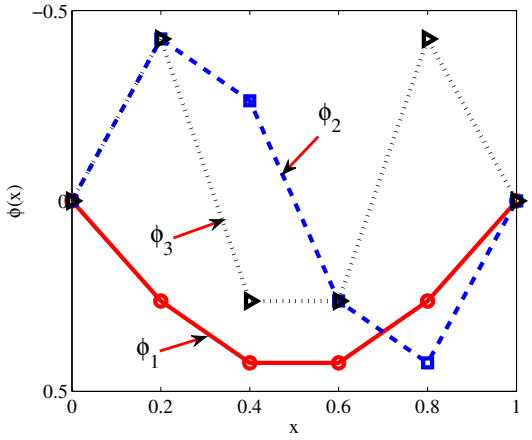


Fig. 6.8. Computed mode shapes 1,2,3

which is in the so-called "mass matrix" form

$$\widetilde{\boldsymbol{M}}\dot{\boldsymbol{y}} = \boldsymbol{f}(t, \boldsymbol{y}) \; . \tag{6.4}$$

The unknown functions of time are both the deflections and the velocities

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{v} \end{bmatrix} \; ,$$

and we define the so-called right-hand-side function as

$$\boldsymbol{f}(t, \boldsymbol{y}) = \begin{bmatrix} \boldsymbol{v} \\ -\boldsymbol{K}\boldsymbol{w} + \boldsymbol{F} \end{bmatrix} \; .$$

In what follows we will consider a particular example of transient vibrations: unforced oscillation due to a particular set of initial conditions. In this example we consider initial zero deflection, and initial velocity which corresponds to a singular perturbation: a single node at the midspan is given a nonzero velocity. This is representative of a physical situation in which a stationary taut string is rapped sharply with a hammer. Waves propagate away from the midpoint, hit the fixed end-points, and complex interference pattern develops (see Fig. 6.9).

6.4.1 Using built-in Matlab solver

The Matlab script `wtransient1`[4] computes the solution to the transient vibration problem using a built-in Matlab solver. As expected, the initial conditions need to be supplied.

```
0024      w0 = gather_sysvec (w);
0025      v0 = w0; v0(round(n/2)) = 1;
```

Further, the system matrices are computed.

```
0027      K = start (dense_sysmat, get(w, 'neqns'));
0028      K = assemble (K, stiffness(feb, geom, w));
0029      Kmat =get(K,'mat');
0031      M = start (dense_sysmat, get(w, 'neqns'));
0032      % Assemble the lumped mass matrix
```

[4]Folder: `FAESOR/examples/taut_wire`

```
0033      M = assemble (M, lumped_hrz(feb,mass(feb, geom, w)));
0034      Mmat=get(M,'mat');
0035      Mattilda= [eye(get(w, 'neqns')),
              zeros(get(w, 'neqns'));...
0036          zeros(get(w, 'neqns')),Mmat];
```

Matlab provides a suite of several ODE solvers: `ode23` is an implementation of an explicit Runge-Kutta (2,3) pair of Bogacki and Shampine [BS89]. These solvers use a standard argument list, with one of the arguments being a function handle to the function that evaluates the right-hand side of (6.4). In our case, this function is written as

```
0038      function rhs =f(t,y)
0039          neq=length(y)/2;
0040          w=y(1:neq); v=y(neq+1:end);
0041          rhs= [v;-Kmat*w];
0042      end
```

The invocation of the solver is unremarkable, except that we use `odeset` to supply the mass matrix, $\widetilde{\boldsymbol{M}}$. The output arguments collect an array of output times, and an array of calculated deflections at nodes with free degrees of freedom.

```
0043      [ts,ys]=ode23(@f,[0, 1500],[w0;v0],...
                    odeset('Mass',Mattilda,'RelTol',1e-3));
```

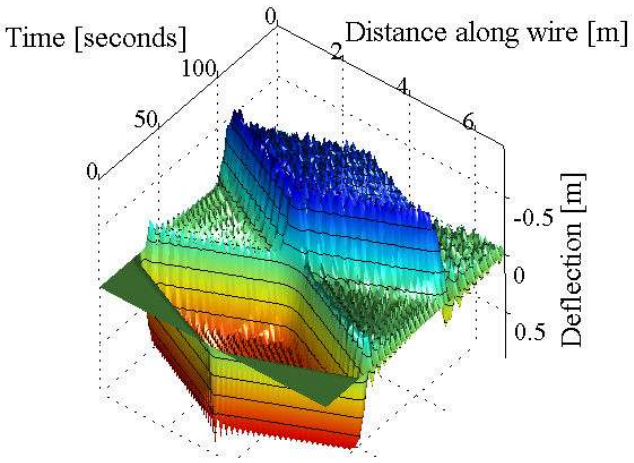


Fig. 6.9. Amplitude of the vertical deflection as a function of the coordinate x and the time.

6.4.2 Using the Trapezoidal integrator

The Runge-Kutta of the previous section seems to be doing an adequate job. However, integrating the second order equations by first converting them to a first order system is sub-optimal: Firstly, the dimension of the ODE system is doubled. Secondly, the Runge-Kutta `ode23` integrator is not actually doing a very good job.

Energy conservation is a very important indicator of the quality of the numerical solution, since energy *should* be conserved in the exact solution to this particular problem. For the taut string problem, the total energy may be defined as

$$TE = \frac{1}{2}\left(\dot{\boldsymbol{w}} \cdot \boldsymbol{M} \cdot \dot{\boldsymbol{w}} + \boldsymbol{w} \cdot \boldsymbol{K} \cdot \boldsymbol{w}\right) \ . \tag{6.5}$$

It may be observed that the solution produced by the `ode23` integrator does not conserve this quantity (refer to Fig. 6.10), and that provides motivation for the development of algorithms specialized to mechanical systems.

The algorithm we are going to develop next is a special form of the well-known Newmark integrator which is very popular and well-respected in the computational mechanics community for a number of reasons [H00]. The starting point is the first order system (6.4). Integrating in time

$$\int_{t_0}^{t} \widetilde{\boldsymbol{M}}\dot{\boldsymbol{y}}\,\mathrm{d}\tau = \int_{t_0}^{t} \boldsymbol{f}(\tau,\boldsymbol{y})\,\mathrm{d}\tau\ , \tag{6.6}$$

yields

$$\widetilde{\boldsymbol{M}}(\boldsymbol{y}_t - \boldsymbol{y}_{t_0}) = \int_{t_0}^{t} \boldsymbol{f}(\tau,\boldsymbol{y})\,\mathrm{d}\tau\ . \tag{6.7}$$

The right-hand side integral may be approximated using the trapezoidal rule, which leads to the algorithm

$$\widetilde{\boldsymbol{M}}(\boldsymbol{y}_t - \boldsymbol{y}_{t_0}) = \frac{t-t_0}{2}\left(\boldsymbol{f}(t,\boldsymbol{y}_t) + \boldsymbol{f}(t_0,\boldsymbol{y}_{t_0})\right)\ . \tag{6.8}$$

Instead of using the first-order form, we may multiply through in (6.8), obtaining a coupled system

$$\begin{aligned} \boldsymbol{w}_t &= \boldsymbol{w}_{t_0} + \frac{t-t_0}{2}\left(\boldsymbol{v}_t + \boldsymbol{v}_{t_0}\right) \\ \boldsymbol{M}\boldsymbol{v}_t &= \boldsymbol{M}\boldsymbol{v}_{t_0} - \frac{t-t_0}{2}\boldsymbol{K}\left(\boldsymbol{w}_t + \boldsymbol{w}_{t_0}\right) + \frac{t-t_0}{2}\left(\boldsymbol{F}_t + \boldsymbol{F}_{t_0}\right)\ , \end{aligned} \tag{6.9}$$

which may be marched forward by substituting the first equation into the second, solving for $\boldsymbol{v}_t$, and then updating $\boldsymbol{w}_t$ from the first equation. The integrator obtained in this way is a special case of the general Newmark integrator [H00]. The Newmark algorithm has two free parameters, and for the special choice $\gamma = 1/2$ and $\beta = 1/4$ (the so-called average acceleration method) one obtains the trapezoidal integrator (6.9).

The Matlab script `wtransient2`[5] computes the solution to the transient vibration problem using a trapezoidal integrator (Newmark average-acceleration integrator). Note that the integrator algorithm is implemented as a nested function, which has access to variables defined in the enclosing environment, the stiffness matrix `Kmat` and the mass matrix `Mmat`.

```
0033      Mmat=get(M,'mat');
0034      % Solve
0035      function [ts,ys] = trapezoidal(nsteps,tspan,w0,v0)
0036          ts= zeros(nsteps, 1);
0037          nu =length(w0);
0038          ys = zeros(2*nu,nsteps);
0039          dt = (tspan(2)-tspan(1))/nsteps;
0040          t =tspan(1);
0041          for i=1:nsteps
0042              ts(i) =t;
0043              ys(1:nu,i) =w0; ys(nu+1:end,i) =v0;
0044              v1=(Mmat+((dt/2)^2)*Kmat)\...
                        ((Mmat-((dt/2)^2)*Kmat)*v0-dt*Kmat*w0);
0045              w1=w0+dt/2*(v0+v1);
0046              w0=w1;v0=v1;
0047              t=t+dt;
0048          end
0049          ys=ys';
```

[5] Folder: `FAESOR/examples/taut_wire`

```
0050      end
0051      %  Now call the integrator
0052      [ts,ys]=trapezoidal(3000,[0, 1500],w0,v0);
```

Figure 6.10 compares the computed total energy for the two integrators introduced in this section. Evidently, the nominally less accurate trapezoidal (Newmark average acceleration) integrator does a much better job than the Runge-Kutta `ode23` integrator.

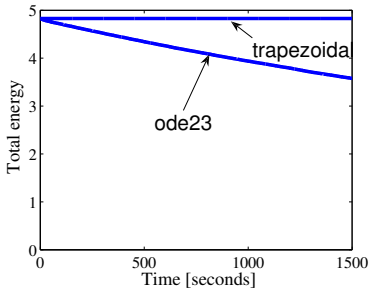


Fig. 6.10. Total energy obtained with two different time integrators.

7

Model of Heat Conduction

In this chapter we will develop a finite element model for heat conduction problems. The excellent thermal analysis textbook by Lienhard and Lienhard [L05] has all the details one might require to supplement the treatment that follows.

7.1 Balance equation

In this section, our goal is to derive the balance equation that describes heat conduction in solids as a partial differential expression. It will be converted to a residual form, which will then be treated with the Galerkin method.

To begin, we pick a control volume, and we keep track of the heat energy within that volume. The control volume may be the whole structure, part of the structure, or just a very small chunk of material surrounding a given point in space (Fig. 7.1). The amount of heat energy in the control volume U is expressed as an integral of the volume density of heat energy, u

$$U = \int_V u \, \mathrm{d}V \tag{7.1}$$

The amount of heat energy within the control volume may change by *outflow* (inflow) of heat

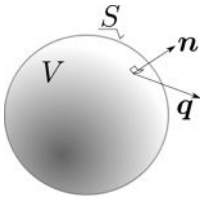


Fig. 7.1. The domain for the heat conduction problem

energy via the boundaries, and *heat generation* (or loss) within the volume. These quantities will be expressed in terms of rates. Therefore, the amount of energy flowing out of the control volume through its bounding surface S per unit time is

$$\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \ , \tag{7.2}$$

where $\boldsymbol{n}$ is the outer normal to the surface S, and $\boldsymbol{q}$ is the heat flux (amount of heat flowing through a unit area per unit time). The amount of energy generated within the control volume per unit time is

$$\int_V Q \, \mathrm{d}V \ , \tag{7.3}$$

where Q is the rate of heat generation per unit volume; for example, heat is released or consumed by various deformation and chemical processes (as work of viscous stresses, reaction product of curing concrete or polymer resins, and so on).

Collecting the terms, we can write for the change of the heat energy within the control volume the rate equation

$$\frac{\mathrm{d}U}{\mathrm{d}t} = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \ . \tag{7.4}$$

Finally, differentiating U with respect to time will be possible if we assume that $U = U(T)$, i.e. if U is a function of the absolute temperature T. Holding the control volume fixed in time, the time differentiation may be taken inside the integral over the volume

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \frac{\mathrm{d}}{\mathrm{d}t} \int_V u \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V \ , \tag{7.5}$$

and with the application of the chain rule, the relationship (7.5) is expressed as

$$\frac{\mathrm{d}U}{\mathrm{d}t} = \int_V \frac{\mathrm{d}u}{\mathrm{d}t} \, \mathrm{d}V = \int_V \frac{\mathrm{d}u}{\mathrm{d}T} \frac{\partial T}{\partial t} \, \mathrm{d}V \ . \tag{7.6}$$

The quantity $c_V = \mathrm{d}u/\mathrm{d}T$ is a characteristic property of a solid material (called specific heat at constant volume). It is typically dependent on temperature, but we will assume that it is a constant; otherwise it leads to nonlinear models.

Substituting, we write

$$\int_V c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = -\int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S + \int_V Q \, \mathrm{d}V \ . \tag{7.7}$$

This equation consists of volume integrals and a surface integral. If all the integrals were volume integrals, over the same volume of course, we could proclaim that the integral statement (sometimes called a global balance equation) would hold provided the integrands satisfied a so-called local balance equation (recall that to get the local balance equation is our goal). For instance, from the integral statement

$$\int_V \alpha \frac{\partial M}{\partial t} \, \mathrm{d}V = \int_V \mu \, \mathrm{d}V \ , \tag{7.8}$$

where α, M, and μ are some functions, one could conclude that

$$\alpha \frac{\partial M}{\partial t} = \mu \ , \tag{7.9}$$

which is a local version of (7.8). An argument along these lines could for instance invoke the assumption that the volume V was arbitrary, and that it could be shrunk around a given point, which in the limit would allow the volume to be canceled on both sides of the equation.

To execute this program for Eq. (7.7), we have to convert the surface integral to a volume integral. We have the needed tool in the celebrated ***divergence theorem*** (also known as the Gauss theorem)

$$\int_V \mathrm{div}\boldsymbol{q} \, \mathrm{d}V = \int_S \boldsymbol{n} \cdot \boldsymbol{q} \, \mathrm{d}S \ , \tag{7.10}$$

where the divergence of the flux vector is defined in Cartesian coordinates as

$$\mathrm{div}\boldsymbol{q} = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} \ .$$

Consequently, Eq. (7.7) may be rewritten

$$\int_V c_V \frac{\partial T}{\partial t}\, \mathrm{d}V = -\int_V \mathrm{div}\boldsymbol{q}\, \mathrm{d}V + \int_V Q\, \mathrm{d}V\ , \tag{7.11}$$

and grouping the terms as

$$\int_V \left[c_V \frac{\partial T}{\partial t} + \mathrm{div}\boldsymbol{q} - Q \right]\, \mathrm{d}V = 0\ . \tag{7.12}$$

we may conclude that the inside of the bracket has to vanish since the volume could be entirely arbitrary. Therefore, we arrive at the ***local balance equation***

$$c_V \frac{\partial T}{\partial t} + \mathrm{div}\boldsymbol{q} - Q = 0\ . \tag{7.13}$$

7.2 Constitutive equation

Equation (7.13) contains too many variables: both temperature and heat flux. Since it is a scalar equation, the logical step is to express the heat flux in terms of temperature. That is the content of the Fourier model: heat flows opposite to the gradient of the temperature (downhill). In matrix form

$$\boldsymbol{q} = -\boldsymbol{\kappa}(\mathrm{grad}T)^T\ . \tag{7.14}$$

The matrix $\boldsymbol{\kappa}$ is the conductivity matrix of the material. The most common forms of $\boldsymbol{\kappa}$ are

$$\boldsymbol{\kappa} = \kappa \mathbf{1} \tag{7.15}$$

for the so-called thermally isotropic material, and

$$\boldsymbol{\kappa} = \begin{pmatrix} \kappa_x & 0 & 0 \\ 0 & \kappa_y & 0 \\ 0 & 0 & \kappa_z \end{pmatrix}\ , \tag{7.16}$$

for materials that have three orthogonal directions of different thermal conductivities (orthotropic material); κ is the isotropic thermal conductivity coefficient, $\mathbf{1}$ is the identity matrix, and κ_x, κ_y, and κ_z are the orthotropic thermal conductivities. To explain the orthotropic conductivity model we note that some materials have preferred directions in which heat would like to flow, for instance along the fibers in a composite. Visually, we can imagine a corrugated steel roof, with the channels running not directly downhill, but tilted away from the slope – the water would run preferentially in the channels, but generally downhill.

The funny looking transpose of the temperature gradient follows from the definition: the gradient of the scalar is a row matrix

$$\mathrm{grad}T = \left[\frac{\partial T}{\partial x}, \frac{\partial T}{\partial y}, \frac{\partial T}{\partial z} \right]\ . \tag{7.17}$$

With the constitutive equation, the balance equation (7.13) is now expressed purely in terms of the absolute temperature,

$$c_V \frac{\partial T}{\partial t} - \mathrm{div}\left[\boldsymbol{\kappa}(\mathrm{grad}T)^T\right] - Q = 0\ . \tag{7.18}$$

7.3 Boundary conditions

From now on, V is going to be the volume of the whole solid domain. The most important fact about the boundary conditions is that we need to have a boundary condition *at each point* of the surface S (see Figure 7.2). As we may suspect by now, the model is all about temperature. Correspondingly, the boundary conditions are an expression of our *a priori* knowledge of the temperature distribution in the solid or on its surface.

The simplest boundary condition results if we know the surface temperature along one part of S at all times. This part of the surface will be called S_1 (see Fig. 7.2). Therefore,

$$T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \quad \boldsymbol{x} \text{ on } S_1 \ . \tag{7.19}$$

This type of condition is known as the primary, or ***essential***, boundary condition. Sometimes it is also referred to as the boundary condition of the first kind, or the Dirichlet boundary condition.

The heat flux entering or leaving the solid may also be known (measured by a heat flux gauge, for instance). Generally, we do not know the heat flux along the surface, only the normal component, which is available from the normal to the surface and the heat flux as $q_n = \boldsymbol{n} \cdot \boldsymbol{q}$. Therefore, along the S_2 part of the surface the normal component of the heat flux may be prescribed

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = \overline{q}_n, \quad \boldsymbol{x} \text{ on } S_2 \ . \tag{7.20}$$

All quantities are given at a particular point on the boundary as functions of time, similarly to the first boundary condition. This type of condition is known as the ***natural*** (or flux) boundary condition. Sometimes it is also referred to as the boundary condition of the second kind, or the Neumann boundary condition.

As the last example of a boundary condition, we will mention heat transfer driven by a temperature difference across a surface. The normal component of the heat flux is given as

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = h(T - T_a), \quad \boldsymbol{x} \text{ on } S_3 \ , \tag{7.21}$$

where T_a is the known temperature of the surrounding medium (ambient temperature), and h is the surface heat transfer coefficient. This boundary condition is often used in the so-called forced-convection situations where a solid body is exposed to a forced flow of some fluid medium at a given temperature. As a result of the sticking of the fluid to the solid a flow boundary layer develops. In general yet another boundary layer develops, a thermal boundary layer, over which there will be a temperature gradient between the fluid outside of the boundary layer and the surface of the wetted solid body. The inverse of the thermal resistance of the boundary layer is expressed through the surface heat transfer coefficient which depends on the many parameters involved (the properties of the surface of the body, the properties of the fluid, the properties of the convection flow, etc.). Sometimes it is also referred to as the boundary condition of the third kind, or the Newton boundary condition. Since this kind of boundary condition has application also in other situations such as modeling of the thermal resistance of contact between two solid bodies, or modeling of thin insulation layers, as surfaces with thermal gradients across them, we will refer to this condition as the ***surface heat transfer*** boundary condition.

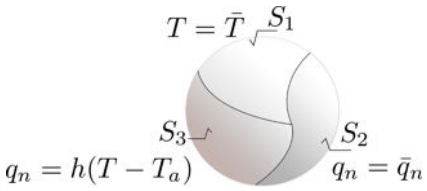


Fig. 7.2. The partitioning of the boundary surface used in the description of the boundary conditions

7.3.1 On the sufficiency of boundary conditions.

As pointed out earlier in this section, one boundary condition is needed at each point on the boundary. The precise mathematical statement of the necessity of having one boundary condition in place is somewhat involved, but we can build on intuition fairly easily.

Would it be possible to specify one boundary condition at only a subset of the complete boundary, leaving the behavior of the solution along a part or parts of the boundary unspecified? As a thought experiment, we consider a square domain, shown in Fig. 7.3, with no source of heat generation, and zero temperature prescribed on the S_1 subset of the boundary. On the $\widehat{S}$ part of the boundary we assume nothing is known about the temperature distribution. Is it possible that the temperature field is completely determined by these boundary conditions?

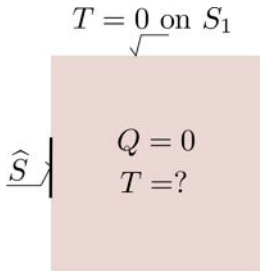


Fig. 7.3. The square domain with partially undefined boundary condition

If it was true, the variation of temperature along $\widehat{S}$ wouldn't affect the solution in the domain. However, if zero temperature was prescribed all around the circumference of the square, the solution to this problem would be zero temperature everywhere. Consequently, also the normal component of the flux (in fact all components of the flux) would vanish everywhere. Evidently, if the temperature along $\widehat{S}$ was nonzero, it would require transitioning to zero temperature on S_1 (and elsewhere within the domain), hence the solution within the domain *would* depend on the temperature along $\widehat{S}$; alternatively, if there was nonzero heat flux along $\widehat{S}$, the temperature distribution within the square domain would be affected. This is illustrated in Fig. 7.4: varying the heat flux along $\widehat{S}$ (here shown for two different uniform distributions, positive and negative) changes the distribution of temperature. Therefore, we must conclude that *prescribing one boundary condition along the entire boundary* of the domain is a *necessary condition* for making the solution unique.

Is not a sufficient condition, however. In the so-called Neumann problem only the heat flux is being prescribed along the entire boundary. This is equivalent to the pure-traction problem of Section 5.3. The solution is not unique, because any temperature distribution of the form $T(x, y) + \widetilde{T}$, where $T(x, y)$ satisfies the balance equation and the natural boundary conditions, and $\widetilde{T}$ is a constant, is also a solution (the constant term disappears with differentiation). Typically, the Neumann boundary conditions are supplemented with temperature being prescribed at one point to remove the constant $\widetilde{T}$ from consideration.

7.4 Example of Boundary Condition formulation

Consider Figure 7.5. It shows a cross-section of an insulated wall, with the detail of the transition of the wall into the roof. Away from this transition the flow of heat from the interior (warm) to the exterior (cold) may be considered to be strictly through the thickness of the wall. Consequently, on the interior and exterior flat faces of the wall we may consider for instance Newton's boundary conditions.

Since the heat flux is orthogonal to the flat faces of wall, and also to the interfaces between the various material layers, we may undertake to carve out a "pipe"-like conduit by drawing an

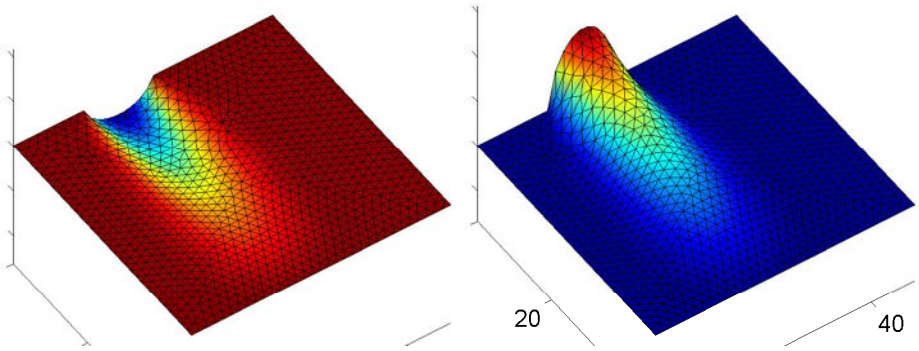


Fig. 7.4. The square with two different distributions of heat flux along $\widehat{S}$

imaginary surface that separates a cylindrical volume of the wall from the rest. The heat flux would enter the pipe at the warm surface S_{c2}, pass through it without leaving or entering through the cylindrical surface S_i, and exit on the cold face of the wall S_{c1} (see Figure 7.6). Therefore, our model could be limited to a finite volume of the wall, the cylinder bounded by S_{c2}, S_i, and S_{c1}, and the boundary conditions would be specified on the two flat base faces

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = h_1(T - T_{a1}), \quad \boldsymbol{x} \text{ on } S_{c1} , \quad q_n = \boldsymbol{n} \cdot \boldsymbol{q} = h_2(T - T_{a2}), \quad \boldsymbol{x} \text{ on } S_{c2} , \tag{7.22}$$

and on the cylindrical surface, where we state that no heat enters or leaves through this surface

$$q_n = \boldsymbol{n} \cdot \boldsymbol{q} = 0, \quad \boldsymbol{x} \text{ on } S_i . \tag{7.23}$$

Such a three-dimensional heat flux picture is conducive to considering a model with just a single coordinate in which the heat flux is described: along the through the thickness direction.

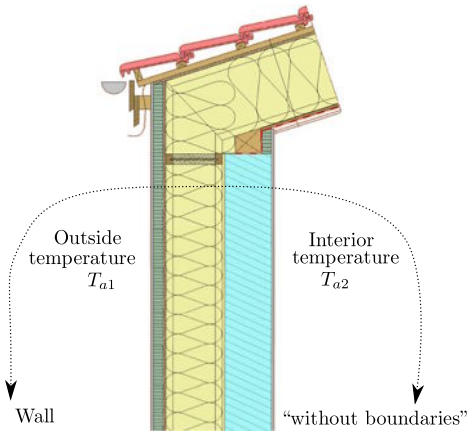


Fig. 7.5. Insulated wall: Outside of the influence of the irregularities caused by the construction of the eaves the wall may be considered to be "without boundaries".

7.5 Initial condition

The primary variable in our problem is the temperature, T, and it is present in the balance equation (7.18) with the first order time derivative. Therefore, we will need one initial condition,

$$T(\boldsymbol{x}, 0) = \overline{T}_0(\boldsymbol{x}) \quad \boldsymbol{x} \text{ in } V . \tag{7.24}$$

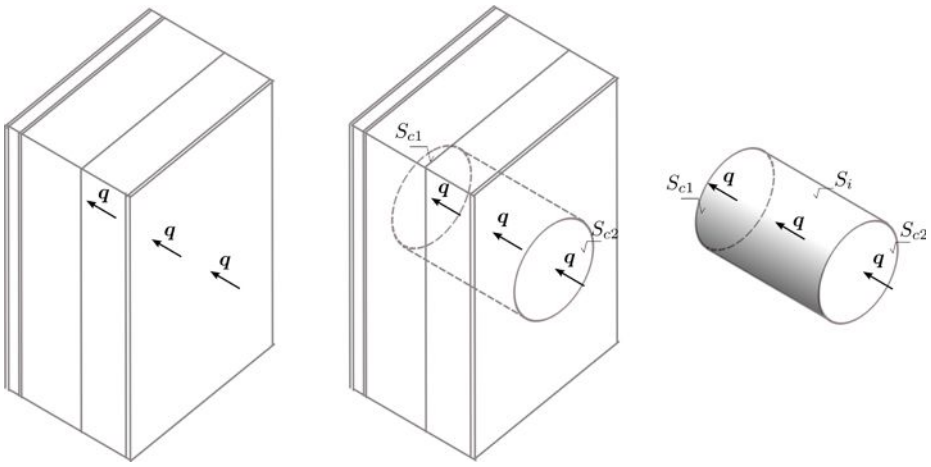


Fig. 7.6. Insulated wall: Heat flux passing through the wall from the interior to the exterior. Left: Wall of infinite extent. Middle: wall with an imaginary volume in the form of a rectangular cylinder. Right: computational volume and its boundaries.

The initial condition must match any boundary condition on S_1 at time $t = 0$:

$$\overline{T}_0(\boldsymbol{x}) = \overline{T}(\boldsymbol{x}, 0), \quad \boldsymbol{x} \text{ on } S_1 \ . \tag{7.25}$$

7.6 Summary of the PDE model of heat conduction

Figure 7.7 gives a pictorial overview of the terminology and the various equations of the model of heat conduction (for the curious: it is a simplified Tonti diagram). One of the main points of a picture is to save a thousand words, so I let it do its magic.

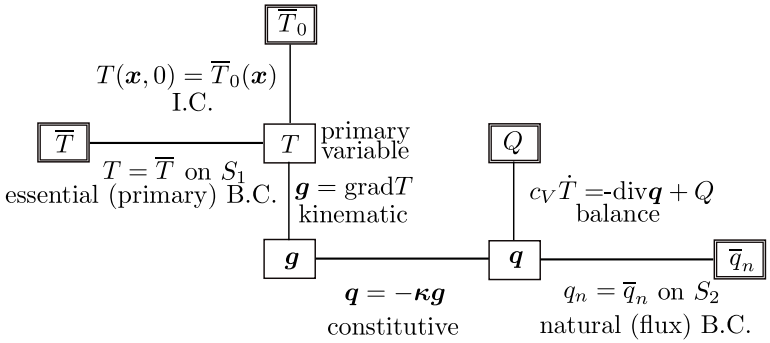


Fig. 7.7. Diagram of the heat conduction model (Tonti diagram). BC: boundary condition; IC: initial condition. Quantities given as data are shown in doubled boxes. The primary variable is indicated. All equations are represented by the line connections of boxes and labeled.

7.7 Parallels between the taut wire and the heat conduction model

It is really useful to be able to relate the material one has digested in the past and the material that is new, to discover their commonalities and differences. In this Chapter we have studied the heat conduction model (IBVP). At some points we must have realized that we have seen something like

that before. Perhaps the symbols are different, but the meaning may be the same? That is in fact true. The model of the taut wire and the heat conduction model share a number of features. For one, the PDE looks familiar. There are boundary conditions and initial conditions, and so on. In this section we will establish a table that matches up the features of one model with the features in the other. In what follows the heat conduction model is on the left, the taut wire model is on the right.

Primary variable. The primary variable is (we also indicate the independent variables, space and time) is a scalar function that varies in space and time

$T(\boldsymbol{x},t)$	$w(x,t)$

Derived variable and the kinematic equation. The derived variable is obtained from the primary variable by differentiation. In the heat conduction model this is the gradient of temperature, and the wire model this is the slope of the deflection curve. The equation that links the primary and arrived variable is called kinematic equation. The derived quantity in the heat conduction model is a row matrix whereas it is a scalar quantity for the wire model. This difference is due to the number of space dimensions: three for the heat conduction, one for the wire model.

$\boldsymbol{g} = (\text{grad}T)^T$	$\theta = w'$

Flux variable and constitutive equation. The flux variable is obtained from the derived variable using the constitutive equation. In the heat conduction model this is heat flux linked to the gradient of temperature by the Fourier law, and the wire model the flux variable is the transverse force S, linked to the slope of the deflection curve θ through the prestress force (so yes, the pre-stress force P is a constitutive property for the wire):

$\boldsymbol{q} = -\boldsymbol{\kappa g}$	$S = P\theta$

Balance equation. The evolution of the primary variable is governed by the balance equation. This is the PDE that one needs to solve. Note that it is written in terms of the flux variable, but also the primary variable, and finally the given so-called source term which is the heat generation rate in the heat conduction problem and the transverse distributed load in the taut wire problem. Note that the heat conduction equation is first order in time, whereas the wire equation is second order in time. This means that the behavior of the solutions is going to be of different character with respect to time: exponential decay for the heat conduction, oscillation for the wire.

$c_V\dot{T} = -\text{div}\boldsymbol{q} + Q$	$\mu\ddot{w} = S' + q$

Essential boundary condition. Sure, the PDE is important, but the model is not called IBVP for nothing: the boundary and initial conditions dominate the solution. The primary variable may have a boundary condition associated with it, the essential boundary condition. We write the corresponding part of the boundary S_1 for both models with the understanding that for the wire model S_1 would consist of one or two endpoints of the wire at which displacement is prescribed (pin support):

$T(\boldsymbol{x},t) = \overline{T}(\boldsymbol{x},t), \quad \boldsymbol{x} \text{ on } S_1$	$w(x,t) = \overline{w}(x,t), \quad x \text{ on } S_1$

Flux boundary condition. The flux variable may have a boundary condition associated with it, the flux boundary condition. We write the corresponding part of the boundary S_2 for both models with the understanding that for the wire model S_2 would consist of one or two endpoints of the wire at which concentrated force is prescribed (roller support). Furthermore we have to take into account the direction of the transverse force S: we can see from Figures 1.4 and 1.5 that the cross sections in which the transverse forces are accounted for have opposite normals. At the right hand side end of the wire we take the cross-section whose normal points to the left and vice versa at the left-hand side end of the wire. Therefore we get the two boundary conditions

$$S(L,t) = Pw'(L,t) = F_L(t) \ , \quad S(0,t) = Pw'(0,t) = -F_0(t) \ .$$

This dependence on the normal to the boundary can be accommodated by writing the boundary condition for the wire as

$$S(x,t) \cdot n(x) = \bar{F}(x,t),$$

where the point x is on the boundary of the wire S_2 (either the left-hand side or the right-hand side end), and the normal to the boundary is either $n(L) = +1$ or $n(0) = -1$. This will make the two models correspond to each other nicely in the formulation of the boundary condition:

$\boldsymbol{q}(\boldsymbol{x},t) \cdot \boldsymbol{n} = \overline{q}_n(\boldsymbol{x},t), \quad \boldsymbol{x} \text{ on } S_2$	$S(x,t) \cdot n(x) = \bar{F}(x,t), \quad x \text{ on } S_2 \ .$

Newton's boundary condition. The third kind of boundary condition is the Newton's surface heat transfer (which corresponds to the spring support for the wire model). We write the corresponding part of the boundary S_3 for both models with the understanding that for the wire model S_3 would consist of one or two endpoints of the wire at which the spring support is applied. Furthermore we have to take into account the direction of the transverse force S: we can see from Figures 1.4 and 1.5 that the cross sections in which the transverse forces are accounted for have opposite normals. The dependence on the normal to the boundary can be accommodated by writing the boundary condition for the wire as

$$S(x,t) \cdot n(x) = k(x)(w(x) - w_{a,x})$$

where the point x is on the boundary of the wire S_3 (either the left-hand side or the right-hand side end), and the normal to the boundary is either $n(L) = +1$ or $n(0) = -1$. Hence the correspondence

$\boldsymbol{q}(\boldsymbol{x},t) \cdot \boldsymbol{n} = h(\boldsymbol{x})(T(\boldsymbol{x},t) - T_{a,\boldsymbol{x}}(t))$ $\boldsymbol{x} \text{ on } S_3$	$S(x,t) \cdot n(x) = k(x)(w(x,t) - w_{a,x}(t))$ $x \text{ on } S_3 \ .$

Initial condition. The primary variable has initial condition attached to it, one or two of them depending on the order of the time derivative in the balance equation. We write the interior of the domain V for both models with the understanding that for the wire model V would consist of the entire length of the wire:

$T(\boldsymbol{x},0) = \overline{T}_0(\boldsymbol{x}), \quad \boldsymbol{x} \text{ in } V$	$w(x,0) = \overline{W}(x), \quad \dot{w}(x,0) = \overline{V}(x), \quad x \text{ in } V$

To summarize the wire model we can take advantage of the Tonti diagram again – Figure 7.8. This may be compared with the equivalent diagram for the heat conduction model in Figure 7.7. Clearly there is much that is shared in terms of structure between the two models. That is good news since if we understand a finite element procedure for one model we already have a good understanding of the other.

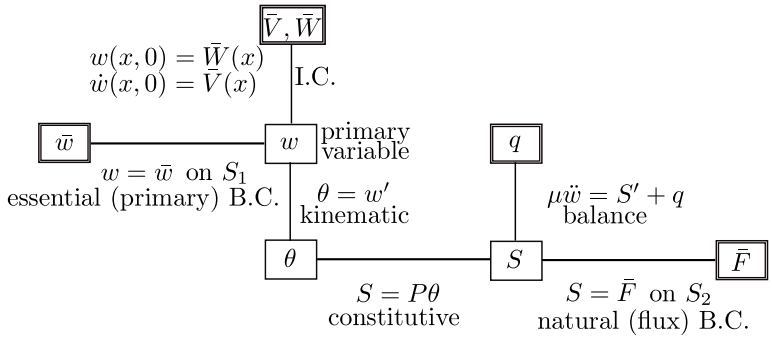


Fig. 7.8. Diagram of the taut wire model (Tonti diagram). BC: boundary condition; IC: initial condition. Quantities given as data are shown in doubled boxes. The primary variable is indicated. All equations are represented by the line connections of boxes and labeled.

8

Galerkin Method for the Model of Heat Conduction

In this section we will take up the Galerkin finite element discretization for the heat conduction IBVP. We will formulate the model in two space dimensions which will allow us present the finite element technique for meshes composed of triangles.

8.1 Weighted residual formulation

We will follow the same path as in Chapter 2. The major steps are: satisfy the essential boundary conditions by designing the trial function, shift the derivatives in the balance equation residual, and combine the balanced equation residual with the natural boundary condition in one residual equation.

The essential boundary condition is satisfied by restricting possible trial functions to only those that conform to the essential boundary conditions a priori

$$T(\boldsymbol{x},t) - \overline{T}(\boldsymbol{x},t) = 0, \quad \boldsymbol{x} \text{ on } S_1 \ , \tag{8.1}$$

The next step equalizes the number of derivatives on the test and trial functions. The balance equation (7.18) yields the balance residual as

$$r_B = c_V \frac{\partial T}{\partial t} - \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] - Q \ . \tag{8.2}$$

so that the weighted balance residual reads

$$\int_V \eta(\boldsymbol{x}) r_B(\boldsymbol{x},t) \ \text{d}V \ . \tag{8.3}$$

The first term ($\eta c_V \frac{\partial T}{\partial t}$) and the third term ($\eta Q$) are kept without change, but the second term

$$-\eta \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right]$$

reminds us of a similar term in Eq. (2.6): the test function η multiplies an expression that contains the second derivatives of temperature (the div $\left[\boldsymbol{\kappa}(\text{grad}T)^T\right]$ term). Balancing the order of differentiation by shifting one derivative from the temperature to the test function η will be beneficial: similarly to Section 2.5, we will be able to use basis functions that are less smooth since we would not require the second derivatives, and also we will be able to satisfy the natural boundary conditions without having to include them as a separate residual (naturally!). As before, the price to pay is the need to place some restrictions on the test function.

Integration by parts was used in Section 2.5, and just a little bit more general tool will work here too. For the moment, it will be convenient to work with the expression

$$-\eta \ \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] = \eta \ \text{div}\boldsymbol{q} \ ,$$

that is, we work with the flux variable instead of $-\boldsymbol{\kappa}(\text{grad}T)^T$.

The integration by parts in the case of a multidimensional integral is generalized in the divergence theorem (7.10). We may anticipate that $\eta\ \text{div}\boldsymbol{q}$ is the result of the chain rule applied to the vector $\eta\ \boldsymbol{q}$. That is indeed the case, as we have

$$\text{div}\,(\eta\ \boldsymbol{q}) = \eta\ \text{div}\boldsymbol{q} + (\text{grad}\eta)\ \boldsymbol{q}\ , \tag{8.4}$$

which is easily verified in components. Therefore, we may start by inspecting the integral

$$\int_V \eta\ \text{div}\boldsymbol{q}\ \text{d}V$$

where we substitute from (8.4)

$$\int_V \eta\ \text{div}\boldsymbol{q}\ \text{d}V = \int_V \text{div}\,(\eta\ \boldsymbol{q})\ \ \text{d}V - \int_V (\text{grad}\eta)\ \boldsymbol{q}\ \text{d}V\ . \tag{8.5}$$

The divergence theorem may be applied to the first integral on the right to give the identity

$$\int_V \eta\ \text{div}\boldsymbol{q}\ \text{d}V = \int_S \eta\ \boldsymbol{q}\cdot\boldsymbol{n}\ \text{d}S - \int_V (\text{grad}\eta)\ \boldsymbol{q}\ \text{d}V\ . \tag{8.6}$$

Since $\boldsymbol{q}\cdot\boldsymbol{n}$ is known on some parts of the boundary, but unknown on the others – see Eqs. (7.20) and (7.21), we will split the surface integral into one for each sub-surface,

$$\int_V \eta\ \text{div}\boldsymbol{q}\ \text{d}V = \\ \int_{S_1} \eta\ \boldsymbol{q}\cdot\boldsymbol{n}\ \text{d}S + \int_{S_2} \eta\ \boldsymbol{q}\cdot\boldsymbol{n}\ \text{d}S + \int_{S_3} \eta\ \boldsymbol{q}\cdot\boldsymbol{n}\ \text{d}S - \int_V (\text{grad}\eta)\ \boldsymbol{q}\ \text{d}V\ . \tag{8.7}$$

We see that the situation is analogous to the one discussed below Eq. (2.13): The integral over the part of the surface S_1 is troublesome, because $\boldsymbol{q}\cdot\boldsymbol{n}$ is unknown there. However, we have the option of making η vanish along S_1, making

$$\int_{S_1} \eta\ \boldsymbol{q}\cdot\boldsymbol{n}\ \text{d}S = 0\,,$$

where the test function now must satisfy $\eta(\boldsymbol{x}) = 0$ for $\boldsymbol{x}\in S_1$. Now that we eliminated the integral over S_1 we switch back from $\boldsymbol{q}$ to $-\boldsymbol{\kappa}(\text{grad}T)^T$ to obtain from (8.7)

$$\int_V \eta\ \text{div}\boldsymbol{q}\ \text{d}V = -\int_V \eta\ \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right]\ \text{d}V = \\ \int_{S_2} \eta\ (\boldsymbol{q}\cdot\boldsymbol{n})\ \text{d}S + \int_{S_3} \eta\ (\boldsymbol{q}\cdot\boldsymbol{n})\ \text{d}S + \int_V (\text{grad}\eta)\ \boldsymbol{\kappa}(\text{grad}T)^T\ \text{d}V\ . \tag{8.8}$$

The heat flux passing through the surface S_2 is known: see the boundary condition (7.20). Therefore, if we attempt to satisfy this boundary condition in a weighted residual sense we write

$$\int_{S_2} \underline{\eta}\ \left[\overline{q}_n - \underline{(\boldsymbol{q}\cdot\boldsymbol{n})}\right]\ \text{d}S = 0 \tag{8.9}$$

where we anticipate a possible cancellation if we take η as the test function. Note that the underlined term is present with the opposite sign in (8.7). Similarly, on the boundary surface S_3 we might attempt to satisfy the boundary condition with a weighted residual equation

$$\int_{S_3} \underline{\eta}\ \left[h(T-T_a) - \underline{(\boldsymbol{q}\cdot\boldsymbol{n})}\right]\ \text{d}S = 0 \tag{8.10}$$

Finally we have all the pieces ready. We take as the *weighted residual statement for the heat conduction* problem the weighted balance residual (8.3) to which we add the natural boundary condition residuals (8.9) and (8.10).

$$\int_V \eta \left(c_V \frac{\partial T}{\partial t} - \text{div}\left[\boldsymbol{\kappa}(\text{grad}T)^T\right] - Q \right) \, \text{d}V + \int_{S_2} \underline{\eta} \left[\overline{q}_n - \underline{(\boldsymbol{q}\cdot\boldsymbol{n})} \right] \, \text{d}S + \int_{S_3} \underline{\eta} \left[h(T - T_a) - \underline{(\boldsymbol{q}\cdot\boldsymbol{n})} \right] \, \text{d}S = 0 \, . \tag{8.11}$$

Expression (8.8) is introduced to replace the second volume term. The underlined surface terms cancel with the surface integrals in (8.8) and we arrive at

$$\begin{aligned} &\int_V \eta c_V \frac{\partial T}{\partial t} \, \text{d}V + \int_V (\text{grad}\eta) \, \boldsymbol{\kappa} (\text{grad}T)^T \, \text{d}V - \int_V \eta Q \, \text{d}V \\ &+ \int_{S_2} \eta \, \overline{q}_n \, \text{d}S + \int_{S_3} \eta \, h(T - T_a) \, \text{d}S = 0, \quad \eta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in S_1 \, . \end{aligned} \tag{8.12}$$

In this equation we have our result: a single weighted residual statement with balanced derivatives.

8.2 One-dimensional heat conduction model

In this section we will reduce the Galerkin model for heat conduction so that there will be one space coordinate and the time. We will consider the flow of heat energy through a wall whose boundary (where it meets the floor, other walls, or the ceiling) is going to be ignored. See Figure 7.6. The heat flux through the boundary S_i is zero: the heat flows through the volume as if through a pipe

$$q_n = \boldsymbol{q} \cdot \boldsymbol{n} = 0 \text{ on } S_i$$

The through the thickness coordinate will be x and the Galerkin formulation will be independent of the other two coordinates y, z (in the plane of the wall) if the following conditions are satisfied.

1. The boundary conditions on S_{c1} and S_{c2} must be independent of y, z.
2. The initial distribution of temperature must be independent of y, z.
3. The thermal material properties of the wall change only through the thickness, and are independent of y, z.
4. The components of the gradient of the temperature in the plane of the wall must be zero, $\partial T/\partial y = 0$ and $\partial T/\partial z = 0$.
5. Since the heat flux components q_y and q_z must also be zero, the material must satisfy the conditions

   $$q_y = -\kappa_{yx}\frac{\partial T}{\partial x} - \kappa_{yy}\frac{\partial T}{\partial y} - \kappa_{yz}\frac{\partial T}{\partial z} = 0 \Rightarrow \kappa_{yx} = 0$$

   and

   $$q_z = -\kappa_{zx}\frac{\partial T}{\partial x} - \kappa_{zy}\frac{\partial T}{\partial y} - \kappa_{zz}\frac{\partial T}{\partial z} = 0 \Rightarrow \kappa_{zx} = 0$$

   so that the gradient component $\partial T/\partial x \neq 0$ doesn't drive heat flux in the plane of the wall. All materials which are reasonably isotropic (i.e. whose properties are the same in all directions) satisfy this condition. Many orthotropic materials consisting of layers parallel to the wall also do.

Figure 8.1 shows the three-dimensional volume for which we now write the Galerkin weighted residual equation. At the two cross-sections, S_{c1} at $x = L$ and S_{c2} at $x = 0$, we will consider all three possible boundary conditions, prescribed temperature, heat flux, and surface heat transfer. Of course we realize that there are only two surfaces on which to apply one boundary condition on each. What we mean by considering all three possibilities is that we pick one out of three for each cross-section.

The starting point is equation (8.12). One by one we will introduce our simplifying assumptions formulated above. For instance, let us take the volume integral

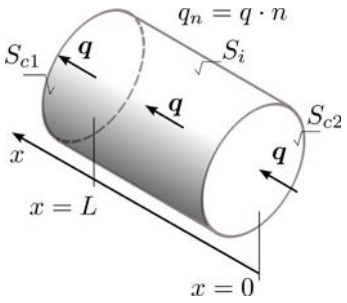


Fig. 8.1. Model of heat conduction through the thickness of the wall

$$\int_V \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}V$$

As dictated by our assumptions the only quantity that could possibly depend on y, z is the test function, and we are at liberty to require that it be only function of x, i.e. $\eta = \eta(x)$. Then we can partially evaluate the integral by integrating first in the cross section of the wall S

$$\int_V \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}V = \int_0^L \int_S \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}S \mathrm{d}x = \int_S \mathrm{d}S \int_0^L \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}x = S \int_0^L \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}x$$

Similarly, since only the gradient components $\partial T/\partial x$ and $\partial \eta/\partial x$ are nonzero we obtain

$$\int_V (\mathrm{grad}\eta) \, \boldsymbol{\kappa} (\mathrm{grad}T)^T \, \mathrm{d}V = S \int_0^L \frac{\partial \eta}{\partial x} \kappa_{xx} \frac{\partial T}{\partial x} \, \mathrm{d}x$$

The same reasoning is applied also to surface integrals. For instance

$$\int_{S_2} \eta \, \overline{q}_n \, \mathrm{d}S = S \eta|_{S_2} \overline{q}_n|_{S_2}$$

where $\eta|_{S_2}$ means the value of the test function at the cross-section where heat flux (boundary condition 2) is applied, and similarly for $\overline{q}_n|_{S_2}$. For instance, let us say that heat flux is prescribed on S_{c2} at $x = 0$. Then

$$S \eta|_{S_2} \overline{q}_n|_{S_2} = S \eta(x = 0) \overline{q}_n(x = 0)$$

In the end we obtain the Galerkin formulation in a single independent space coordinate as

$$\begin{aligned} & S \int_0^L \eta c_V \frac{\partial T}{\partial t} \, \mathrm{d}x + S \int_0^L \frac{\partial \eta}{\partial x} \kappa_{xx} \frac{\partial T}{\partial x} \, \mathrm{d}x - S \int_0^L \eta Q \, \mathrm{d}x \\ & + S \eta|_{S_2} \overline{q}_n|_{S_2} + S \eta|_{S_3} h|_{S_3} (T|_{S_3} - T_a|_{S_3}) = 0, \quad \eta|_{S_1} = 0 \, . \end{aligned} \tag{8.13}$$

or, using primes and dots to denote the partial derivatives

$$\begin{aligned} & S \int_0^L \eta c_V \dot{T} \, \mathrm{d}x + S \int_0^L \eta' \kappa_{xx} T' \, \mathrm{d}x - S \int_0^L \eta Q \, \mathrm{d}x + S \eta|_{S_2} \overline{q}_n|_{S_2} \\ & + S \eta|_{S_3} h|_{S_3} (T|_{S_3} - T_a|_{S_3}) = 0, \quad \eta|_{S_1} = 0 \, . \end{aligned} \tag{8.14}$$

Note that we still keep the cross sectional area S in this expression, even though we could have canceled it. The reason is that it allows us to keep in mind that the equation still models the flow of heat energy through a three-dimensional body. Keeping track of the units is also easier with the cross-sectional area in place since all the terms are in the physical units of power (watt).

8.3 Comparison with the prestressed wire

Equation (3.8) for the prestressed wire dynamics can be brought into correspondence with (8.14). For that purpose we will introduce the notation of Figure 8.2: the boundary of the wire (the endpoints) will be referred to either as S_1 when the cross-section is supported by a pin (essential boundary condition), or as S_2 when the cross-section is on a roller (natural boundary condition). For instance for the configuration of Figure 1.1 we have

$$S_1: \quad x = 0\,, \quad S_2: \quad x = L$$

and therefore the term $\eta(L)F_L$ would be written as $\eta|_{S_2}F|_{S_2}$. Changing the sign on both sides of (3.8) and introducing the notation for the boundary terms yields

$$\int_0^L \eta\mu\ddot{w}\,\mathrm{d}x + \int_0^L \eta' P w'\,\mathrm{d}x - \int_0^L \eta q\,\mathrm{d}x - \eta|_{S_2}F|_{S_2} = 0\,, \qquad \text{where we require } \eta|_{S_1} = 0. \tag{8.15}$$

This then aligns the wire dynamics and the heat conduction model so that we can try to match individual terms. For instance, we can identify

$$S\int_0^L \eta'\kappa_{xx}T'\,\mathrm{d}x = \int_0^L \eta'(S\kappa_{xx})T'\,\mathrm{d}x$$

with

$$\int_0^L \eta' P w'\,\mathrm{d}x\,.$$

We use different symbols, but these two expressions are of the same "type". This is important as it will allow us to save a lot of work when applying the finite element discretization.

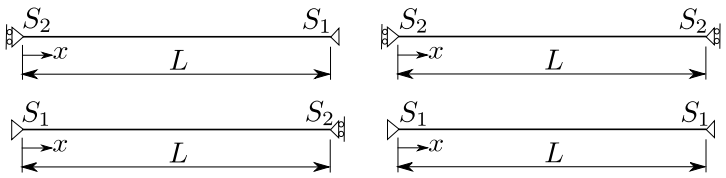


Fig. 8.2. Boundary condition nomenclature for the prestressed wire

8.4 Heat conduction 1D FEM

As for the wire model in the preceding chapters, we will apply the L2 finite element to the residual equations. We have previously derived the elementwise load vector (2.29). Since we can identify

$$S\int_0^L \eta Q\,\mathrm{d}x = \int_0^L \eta(SQ)\,\mathrm{d}x$$

from the heat conduction weighted residual with

$$\int_0^L \eta q\,\mathrm{d}x$$

from the weighted residual for the prestressed wire, for uniform heat generation density Q we can immediately write the ***elementwise vector of heat loads*** due to internal heat generation as

$$[L]^{(e)} = \begin{bmatrix} (SQ)(x_M - x_K)/2 \\ (SQ)(x_M - x_K)/2 \end{bmatrix} = \frac{(SQ)(x_M - x_K)}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{8.16}$$

for a finite element connecting nodes K and M. Analogously, the elementwise stiffness matrix for the cable finite element was derived as (2.30). Therefore, we can write for uniform κ_{xx} (i.e. for a homogeneous material) the so-called ***elementwise conductivity matrix***

$$[K]^{(e)} = \frac{S\kappa_{xx}}{x_M - x_K} \begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} \tag{8.17}$$

The heat flux load term (note the negative sign, which we need to insert in order to move this to the right-hand side)

$$-S\eta|_{S_2}\overline{q}_n|_{S_2}$$

corresponds to the terms involving the concentrated forces F_0, F_L for the wire model. The concentrated forces are added to the load vector. Here is how we will treat this term in the heat conduction model: Assume that node k is on the S_2 boundary where heat flux $\overline{q}_n|_{S_2}$ is prescribed. We set the test function to be the finite element basis function $N_{\langle i \rangle}$ defined on the entire mesh, including the node k. Because of the properties of finite element basis functions, only N_k is nonzero at this node, $N_k(x_k) = 1$. Therefore we have

$$-SN_{\langle i \rangle}(x_k)\overline{q}_n(x_k) = -SN_k(x_k)\overline{q}_n(x_k) = -S\overline{q}_n(x_k) \quad \text{for degree of freedom } i \tag{8.18}$$

In other words, $-S\overline{q}_n(x_k)$ will be assembled to the heat load vector component i. We can think of this as the ***elementwise heat flux load*** (this will be discussed in more detail later in this chapter).

Exercise 27. Solve for the distribution of temperature through a wall, with given heat flux on the left and prescribed temperature on the right (Figure 8.3). Use a single-L2 element mesh. Note that the heat flux being negative enters the wall (it is directed from the outside to the inside).

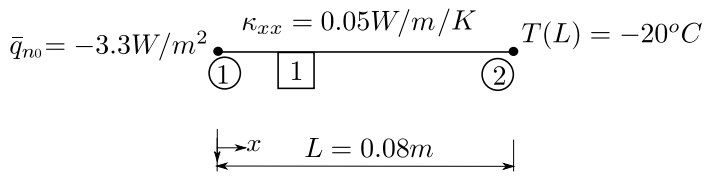


Fig. 8.3. Finite element mesh and data for heat conduction through a wall

Solution: We will assign degree of freedom numbers identical to those of the nodes. Only degree of freedom 1 is unknown; degree of freedom 2 is prescribed as $T_2 = 20°\text{C}$. The conductivity matrix is assembled readily from the elementwise (8.17)

$$[K] = \frac{S\kappa_{xx}}{L} \begin{bmatrix} 1 \end{bmatrix}$$

The heat load vector has two contributions. The first is due to the prescribed temperature, and is obtained by multiplying the second column of the elementwise conductivity matrix by $T_2 = T(L)$ and moving it onto the right-hand side (which explains the negative sign)

$$-\frac{S\kappa_{xx}}{L} \begin{bmatrix} -1 \end{bmatrix} T_2$$

and the second is due to the prescribed heat flux and it is computed as explained above as $-S\bar{q}_{n0}$. Together the heat load vector is assembled as

$$[L] = -\frac{S\kappa_{xx}}{L}\left[-1\right]T_2 - S\bar{q}_{n0}\left[1\right]$$

The solution of

$$[K][T] = [L]$$

for the unknown $[T] = [T_1]$ is

$$[T_1] = [K]^{-1}[L] = \left(\frac{S\kappa_{xx}}{L}\left[1\right]\right)^{-1}\left(-\frac{S\kappa_{xx}}{L}\left[-1\right]T_2 - S\bar{q}_{n0}\left[1\right]\right)$$
$$= T_2 - \frac{\bar{q}_{n0}L}{\kappa_{xx}} = 20°\mathrm{C} - \frac{-3.3\mathrm{W/m}^2 \times 0.08\mathrm{m}}{0.05\mathrm{W/m/°K}} = 25.28°\mathrm{C}$$

Note the temperature decreases left to right, and hence the heat will flow also left to right. This is consistent with the prescribed heat flux which enters the wall, passes through it, and exits on the right. In this case the mathematical model is solved exactly with the one-finite element model. This is because the distribution of temperature is linear through the thickness of the wall, and that can be described without error by the L2 finite element.

The Newton's boundary condition term as found on the left-hand side of the weighted residual equation

$$S\eta|_{S_3}h|_{S_3}(T|_{S_3} - T_a|_{S_3})$$

is treated similarly to the prescribed heat flux: Assume that node k is on the S_2 boundary where surface heat transfer $S\eta|_{S_3}h|_{S_3}(T|_{S_3} - T_a|_{S_3})$ is prescribed. We set the test function to be the finite element basis function $N_{\langle i\rangle}$ defined on the entire mesh, including the node k. Because of the properties of finite element basis functions, only N_k is nonzero at this node, $N_k(x_k) = 1$. Therefore we have

$$SN_{\langle i\rangle}(x_k)h(x_k)(T(x_k) - T_a(x_k)) = Sh(x_k)(T(x_k) - T_a(x_k))$$
$$= Sh(x_k)T_i - Sh(x_k)T_a(x_k) \quad \text{for degree of freedom } i$$

The first expression contributes $Sh(x_k)$ to the system matrix on the left-hand side, component i, i; the second expression $Sh(x_k)T_a(x_k)$ will be assembled to the heat load vector component i (note the positive sign, as this term is moved on to the right-hand side). We can think of this as the ***elementwise surface heat transfer load*** (this will be discussed in more detail later in this chapter).

Exercise 28. Solve for temperature distribution in wall using a mesh of two finite elements. The boundary conditions on both faces of the wall are of the Newton type, and the ambient temperatures and the surface heat transfer coefficients are different on either side (Figure 8.4).

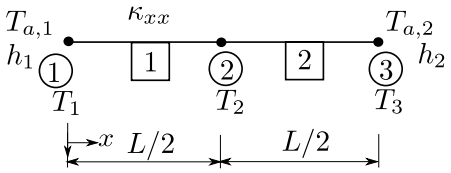


Fig. 8.4. Heat transfer through a wall with Newton's boundary condition at either face

Solution: The conductivity matrix due just to heat conduction is readily assembled as

$$[K] = \frac{2S\kappa_{xx}}{L}\begin{bmatrix} 1\ , -1\ , & 0 \\ -1\ , & 2\ , -1 \\ 0\ , -1\ , & 1 \end{bmatrix}$$

and to this we add Sh_1 to the component 11 (due to the Newton's condition at the left-hand side), and Sh_2 to the component 33 (due to the Newton's condition at the right-hand side) to obtain the system matrix

$$[K] = S\begin{bmatrix} k+h_1\ , -k\ , & 0 \\ -k\ , & 2k\ , & -k \\ 0\ , -k\ , & k+h_2 \end{bmatrix}$$

where we simplify with $k = \frac{2\kappa_{xx}}{L}$.

The right-hand side heat load vector is composed of contributions from the Newton's boundary conditions

$$[F] = S\begin{bmatrix} h_1 T_{a,1} \\ 0 \\ h_2 T_{a,2} \end{bmatrix}$$

A symbolic algebra solution in Matlab reads:

```
>> syms k h1 h2 L Ta1 Ta2  real
K= [k+h1,-k,0;-k,2*k,-k;0,-k,k+h2]
F= [h1*Ta1;0;h2*Ta2]
T=simple(K\F)
K =
[ h1 + k,  -k,       0]
[     -k, 2*k,     -k]
[      0,  -k, h2 + k]
F =
 Ta1*h1
      0
 Ta2*h2
T =
            (2*Ta1*h1*h2 + Ta1*h1*k + Ta2*h2*k)/(2*h1*h2 + h1*k + h2*k)
 (Ta1*h1*h2 + Ta2*h1*h2 + Ta1*h1*k + Ta2*h2*k)/(2*h1*h2 + h1*k + h2*k)
            (2*Ta2*h1*h2 + Ta1*h1*k + Ta2*h2*k)/(2*h1*h2 + h1*k + h2*k)
```

which may be simplified to

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \frac{1}{2 + (k/h_2) + (k/h_1)}\begin{bmatrix} 2T_{a,1} + (k/h_2)T_{a,1} + (k/h_1)T_{a,2} \\ T_{a,1} + T_{a,2} + (k/h_1)T_{a,1} + (k/h_2)T_{a,2} \\ 2T_{a,2} + (k/h_2)T_{a,1} + (k/h_1)T_{a,2} \end{bmatrix}$$

Is instructive to consider the case of infinitely easy transfer of heat between the ambient medium and the wall: $h_1 \to \infty$, $h_2 \to \infty$. This effectively enforces that the surface temperature of the wall is the same as that of the ambient medium, and then we get

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \begin{bmatrix} T_{a,1} \\ (T_{a,1} + T_{a,2})/2 \\ T_{a,2} \end{bmatrix}$$

as expected. The situation is entirely analogous to that of a prestressed wire supported on infinitely stiff springs.

For realistic values of the parameters, for instance for a concrete wall with warm air on the left and cold air on the right, $T_{a,1} = 20°\text{C}$, $T_{a,2} = -30°\text{C}$, $\kappa_{xx} = 0.5\text{W/m/°K}$, $h_1 = 3\text{W/m}^2/°\text{K}$, $h_2 = 15\text{W/m}^2/°\text{K}$, $L = 0.5\text{m}$, we obtain

```
>> Ta1=20; Ta2=-30; k=2*0.5/0.5; h1=3; h2=15;
eval(T)
ans =
    8.0952
   -9.7619
  -27.6190
```

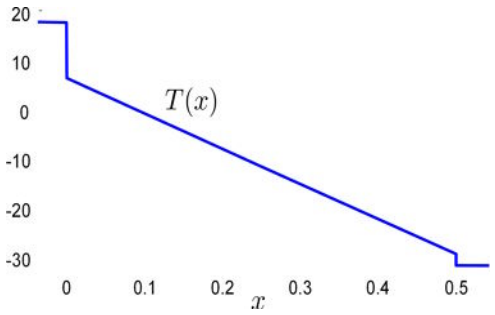


Fig. 8.5. Temperature through the wall with Newton's boundary condition at either face

---

An important difference of the one-coordinate wire model and the heat conduction model is that in heat conduction problems one commonly encounters layered structures, for instance walls composed of several different materials. The thermal conductivity would be different in each layer, and the finite element mesh constructed along the thickness would need to reflect this by assigning different material properties to the elements. Here is an example:

Exercise 29. Solve for the distribution of temperature in the layered wall in Figure 8.6. Discretize the computational domain with one L2 finite element per layer.

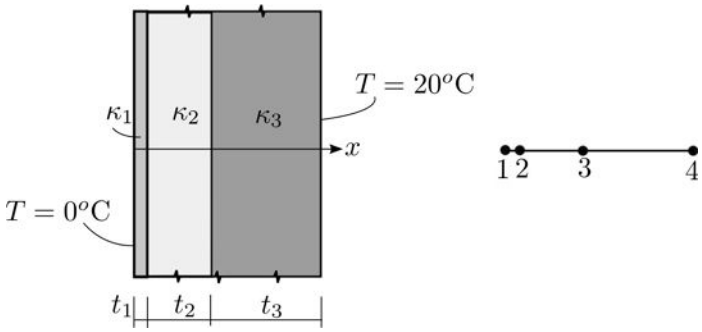


Fig. 8.6. Layered wall exposed to a temperature difference

Solution: Cladding $t_1 = 0.02\text{m}$, $\kappa_1 = 0.7\text{W/m/°K}$, Perlite $t_2 = 0.12\text{m}$, $\kappa_2 = 0.1\text{W/m/°K}$, Brick $t_3 = 0.22\text{m}$, $\kappa_3 = 1.7\text{W/m/°K}$.

The unknowns are temperatures T_1, T_2 at nodes 2, 3, $T_3 = 0°\text{C}$ at node 1 and $T_4 = 20°\text{C}$ at node 4 are known.

The conductivity matrices are obtained readily from (8.17): Element 1

$$[K]^{(e)} = \frac{S\kappa_1}{t_1}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} = \frac{S0.7}{0.02}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} ,$$

element 2

$$[K]^{(e)} = \frac{S\kappa_2}{t_2}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} = \frac{S0.1}{0.12}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} ,$$

and element 3

$$[K]^{(e)} = \frac{S\kappa_3}{t_3}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} = \frac{S1.7}{0.22}\begin{bmatrix} 1, & -1 \\ -1, & 1 \end{bmatrix} ,$$

The conductivity matrix of the structure is assembled from the elementwise matrices above as

$$[K] = S\begin{bmatrix} \dfrac{0.1}{0.12} + \dfrac{0.7}{0.02}, & -\dfrac{0.1}{0.12} \\ -\dfrac{0.1}{0.12}, & \dfrac{0.1}{0.12} + \dfrac{1.7}{0.22} \end{bmatrix} ,$$

The heat load vector generated by the essential boundary conditions is assembled from the elementwise contributions: element 1

$$[L]^{(e)} = -\frac{S\kappa_1}{t_1}\begin{bmatrix} 1 \\ -1 \end{bmatrix} T_3 = -\frac{S\kappa_1}{t_1}\begin{bmatrix} 1 \\ -1 \end{bmatrix} \times 0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix} ,$$

nothing from element 2, and from element 3

$$[L]^{(e)} = -\frac{S\kappa_3}{t_3}\begin{bmatrix} -1 \\ 1 \end{bmatrix} T_4 = -\frac{S1.7}{0.22}\begin{bmatrix} -1 \\ 1 \end{bmatrix} \times 20 ,$$

The heat load vector of the structure is therefore assembled as

$$[L] = -\frac{S1.7}{0.22}\begin{bmatrix} 0 \\ -1 \end{bmatrix} \times 20 ,$$

The solution of the discrete equations reads

$$\begin{bmatrix} T_1 \\ T_2 \end{bmatrix} = \begin{bmatrix} 0.42°\mathrm{C} \\ 18.09°\mathrm{C} \end{bmatrix}$$

Represented graphically the temperature variation shows the characteristic drop off of the temperature through the insulation layer (perlite), see Figure 8.7. It is worth noting that the finite element solution is for the present model exact since the L2 elements can exactly represent linear variation of temperature.

8.5 Reducing the model dimension to two

In this section we show how the originally three-dimensional model can be reduced to just two active coordinates. The reduced model will still describe the heat conduction through a *three-dimensional domain*; the function describing the temperature distribution will depend only on *two* spatial coordinate variables though.

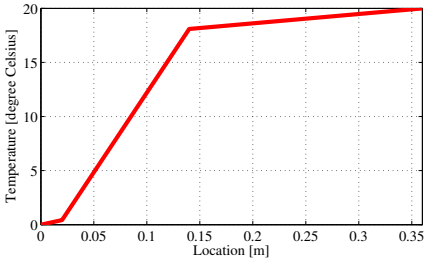


Fig. 8.7. Layered wall: variation of temperature

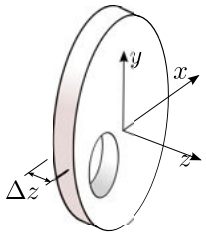


Fig. 8.8. Slice of a long cylindrical structure

For some physical situations we can make the observation that the temperature does not vary significantly along one coordinate direction, say along the z direction. Figure 8.8 shows a disk of thickness Δz. It is a slice of a structure of an unchanging cross-section which is very along in the z direction compared to the transverse dimensions. If we can neglect what is happening near the end sections, and if the component of the temperature gradient along the z direction is negligible, $\partial T/\partial z \approx 0$, a necessary condition for the formulation of a simplified model is met. However, it is not a sufficient condition as it does not necessarily mean that the z component of the heat flux is also zero: the partial derivatives $\partial T/\partial x$, and $\partial T/\partial y$ multiply the first two columns in row three of (7.14) to yield

$$q_z = \kappa_{zx}\partial T/\partial x + \kappa_{zy}\partial T/\partial y \ .$$

However, for the two classes of materials (7.15) and (7.16) the two coefficients κ_{zx} and κ_{zy} are identically zero, which means that if the temperature gradient $\partial T/\partial z$ is zero, the heat flux in that direction also vanishes.

Going back to Fig. 8.8: the heat flux through the cross sections is zero, and the temperature through the thickness of the disk is uniform (i.e. the temperature does not vary with z). The surface of the three-dimensional solid consists of the two cross sections, and of the cylindrical surfaces, the inner and the outer. The two cylindrical surfaces may be associated with boundary condition of any type. The two cross sections are associated with the boundary condition of zero heat flux, $\overline{q}_n = 0$ (type S_2, Eq. (7.20))

$$\boldsymbol{n} \cdot \boldsymbol{q} = \pm q_z = 0, \quad \text{on the cross sections} \ . \tag{8.19}$$

Since the temperature does not vary with z, the integrals (8.12) may be simplified by pre-integrating in the thickness direction, $\mathrm{d}V = \Delta z \ \mathrm{d}S$ and $\mathrm{d}S = \Delta z \ \mathrm{d}C$. The volume integrals are then evaluated over the cross-sectional area, S_c, (see Fig. 8.9); provided $\overline{q}_n$ and h are independent of z, the surface integrals are computed as integrals over the contour of the cross-section, C_c.

$$\begin{aligned} &\int_{S_c} \eta c_V \frac{\partial T}{\partial t} \ \Delta z \ \mathrm{d}S + \int_{S_c} (\mathrm{grad}\eta) \ \boldsymbol{\kappa} (\mathrm{grad}T)^T \ \Delta z \ \mathrm{d}S - \int_{S_c} \eta Q \ \Delta z \ \mathrm{d}S \\ &+ \int_{C_{c,2}} \eta \ \overline{q}_n \ \Delta z \ \mathrm{d}C + \int_{C_{c,3}} \eta \ h(T - T_a) \ \Delta z \ \mathrm{d}C = 0, \\ &\eta(\boldsymbol{x}) = 0 \text{ for } \boldsymbol{x} \in C_{c,1} \ . \end{aligned} \tag{8.20}$$

Note that the thickness Δz is a constant and could cancel without any effect on the solution. Nevertheless, Eq. (8.20) still applies to a fully three-dimensional body. To maintain this notion throughout the book, we shall not cancel the thickness.

Note that (8.20) does not refer to z, except in the term $\partial./\partial z$. We know that the temperature does not depend on z, and concerning the gradient of η: we simply assume that η does not depend on z: $\eta = \eta(x, y)$. The last assumption completes the reduction of the problem to two dimensions: all the functions depend on x and y only.

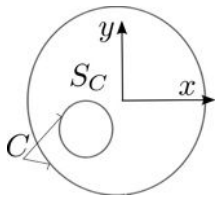


Fig. 8.9. Diagram of the heat conduction model

8.6 Test and trial functions: basis functions on triangulations

It is time to talk about the test and trial function. They are both functions of x and y only, $\eta = \eta(x, y)$ and $T = T(x, y, t)$ (and for the trial function, time). The only difference between them is the value they assume on one part of the boundary (which is a part of the cross-section contour, for our two-dimensional disk) where the temperature is being prescribed, $C_{c,1}$:

$$\text{Trial function: } T(\boldsymbol{x}, t) = \overline{T}(\boldsymbol{x}, t), \quad \text{Test function: } \eta(\boldsymbol{x}) = 0\ , \quad \boldsymbol{x} \text{ on } C_{c,1}\ .$$

Let us consider first the test function. It needs to be defined as a function of x and y over arbitrarily shaped domains. The concept of piecewise linear functions defined over tilings of arbitrary domains into triangles is quite ancient (at least in terms of the development of computational mechanics). The so-called "linear triangle" made its first appearance in a lecture by Courant in 1943, applied to Poisson's equation, which is a time-independent version of the heat conduction equation of this chapter. It was then picked up as a structural element in aerospace engineering to model Delta wing skin panels, as described in the 1956 paper by Turner, Clough, Martin and Topp. Clough then applied the triangle to problems in civil engineering, and he also coined the terminology "finite element". The triangle with three nodes is the simplest finite element in more than one coordinate. In this book we will call this element T3 (Triangle with 3 nodes).

The domain of the disk with a hole (shown in Fig. 8.9) is approximated as a collection of triangles (in other words, it is *tiled* with triangles, or *triangulated*), see Fig. 8.10. The mesh consisting of triangles is typically called ***triangulation***, even though sometimes *any* mesh is called that. The vertices of the triangulation are called ***nodes*** (compare with Section 2.9), while the line segments connecting the nodes are called ***edges***. Evidently, the triangles are the finite elements.

Interpolation on the triangle mesh will be treated as a linear combination of "tent" functions. Each individual tent is formed by grabbing one of the nodes (say J) and raising it out of the plane of the triangulation (traditionally to a unit height). The tent canvas is stretched over the edges that connect at the node J, and are clamped down by the ring of the edges that surround node J. The cartoon of one particular basis function tent is shown in Fig. 8.11. For those who do not like tents (perhaps it rained a lot during the summer camp), the term *hat function* may be preferable.

All the triangles that are connected in the node J ***support*** the function N_J, which is another way of saying that the function N_J is nonzero in these triangles; it is defined to be zero everywhere else. (If we are inside the "tent", we are standing on the support of the function.) Mathematically, the support of the basis function N_J is

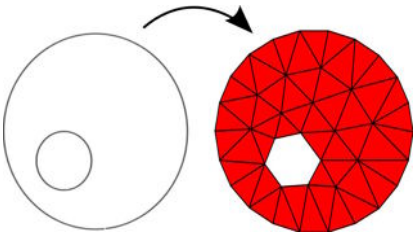


Fig. 8.10. Mesh of the disk domain

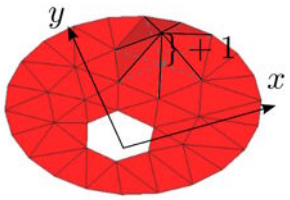


Fig. 8.11. Visual representation of one basis function on the mesh of the disk

$$\text{supp}N_J = \{\boldsymbol{x} : N_J(\boldsymbol{x}) \neq 0\} \ .$$

Since the set $\text{supp}N_J$ is a finite piece of the (typically finite) computational domain, it is also called a ***compact support***. The compact supports of the finite element basis functions make the finite element matrices sparse, and hence are crucial for the efficiency of these methods.

It remains to write down the equations that define the function N_J at any point within its support. That means writing an expression for each triangle within the support separately. Referring to Fig. 8.11, there are only three such functions: the three basis functions associated with the nodes at the corners of the element; all the other basis functions in the mesh are identically zero over this element. Thus, our task is to write down the expressions for the three basis functions over the domain of a single triangle.

8.7 Basis functions on the standard triangle

Each of the three basis functions is zero along one edge of the triangle: again, refer to Fig. 8.11. The task is accomplished most readily when the triangle is in a special position with respect to the coordinates: the ***standard triangle***; see Fig. 8.12. The basis functions associated with nodes ②and ③ are simply

$$N_2(\xi, \eta) = \xi \ , \tag{8.21}$$

and

$$N_3(\xi, \eta) = \eta \ . \tag{8.22}$$

As is easily verified, N_2 is zero along the edge ①③, and assumes value $+1$ at node ②; analogous properties hold for N_3. If N_1 should be equal to $+1$ at the origin, it must be written as

$$N_1(\xi, \eta) = 1 - \xi - \eta \ . \tag{8.23}$$

Clearly, N_1 vanishes at the edge opposite node ①. Thus, we see that the three functions we just formulated satisfy the Kronecker delta property, equation (2.23). As in Section 2.9, this means the degree of freedom at each node of the triangle is the value of the interpolated function at the node.

Also, we have the following property of the ***partition of unity***

$$\sum_{k=1}^{3} N_k(\xi,\eta) = 1 \ , \tag{8.24}$$

which should be interpreted in this sense: the basis functions "partition" +1 at any point within the triangle, and we will make use of this property later to show which functions will be reproduced exactly when interpolated over an element.

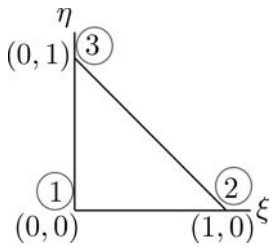


Fig. 8.12. Standard triangle

As the three functions satisfy the Kronecker delta property (2.23), the degree of freedom at each node of the triangle is the value of the interpolated function at the node, $T_i = T(\boldsymbol{x}_i)$. Therefore, we may make the observation that data that sit at the corners of the triangle are ***naturally interpolated***. One particularly useful quantity that one can interpolate on the standard triangle are the Cartesian coordinates of the corners in the physical space,

$$\boldsymbol{x} = \sum_{i=1}^{3} N_i(\xi,\eta)\boldsymbol{x}_i \ , \tag{8.25}$$

where the result of the interpolation is a point in the Cartesian coordinates

$$\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix} \ ,$$

and

$$\boldsymbol{x}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad i = 1, 2, 3 \ ,$$

are the coordinates of the three points that are being interpolated. Finite elements on which the geometrical coordinates $\boldsymbol{x}$ are interpolated in exactly the same way as the variable(s) of the PDE (deflection of the cable, temperature, displacements, and so on) are called ***isoparametric elements***.

Exercise 30.

Formulate the L2 element from the previous chapters as an isoparametric element.

Solution: The standard shape for the L2 finite element will be the bi-unit interval $-1 \leq \xi \leq +1$. The basis functions on the standard interval were written in equation (4.7). Now they will be used to define interpolation of the geometric coordinate

$$x = \sum_{i=1}^{2} N_i(\xi)x_i \ ,$$

where node j is at location x_j ($x_1 < x_2$ so that the length of the element is positive). The deflection (for the prestressed wire model) or the temperature (for the heat conduction model) are interpolated over an element using the *same* basis functions

$$w = \sum_{i=1}^{2} N_i(\xi) w_{\langle i \rangle} = \sum_{i=1}^{2} N_{\langle i \rangle}(\xi) w_i \ , \quad \text{or} \quad T = \sum_{i=1}^{2} N_i(\xi) T_{\langle i \rangle} = \sum_{i=1}^{2} N_{\langle i \rangle}(\xi) T_i \ .$$

Substituting the expressions for the basis functions into the expression for x we obtain

$$x = \frac{\xi - 1}{-2} x_1 + \frac{\xi + 1}{+2} x_2 = \frac{1}{2}(x_1 + x_2) + \frac{1}{2}(x_2 - x_1)\xi$$

which is precisely the map of the standard interval (4.2) to the physical interval $x_1 \le x \le x_2$ we have introduced for the purpose of numerical integration. Clearly the element L2 was an isoparametric element all along, we have just not explicitly labeled it as such.

Equation (8.25) is a mapping from the pair ξ, η to the point x, y. Substituting for the basis functions, it may be written explicitly as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} (x_2 - x_1) & (x_3 - x_1) \\ (y_2 - y_1) & (y_3 - y_1) \end{bmatrix} \begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \ . \tag{8.26}$$

This matrix equation is accompanied by the picture in Fig. 8.13. The two vectors, $\boldsymbol{v}$ and $\boldsymbol{w}$, are the two columns of the square matrix in (8.26):

$$\boldsymbol{v} = \begin{bmatrix} (x_2 - x_1) \\ (y_2 - y_1) \end{bmatrix} \ , \quad \boldsymbol{w} = \begin{bmatrix} (x_3 - x_1) \\ (y_3 - y_1) \end{bmatrix} \ . \tag{8.27}$$

If both ξ and η vary between zero and one, equation (8.26) adds the two vectors, $\xi\boldsymbol{v}$ and $\eta\boldsymbol{w}$ to the vector $[x_1, y_1]^T$, and the result then covers the entire parallelogram; on the other hand, if ξ and η are confined to the interior of the standard triangle, Eq. (8.26) produces points to cover the area of the filled triangle. To summarize, Eq. (8.26) is a ***map*** from the standard triangle to a triangle in the Cartesian coordinates with corners in given locations.

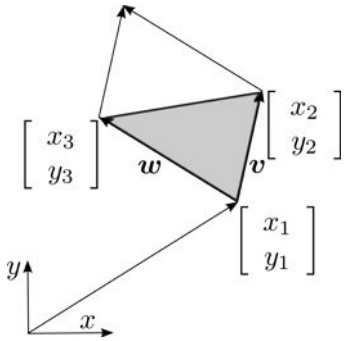


Fig. 8.13. Interpolating Cartesian coordinates on the standard triangle

Inverting (8.26) to express ξ and η, which could then be substituted into (8.21) – (8.23) to produce basis functions in terms of x and y, may look appealing but should be resisted. The reason is that numerical quadrature is available on the standard triangle and is much harder on general triangles. This will become especially clear with quadratic elements later in the book.

However, since Eq. (8.26) is an invertible map from the standard triangle to a triangle in the Cartesian coordinates (invertibility follows if the triangle does not have its corners in a single straight-line: why?), we do get an algorithm for *evaluating basis functions on a general triangle*. Given a point $\bar{x}, \bar{y}$ in the Cartesian coordinates, and within the bounds of a triangle, we can use the inverse of the map (8.26) to obtain point $\bar{\xi}, \bar{\eta}$ in the standard triangle (path 1 in Fig. 8.14). Therefore, we may then evaluate $N_K(\bar{\xi}, \bar{\eta})$, which is the value $N_K(\bar{x}, \bar{y})$ (path 2 in Fig. 8.14). That may seem awkward,

and it is. However, this is normally not needed as the usual operation on the triangulation is to evaluate the basis functions in order to perform numerical quadrature, that is at a particular point (quadrature point) within the *standard* triangle. In that case, $\bar{\xi}, \bar{\eta}$ would be *known* (and $\bar{x}, \bar{y}$ would be unknown), and calculation of the function value is easy. Evaluation of the *derivatives* of the basis functions is a little bit more complex, and will be therefore discussed separately in the section on numerical quadrature.

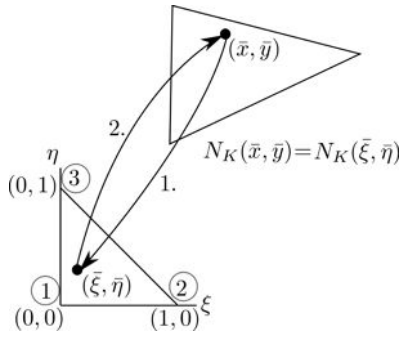


Fig. 8.14. Using the map from the standard triangle to evaluate basis functions over a general triangle

We understand now that each node in the mesh is associated with a single basis function. In the following, whenever we write

$$N_i = N_i(x, y) \ ,$$

it has to be understood that within each triangle in the mesh, the coordinates of the point are given as $x = x(\xi, \eta)$, $y = y(\xi, \eta)$, where ξ and η are coordinates in the standard triangle.

In this book will formulate the basis functions on an element of standard shape first, and only then map them to the general shape. Nevertheless, it will be instructive to have a look at the alternative, the direct construction of the basis functions and computation of their derivatives for triangles of general shape.

8.8 Direct construction of the T3 basis functions

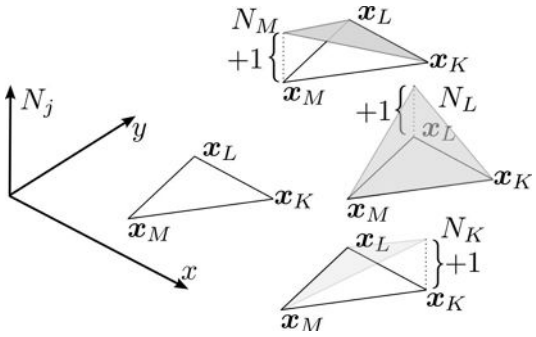


Fig. 8.15. Directly computing the basis functions on a general triangle

For the T3 triangle we can use the Kronecker property to construct the basis functions. As shown in Figure 8.15 for the general triangle $\triangle KLM$ each basis function assumes a value 1 at one node, and value 0 at the remaining two nodes. If we write down these three conditions for instance for basis function N_K

$$N_K(x_K, y_K) = 1 \,, \quad N_K(x_L, y_L) = 0 \,, \quad N_K(x_M, y_M) = 0 \,,$$

we see that these represent three conditions from which three coefficients may be determined. It so happens that a linear function in two coordinates has three coefficients. Therefore we can anticipate N_K to be of the form

$$N_K(x, y) = a_K x + b_K y + c_K$$

which upon substitution into the three Kronecker conditions yields

$$\begin{aligned} &N_K(x_K, y_K) = a_K x_K + b_K y_K + c_K = 1 \,, \\ &N_K(x_L, y_L) = a_K x_L + b_K y_L + c_K = 0 \,, \\ &N_K(x_M, y_M) = a_K x_M + b_K y_M + c_K = 0 \,, \end{aligned}$$

This can be written in matrix form as

$$\begin{bmatrix} x_K, & y_K, & 1 \\ x_L, & y_L, & 1 \\ x_M, & y_M, & 1 \end{bmatrix} \begin{bmatrix} a_K \\ b_K \\ c_K \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}$$

The three coefficients may be easily obtained by matrix inverse. For instance Matlab symbolic algebra yields

```
syms x y  x_K   y_K  x_L   y_L  x_M   y_M real
inv( [x_K  y_K  1 ;
    x_L  y_L  1 ;
    x_M  y_M  1])*[1;0;0]
ans =

         (y_L - y_M)/(x_K*y_L - x_L*y_K - x_K*y_M + x_M*y_K + x_L*y_M - x_M*y_L)
        -(x_L - x_M)/(x_K*y_L - x_L*y_K - x_K*y_M + x_M*y_K + x_L*y_M - x_M*y_L)
 (x_L*y_M - x_M*y_L)/(x_K*y_L - x_L*y_K - x_K*y_M + x_M*y_K + x_L*y_M - x_M*y_L)
```

In the first to third row we have a_K, b_K, c_K. Repeating this for the other two basis functions is entirely analogous. In fact we can see that we could get the 3×3 coefficients in one fell swoop. For N_L the Kronecker conditions may be written in matrix form as

$$\begin{bmatrix} x_K, & y_K, & 1 \\ x_L, & y_L, & 1 \\ x_M, & y_M, & 1 \end{bmatrix} \begin{bmatrix} a_L \\ b_L \\ c_L \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

and for N_M they are

$$\begin{bmatrix} x_K, & y_K, & 1 \\ x_L, & y_L, & 1 \\ x_M, & y_M, & 1 \end{bmatrix} \begin{bmatrix} a_M \\ b_M \\ c_M \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We can see that the matrix on the left is unchanged. The right-hand side does change, and we can accommodate both features by writing all nine Kronecker conditions together as

$$\underbrace{\begin{bmatrix} x_K, & y_K, & 1 \\ x_L, & y_L, & 1 \\ x_M, & y_M, & 1 \end{bmatrix}}_{\boldsymbol{X}} \underbrace{\begin{bmatrix} a_K, & a_L, & a_M \\ b_K, & b_L, & b_M \\ c_K, & c_L, & c_M \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} 1, & 0, & 0 \\ 0, & 1, & 0 \\ 0, & 0, & 1 \end{bmatrix}}_{\mathbf{1}} \tag{8.28}$$

The coefficients we are looking for are therefore obtained as

$$\boldsymbol{A} = \boldsymbol{X}^{-1} \,.$$

Exercise 31. Construct the basis functions on the standard triangle using the direct computation of (8.28).

Solution: For the standard triangle the corners (locations of the nodes) are in the ξ, η coordinates at

$$(0,0), \quad (1,0), \quad (0,1)$$

Substituting into the matrix $\boldsymbol{X}$ we obtain

$$\boldsymbol{X} = \begin{bmatrix} 0, & 0, & 1 \\ 1, & 0, & 1 \\ 0, & 1, & 1 \end{bmatrix}$$

and its inverse reads

$$\boldsymbol{A} = \boldsymbol{X}^{-1} = \begin{bmatrix} -1, & 1, & 0 \\ -1, & 0, & 1 \\ 1, & 0, & 0 \end{bmatrix}$$

In each column we have the three coefficients of the linear polynomial in ξ, η. Clearly we obtain equations (8.23), (8.21), and (8.22).

The derivatives of the basis functions are the components of the ***basis function gradient***. A gradient can be visualized as the normal to the level curves (level surfaces) of the function. It points in the direction in which the function increases. Figure 8.16 shows the level curves of basis function N_J. Note that the level curve $N_J = 0$ is the boundary of the support of the basis function. It runs along the edges of the triangles that are connected to node J. Also shown is the level curve $N_J = 1$ which collapses to a point at node J. Finally, an intermediate level curve at 0.6 is shown. The gradient at one point of the level curve $N_J = 0$ is visualized with an arrow. Note that the arrow points in the direction in which the function increases.

For the basis function N_J we can write

$$N_J(x,y) = a_J x + b_J y + c_J$$

for each flat piece of surface of which it consists. In other words, the coefficients a_J, b_J, c_J vary from triangle to triangle. In each triangle the gradient of the basis function is constant. We see this by computing the gradient as

$$\text{grad}N_J(x,y) = \left[\frac{\partial N_J}{\partial x}, \frac{\partial N_J}{\partial y}\right] = [a_J, b_J] \ . \tag{8.29}$$

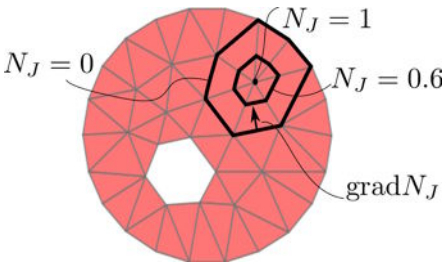
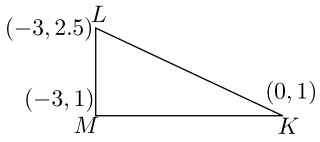


Fig. 8.16. Level curves and gradients of the basis function N_J

Exercise 32.

Compute the gradients of the three basis functions of the triangle $\triangle KLM$ shown below.



Solution: The matrix $\boldsymbol{X}$ reads

$$\boldsymbol{X} = \begin{bmatrix} 0, & 1, 1 \\ -3, & 2.5, 1 \\ -3, & 1, 1 \end{bmatrix}$$

and its inverse follows as

$$\boldsymbol{A} = \boldsymbol{X}^{-1} = \begin{bmatrix} 1/3, & 0, & -1/3 \\ 0, & 2/3, & -2/3 \\ 1, & -2/3, & 2/3 \end{bmatrix}$$

In the first row of matrix $\boldsymbol{A}$ are the a_j coefficients, in the second row are the b_j coefficients. As we have seen above, these two sets of coefficients are the components of the gradients. For instance,

$$\text{grad}N_L(x,y) = \left[\frac{\partial N_L}{\partial x}, \frac{\partial N_L}{\partial y}\right] = [a_L, b_L] = [0, 2/3] \; .$$

The gradients of all three basis functions may be written down in one single expression by taking the first two rows of $\boldsymbol{A}$ and transposing the resulting submatrix:

$$\begin{bmatrix} \text{grad}N_K(x,y) \\ \text{grad}N_L(x,y) \\ \text{grad}N_M(x,y) \end{bmatrix} = \boldsymbol{A}\texttt{(1:2,:)'} = \begin{bmatrix} 1/3, & 0 \\ 0, & 2/3 \\ -1/3, & -2/3 \end{bmatrix}$$

Here we use (abuse?) the Matlab notation `(1:2,:)'`: take the first two rows and transpose them.

Note that incidentally for this triangle it is particularly easy to verify that the gradients have been computed correctly by taking the ratios of rise over run. For instance for the basis function N_M in the x direction we take rise -1 and run 3 along the edge MK, which gives $-1/3$. In the y direction we take rise -1 and run 1.5 along the edge ML, which gives $-2/3$.

8.9 Discretizing the weighted residual equation

The trial function will be expressed using the basis functions as (compare with Section 2.9) (recall that (i) means the degree of freedom associated with node i)

$$T(x,y,t) = \sum_{i=1}^{N} N_i(x,y) T_{(i)}(t) \; , \tag{8.30}$$

where the sum ranges over all the basis functions (i.e. over all the nodes in the mesh). Included are also basis functions associated with the nodes on the boundary where the temperature is being prescribed, $C_{c,1}$. On the contrary, these nodes do not contribute basis functions to the set of the test functions (these are expected to vanish along $C_{c,1}$). Therefore, we will choose

$$\eta(x,y) = N_i(x,y), \quad i \text{ excluded when node } i \in C_{c,1} \; .$$

The nodes whose basis functions are not part of the linear combination for the test function are shown as empty circles in Fig. 8.17.

As before for the wire model, we shall adopt the following notation:

$$\eta(x,y) = N_{\langle i\rangle}(x,y), \quad i = 1, ..., N_{\mathrm{f}},$$

where N_{f} is the number of unknown degrees of freedom, and

$$T(x,y,t) = \sum_{i=1}^{N} N_{\langle i\rangle}(x,y) T_i(t) \ ,$$

In addition, because the basis on the standard triangle satisfies the Kronecker delta property (2.23), the values of the degrees of freedom $T_i(t)$ at the nodes "prescribed i" (the nodes with the empty circles in Fig. 8.17) are simply the values of the interpolated prescribed temperature at the nodes, $T_i(t) = \overline{T}(x_i, y_i, t)$.

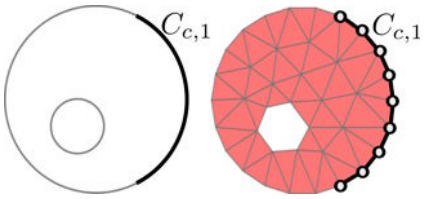


Fig. 8.17. Approximation of the boundary by the edges of the triangulation and the nodes on the boundary

Remark: Apropos curved boundaries: Figure 8.17 clearly shows that with straight edges we are only approximating any boundaries that are curved. Some error is involved, but fortunately we are able to control this error by reducing the length of the edges.

The finite element expansions for the trial and test functions are now substituted into the weighted residual integral (8.20). For clarity, the substitution will be shown term-by-term (henceforth we will omit the arguments):

$$\int_{S_c} \eta c_V \frac{\partial T}{\partial t} \, \Delta z \, \mathrm{d}S = \int_{S_c} N_{\langle j\rangle} c_V \sum_{i=1}^{N} N_{\langle i\rangle} \frac{\partial T_i}{\partial t} \, \Delta z \, \mathrm{d}S \ , \quad j = 1, ..., N_{\mathrm{f}}, \ , \tag{8.31}$$

which simplifies to

$$\sum_{i=1}^{N} \left[\int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle} \, \Delta z \, \mathrm{d}S \right] \frac{\partial T_i}{\partial t} \ , \quad j = 1, ..., N_{\mathrm{f}} \ . \tag{8.32}$$

The term in the bracket mixes together i and j from two different sets, and some of the degrees of freedom $\partial T_i/\partial t$ are known. Therefore, separating the known and unknown quantities may be a good idea:

$$\begin{aligned} &\sum_{i=1}^{N} \left[\int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle} \, \Delta z \, \mathrm{d}S \right] \frac{\partial T_i}{\partial t} = \\ &\sum_{i=1}^{N_{\mathrm{f}}} \left[\int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle} \, \Delta z \, \mathrm{d}S \right] \frac{\partial T_i}{\partial t} + \sum_{i=N_{\mathrm{f}}+1}^{N} \left[\int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle} \, \Delta z \, \mathrm{d}S \right] \frac{\partial \overline{T}_i}{\partial t} \ , \quad j = 1, ..., N_{\mathrm{f}} \ , \end{aligned} \tag{8.33}$$

where we indicate by the barred $\partial \overline{T}_i/\partial t$ that for $i = N_{\mathrm{f}} + 1, ..., N$ these quantities are prescribed.

The first integral on the right-hand side of (8.33) suggests defining a square matrix

$$C_{ji} = \int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle} \, \Delta z \, \mathrm{d}S \ , \quad i, j = 1, ..., N_{\mathrm{f}} \ , \tag{8.34}$$

the ***capacity matrix***. This matrix is multiplied by the vector of the unknown degrees of freedom.

The integral in the second term will be given a different symbol, since the meaning of the two terms is different. We define

$$L_{\overline{C},j} = -\sum_{i=N_\mathrm{f}+1}^{N} \left[\int_{S_c} N_{\langle j\rangle} c_V N_{\langle i\rangle}\ \Delta z\ \mathrm{d}S\right] \frac{\partial \overline{T}_i}{\partial t}\ , \quad j = 1, ..., N_\mathrm{f}. \tag{8.35}$$

as a contribution to a heat load (the known "right-hand side"). We will call this the contribution of the prescribed temperatures: ***essential boundary condition heat load***.

Next, the second term in (8.20):

$$\begin{aligned}
&\int_{S_c} (\mathrm{grad}\eta)\ \boldsymbol{\kappa}(\mathrm{grad}T)^T\ \Delta z\ \mathrm{d}S = \int_{S_c} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa}(\mathrm{grad}\sum_{i=1}^{N} N_{\langle i\rangle}T_i)^T\ \Delta z\ \mathrm{d}S = \\
&\sum_{i=1}^{N_\mathrm{f}} \left[\int_{S_c} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa}(\mathrm{grad}N_{\langle i\rangle})^T\ \Delta z\ \mathrm{d}S\right] T_i + \\
&\sum_{i=N_\mathrm{f}+1}^{N} \left[\int_{S_c} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa}(\mathrm{grad}N_{\langle i\rangle})^T\ \Delta z\ \mathrm{d}S\right] \overline{T}_i \qquad j = 1, ..., N_\mathrm{f},
\end{aligned} \tag{8.36}$$

and the ***conductivity matrix*** may be defined as

$$K_{ji} = \int_{S_c} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa}(\mathrm{grad}N_{\langle i\rangle})^T\ \Delta z\ \mathrm{d}S\ , \qquad i, j = 1, ..., N_\mathrm{f}\ . \tag{8.37}$$

The contribution to the heat load vector due to the second term on the right of (8.36) (the prescribed-temperature essential boundary condition heat load) reads

$$L_{\overline{K},j} = -\sum_{i=N_\mathrm{f}+1}^{N} \left[\int_{S_c} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa}(\mathrm{grad}N_{\langle i\rangle})^T\ \Delta z\ \mathrm{d}S\right] \overline{T}_i\ , \quad i, j = 1, ..., N_\mathrm{f}. \tag{8.38}$$

Next, the load term corresponding to the ***internal heat generation***:

$$L_{Q,j} = \int_{S_c} N_{\langle j\rangle} Q\ \Delta z\ \mathrm{d}S\ , \quad j = 1, ..., N_\mathrm{f}\ . \tag{8.39}$$

Almost done: the term corresponding to natural boundary condition. On the $C_{c,2}$ part of the boundary, only a load term results

$$L_{q2,j} = -\int_{C_{c,2}} N_{\langle j\rangle}\ \overline{q}_n\ \Delta z\ \mathrm{d}C\ . \tag{8.40}$$

Finally, on the $C_{c,3}$ part of the boundary, where the heat flux is proportional to the difference between the ambient temperature and the surface temperature, we get an ***ambient-temperature load term***

$$L_{q3,j} = \int_{C_{c,3}} N_{\langle j\rangle}\ hT_a\ \Delta z\ \mathrm{d}C\ , \quad j = 1, ..., N_\mathrm{f}\ , \tag{8.41}$$

and a ***surface heat transfer matrix***:

$$H_{ji} = \int_{C_{c,3}} N_{\langle j\rangle}\ hN_{\langle i\rangle}\ \Delta z\ \mathrm{d}C\ , \quad i, j = 1, ..., N_\mathrm{f}\ , \tag{8.42}$$

and one more essential boundary condition load term

$$L_{\overline{H},j} = -\sum_{i=N_\mathrm{f}+1}^{N} \left[\int_{C_{c,3}} N_{\langle j\rangle}\ hN_{\langle i\rangle}\ \Delta z\ \mathrm{d}C\right] \overline{T}_i\ , \quad j = 1, ..., N_\mathrm{f}\ . \tag{8.43}$$

To summarize, using the definitions of the various matrices and load terms, the system of ordinary differential equations that results from the introduction of the finite element test and trial functions (the so-called discretization in space) reads

$$\sum_{i=1}^{N_\mathrm{f}} C_{ji}\frac{\partial T_i}{\partial t} + \sum_{i=1}^{N_\mathrm{f}} K_{ji}T_i + \sum_{i=1}^{N_\mathrm{f}} H_{ji}T_i = L_{\overline{C},j} + L_{\overline{K},j} + L_{\overline{H},j} + L_{Q,j} + L_{q2,j} + L_{q3,j} \qquad (8.44)$$
$$j = 1, ..., N_\mathrm{f}\ .$$

It is easy to verify the physical units of the quantities on the right: they are all in the units of power (energy per unit time). We can say that for instance $L_{Q,j}$ it is a *nodal power* corresponding to internal heat generation rate. Consequently also the quantities on the left must have the same physical units. For instance the physical units of $K_{ji}T_i$ are worked out as $T_i[^\circ\mathrm{K}]$ and

$$K_{ji}[?] = \int_{S_c} (\mathrm{grad}N_{\langle j\rangle})[\mathrm{m}^{-1}]\ \boldsymbol{\kappa}[\mathrm{W}/(\mathrm{m}^{-1}\cdot{}^\circ\mathrm{K})](\mathrm{grad}N_{\langle i\rangle})^T[\mathrm{m}^{-1}]\ \Delta z[\mathrm{m}]\ \mathrm{d}S[\mathrm{m}^2] = [\mathrm{W}/^\circ\mathrm{K}]$$

so that $K_{ji}T_i[\mathrm{W}]$ as required.

Before we talk about ways of solving the set of ordinary differential equations (8.44) we need to tie up a few loose ends. Especially the computation of the derivatives of the basis functions, but also numerical integration, the constitutive equation, and the evaluation of the surface terms.

8.10 Derivatives of the basis functions; Jacobian

The results of this section are much more general than may be expected. While the formulas for the derivatives of basis functions are derived for the linear triangles, the same formulas (and implementation) is used for all the so-called ***isoparametric*** elements in the `FAESOR` toolbox.

To evaluate the conductivity matrix, we need to be able to calculate the derivatives of the basis functions with respect to x and y. Equations (8.21–8.23) define the functions over the standard triangle in terms of ξ and η. Therefore, to express $\partial N_i/\partial x$ we use the chain rule

$$\frac{\partial N_i}{\partial x} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial x} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial x}\ ,$$
$$\frac{\partial N_i}{\partial y} = \frac{\partial N_i}{\partial \xi}\frac{\partial \xi}{\partial y} + \frac{\partial N_i}{\partial \eta}\frac{\partial \eta}{\partial y}\ .$$

For the purpose of this discussion, the function that is being differentiated does not really matter. We will replace it with a ♡, while we arrange the above equation into a matrix expression

$$\left[\frac{\partial \heartsuit}{\partial x},\ \frac{\partial \heartsuit}{\partial y}\right] = \left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]\begin{bmatrix}\dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y}\\[2ex] \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y}\end{bmatrix} = \left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]\left[\widetilde{J}\right]\ . \qquad (8.45)$$

The derivatives are arranged in row matrices because these objects are ***gradients*** of the ♡ function [compare with (7.17)]. This

$$\left[\frac{\partial \heartsuit}{\partial \xi},\ \frac{\partial \heartsuit}{\partial \eta}\right]$$

is the gradient of the function ♡ with respect to the coordinates ξ, η and this is the gradient of the same function with respect to coordinates x, y

$$\left[\frac{\partial \heartsuit}{\partial x},\ \frac{\partial \heartsuit}{\partial y}\right]\ .$$

The matrix

$$\left[\widetilde{J}\right] = \begin{bmatrix} \dfrac{\partial \xi}{\partial x} & \dfrac{\partial \xi}{\partial y} \\ \dfrac{\partial \eta}{\partial x} & \dfrac{\partial \eta}{\partial y} \end{bmatrix} , \tag{8.46}$$

is the ***Jacobian matrix*** of the mapping $\xi = \xi(x, y), \eta = \eta(x, y)$, which is the inverse of the map $x = x(\xi, \eta), y = y(\xi, \eta)$ of Eq. (8.26). The question is how to evaluate the partial derivatives of the type $\partial\xi/\partial x$, since the inverse of the map (8.26) is not known (at least not in general).

Here is an idea: If we start the chain rule from the other end (switching the role of the variables), we obtain

$$\left[\frac{\partial \heartsuit}{\partial \xi}, \frac{\partial \heartsuit}{\partial \eta} \right] = \left[\frac{\partial \heartsuit}{\partial x}, \frac{\partial \heartsuit}{\partial y} \right] \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} , \tag{8.47}$$

and inverting the Jacobian matrix in equation (8.45) we get

$$\left[\frac{\partial \heartsuit}{\partial \xi}, \frac{\partial \heartsuit}{\partial \eta} \right] = \left[\frac{\partial \heartsuit}{\partial x}, \frac{\partial \heartsuit}{\partial y} \right] \left[\widetilde{J}\right]^{-1} . \tag{8.48}$$

Comparing (8.47) and (8.48) yields

$$[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \xi} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \left[\widetilde{J}\right]^{-1} , \tag{8.49}$$

where $[J]$ is the ***Jacobian matrix*** of the map (8.26). The elements of $[J]$ are directly available from the matrix in (8.26). However, even more useful is to start from (8.25), and by definition the Jacobian matrix is then

$$[J] = \begin{bmatrix} \displaystyle\sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} x_i \ , \ \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} x_i \\ \displaystyle\sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} y_i \ , \ \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} y_i \end{bmatrix} . \tag{8.50}$$

Defining the matrix of gradients of the basis functions with respect to ξ, η

$$\left[\mathrm{grad}_{(\xi,\eta)} N\right] = \begin{bmatrix} \mathrm{grad}_{(\xi,\eta)} N_1 \\ \mathrm{grad}_{(\xi,\eta)} N_2 \\ \mathrm{grad}_{(\xi,\eta)} N_3 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} \ , \ \dfrac{\partial N_1}{\partial \eta} \\ \dfrac{\partial N_2}{\partial \xi} \ , \ \dfrac{\partial N_2}{\partial \eta} \\ \dfrac{\partial N_3}{\partial \xi} \ , \ \dfrac{\partial N_3}{\partial \eta} \end{bmatrix} . \tag{8.51}$$

and the matrix of gradients of the basis functions with respect to x, y (when there is no possibility of confusion we will use the simplified notation $\mathrm{grad} N_j = \mathrm{grad}_{(x,y)} N_j$)

$$\left[\mathrm{grad}_{(x,y)} N\right] = \begin{bmatrix} \mathrm{grad}_{(x,y)} N_1 \\ \mathrm{grad}_{(x,y)} N_2 \\ \mathrm{grad}_{(x,y)} N_3 \end{bmatrix} = \begin{bmatrix} \mathrm{grad} N_1 \\ \mathrm{grad} N_2 \\ \mathrm{grad} N_3 \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_1}{\partial x} \ , \ \dfrac{\partial N_1}{\partial y} \\ \dfrac{\partial N_2}{\partial x} \ , \ \dfrac{\partial N_2}{\partial y} \\ \dfrac{\partial N_3}{\partial x} \ , \ \dfrac{\partial N_3}{\partial y} \end{bmatrix} . \tag{8.52}$$

we can describe the computation of the ***gradients of the basis functions*** as

$$\left[\text{grad}_{(x,y)}N\right] = \left[\text{grad}_{(\xi,\eta)}N\right][J]^{-1} \tag{8.53}$$

The right-hand side is readily evaluated: the matrix $\left[\text{grad}_{(\xi,\eta)}N\right]$ is easily computed from the definition of the basis functions on the standard element (for the triangle from (8.23), (8.21), and (8.22)), and the Jacobian matrix follows from the definition (8.50).

Computationally, it may be easier to employ matrix multiplications instead of the summations in (8.50). Note that the Jacobian matrix may be expressed as the product of two matrices:

$$[J] = [\mathtt{x}]^T\,[\mathtt{Nder}]\ , \tag{8.54}$$

where [x] collects the coordinates of the nodes (three nodes, for the triangle)

$$[\mathtt{x}] = \begin{bmatrix} x_1\ , y_1 \\ x_2\ , y_2 \\ x_3\ , y_3 \end{bmatrix}\ , \tag{8.55}$$

and [Nder] collects in each row the gradient of the basis function with respect to the parametric coordinates

$$[\mathtt{Nder}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi}\ , \dfrac{\partial N_1}{\partial \eta} \\ \dfrac{\partial N_2}{\partial \xi}\ , \dfrac{\partial N_2}{\partial \eta} \\ \dfrac{\partial N_3}{\partial \xi}\ , \dfrac{\partial N_3}{\partial \eta} \end{bmatrix}\ . \tag{8.56}$$

The calculation of the spatial derivatives by an isoparametric geometric cell (recall that the finite elements in `FAESOR` encapsulate the calculation of basis functions and their derivatives in the `gcell` class) is a straightforward rewrite of the above formulas. The method `bfundsp` takes three arguments: a descendent of the class `gcell` (the objects on which a method is being invoked are by convention called `self` in `FAESOR`), and the two arrays (8.56) and (8.55). The dimensions of the two arrays are (line `0013`): `nbfuns`= number of basis functions (= 3 for the triangle), and `dim`= number of space dimensions (= 2 for the triangle).

```
0013 function Ndersp = bfundsp[1] (self, Nder, x)
0014     [nbfuns,dim] = size(Nder);
0015     if (size(Nder) ~= size(x))
0016         error('Wrong dimensions of arguments!');
0017     end
```

The Matlab code on line `0018` is literally the formula (8.54).

```
0018     J = x' * Nder;% Compute the Jacobian matrix
0019     Ndersp = Nder / J;% and evaluate the spatial gradients
0020 end
```

The generic case is treated in line `0019`, which transcribes equation (8.45) for each basis function at the same time by working with matrices: each row is the gradient of one basis function. Therefore, the result for the triangle is

$$[\mathtt{Ndersp}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial x}\ , \dfrac{\partial N_1}{\partial y} \\ \dfrac{\partial N_2}{\partial x}\ , \dfrac{\partial N_2}{\partial y} \\ \dfrac{\partial N_3}{\partial x}\ , \dfrac{\partial N_3}{\partial y} \end{bmatrix}\ . \tag{8.57}$$

[1] Folder: `FAESOR/classes/gcellset/@gcellset`

The Jacobian (determinant of the Jacobian matrix) should be positive. The Jacobian matrix would then be invertible. If it isn't invertible, the solution of the linear system on line `0019` will fail.

To round off the discussion in this section, we need to present the code that evaluates the basis functions (8.21–8.23) and the derivatives of the basis functions with respect to the parametric coordinates ξ, η. For the linear triangle T3 (class `gcellset_T3`) the two methods are delightfully simple: the method `bfun` computes a column array of the basis function values, N_j in row j, given the parametric coordinates $\xi \leftarrow$ `param_coords(1)`, $\eta \leftarrow$ `param_coords(2)`.

```
0008 function val = bfun[2](self,param_coords)
0009     val = [(1 - param_coords(1) - param_coords(2));...
0010         param_coords(1); ...
0011         param_coords(2)];
0012     return;
0013 end
```

The method `bfundpar` returns the array (8.56) with three rows (one for each basis function), with the gradient of the basis function j with respect to ξ, η in row j.

```
0010 function val = bfundpar[3](self, param_coords)
0011     val = [-1 -1; ...
0012             +1  0; ...
0013              0 +1];
0014     return;
0015 end
```

8.11 Numerical integration

Treading on the stepping stones of the discussion in Section 4.1, we formulate the numerical integration procedure for the linear triangle. We begin by highlighting the role of the Jacobian matrix. Consider a map from the parametric coordinates ξ, η to the Cartesian coordinates x, y: a slight

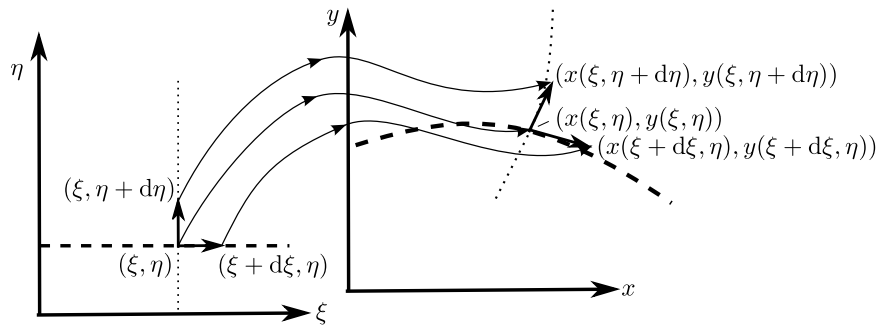


Fig. 8.18. Mapping of points for a general map between coordinates

generalization of (8.25) in that the map is not necessarily linear (see Fig. 8.18)

$$\boldsymbol{p} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x(\xi,\eta) \\ y(\xi,\eta) \end{bmatrix} \ . \tag{8.58}$$

So point (ξ, η) is mapped to $(x(\xi,\eta), y(\xi,\eta))$. Where does the point $(\xi + \mathrm{d}\xi, \eta)$ go? Here is a simple idea that uses the Taylor series, where assuming $\mathrm{d}\xi$ is a infinitesimally small quantity we will neglect all terms higher than first order:

[2]Folder: `FAESOR/classes/gcellset/@gcellset_T3`
[3]Folder: `FAESOR/classes/gcellset/@gcellset_T3`

$$x(\xi + \mathrm{d}\xi, \eta) = x(\xi, \eta) + \frac{\partial x(\xi, \eta)}{\partial \xi}\mathrm{d}\xi$$
$$y(\xi + \mathrm{d}\xi, \eta) = y(\xi, \eta) + \frac{\partial y(\xi, \eta)}{\partial \xi}\mathrm{d}\xi$$

The quantity

$$\begin{bmatrix} \dfrac{\partial x(\xi, \eta)}{\partial \xi} \\ \dfrac{\partial y(\xi, \eta)}{\partial \xi} \end{bmatrix}$$

is called the ***tangent vector*** to the curve $x(\xi, \bar{\eta}), y(\xi, \bar{\eta})$ along which ξ varies and the second argument is fixed at $\bar{\eta}$. The point $(x(\xi + \mathrm{d}\xi, \eta), y(\xi + \mathrm{d}\xi, \eta))$ is therefore obtained from the point $(x(\xi, \eta), y(\xi, \eta))$ by moving some distance (i.e. $\mathrm{d}\xi$) along the tangent vector to the ξ curve. Analogously we could figure out that the point $(\xi, \eta + \mathrm{d}\eta)$ in is mapped to

$$x(\xi, \eta + \mathrm{d}\eta) = x(\xi, \eta) + \frac{\partial x(\xi, \eta)}{\partial \eta}\mathrm{d}\eta$$
$$y(\xi, \eta + \mathrm{d}\eta) = y(\xi, \eta) + \frac{\partial y(\xi, \eta)}{\partial \eta}\mathrm{d}\eta$$

i.e. along the tangent vector to the η curve

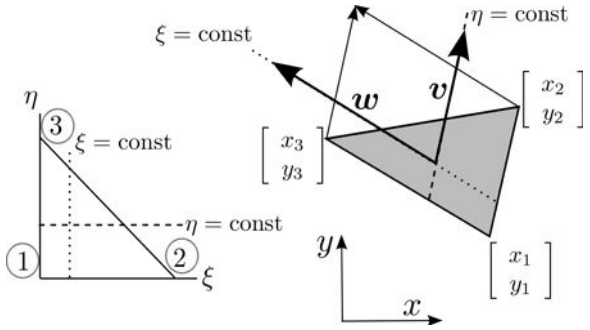
$$\begin{bmatrix} \dfrac{\partial x(\xi, \eta)}{\partial \eta} \\ \dfrac{\partial y(\xi, \eta)}{\partial \eta} \end{bmatrix}$$

Exercise 33. Consider the map (8.58) in the form of the T3 relationship between the standard and general triangles, as defined in equation (8.26). Compute the tangent vectors to the coordinate curves.

Solution: Equation (8.26) is readily differentiated with respect to ξ as

$$\frac{\partial}{\partial \xi}\begin{bmatrix} x \\ y \end{bmatrix} = \frac{\partial}{\partial \xi}\left(\begin{bmatrix} (x_2 - x_1), (x_3 - x_1) \\ (y_2 - y_1), (y_3 - y_1) \end{bmatrix}\begin{bmatrix} \xi \\ \eta \end{bmatrix} + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}\right) = \begin{bmatrix} (x_2 - x_1), (x_3 - x_1) \\ (y_2 - y_1), (y_3 - y_1) \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix}$$
$$= \begin{bmatrix} (x_2 - x_1) \\ (y_2 - y_1) \end{bmatrix}$$

This is the vector $\boldsymbol{v}$ from (8.27). Similarly as the derivative with respect to η we obtain $\boldsymbol{w}$ of (8.27). As illustrated in the figure below, the two vectors are tangents to curves $\eta = \text{const}$ (vector $\boldsymbol{v}$) and $\xi = \text{const}$ (vector $\boldsymbol{w}$).



Exercise 34. Consider the map

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{\xi+1}{2} R\cos[(\eta+1)\pi] \\ \frac{\xi+1}{2} R\sin[(\eta+1)\pi] \end{bmatrix}$$

where $-1 \le \xi \le +1$ and $-1 \le \eta \le +1$, and $R > 0$. This map takes a square in the ξ, η plane (the standard square, we are going to see more of it soon) to a circle in the x, y plane. Compute the tangent vectors to the coordinate curves.

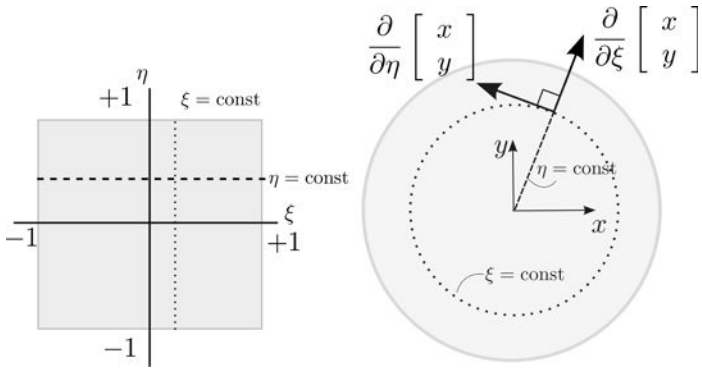
Solution: The map is readily differentiated with respect to ξ as

$$\frac{\partial}{\partial \xi}\begin{bmatrix} x \\ y \end{bmatrix} = \frac{\partial}{\partial \xi}\begin{bmatrix} \frac{\xi+1}{2} R\cos[(\eta+1)\pi] \\ \frac{\xi+1}{2} R\sin[(\eta+1)\pi] \end{bmatrix} = \begin{bmatrix} \frac{R}{2}\cos[(\eta+1)\pi] \\ \frac{R}{2}\sin[(\eta+1)\pi] \end{bmatrix}$$

and with respect to η as

$$\frac{\partial}{\partial \eta}\begin{bmatrix} x \\ y \end{bmatrix} = \frac{\partial}{\partial \eta}\begin{bmatrix} \frac{\xi+1}{2} R\cos[(\eta+1)\pi] \\ \frac{\xi+1}{2} R\sin[(\eta+1)\pi] \end{bmatrix} = \begin{bmatrix} -\frac{\xi+1}{2} R\pi\sin[(\eta+1)\pi] \\ \frac{\xi+1}{2} R\pi\cos[(\eta+1)\pi] \end{bmatrix}$$

As illustrated in the figure below, the two vectors are tangents to curves $\eta = \text{const}$ and $\xi = \text{const}$.



Let us now look at what the map (8.58) does with areas. The parallelogram (rectangle) generated by the vectors $[\mathrm{d}\xi, 0]^T$ and $[0, \mathrm{d}\eta]^T$ (given in components in the Cartesian coordinate system ξ, η), has the area of ($\times$ is the cross product symbol)

$$\begin{bmatrix} \mathrm{d}\xi \\ 0 \end{bmatrix} \times \begin{bmatrix} 0 \\ \mathrm{d}\eta \end{bmatrix} = \mathrm{d}\xi\mathrm{d}\eta \ .$$

Remember, we are talking two dimensions: the cross product is a scalar.

The two vectors $[\mathrm{d}\xi, 0]^T$ and $[0, \mathrm{d}\eta]^T$ are mapped by the map (8.58) to vectors

$$\begin{bmatrix} \mathrm{d}\xi \\ 0 \end{bmatrix} = \mathrm{d}\xi \begin{bmatrix} 1 \\ 0 \end{bmatrix} \longrightarrow \mathrm{d}\xi \begin{bmatrix} \frac{\partial x}{\partial \xi} \\ \frac{\partial y}{\partial \xi} \end{bmatrix} \ , \quad \begin{bmatrix} 0 \\ \mathrm{d}\eta \end{bmatrix} = \mathrm{d}\eta \begin{bmatrix} 0 \\ 1 \end{bmatrix} \longrightarrow \mathrm{d}\eta \begin{bmatrix} \frac{\partial x}{\partial \eta} \\ \frac{\partial y}{\partial \eta} \end{bmatrix} \ , \tag{8.59}$$

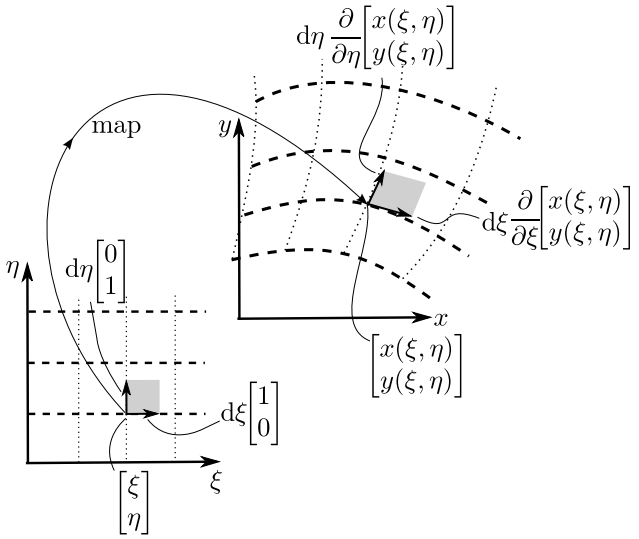


Fig. 8.19. Mapping of areas for a general map between coordinates

where the square brackets hold components in the standard Cartesian basis. Note that these vectors are ***tangent*** to the coordinate curves, which consist of the points in the physical space x, y that are maps of the curves $\xi = \text{const}$ and $\eta = \text{const}$. The area of the hatched parallelogram in Fig. 8.19 is

$$\mathrm{d}\xi \begin{bmatrix} \dfrac{\partial x}{\partial \xi} \\ \dfrac{\partial y}{\partial \xi} \end{bmatrix} \times \mathrm{d}\eta \begin{bmatrix} \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \mathrm{d}\xi\mathrm{d}\eta \begin{bmatrix} \dfrac{\partial x}{\partial \xi} \\ \dfrac{\partial y}{\partial \xi} \end{bmatrix} \times \begin{bmatrix} \dfrac{\partial x}{\partial \eta} \\ \dfrac{\partial y}{\partial \eta} \end{bmatrix} \ . \tag{8.60}$$

Compare this equation with (8.49): the two vectors in the cross product are the columns of the Jacobian matrix from (8.49). In fact, the cross product of the columns is the determinant of the Jacobian matrix (or, as the determinant is known, the Jacobian). Therefore, the map (8.58) maps areas as

$$\mathrm{d}\xi\mathrm{d}\eta \longrightarrow \mathrm{d}\xi\mathrm{d}\eta \det\left[J\right] \ . \tag{8.61}$$

As a consequence of (8.61), we have the following change of coordinates in integrals:

$$\int_{S_{[x,y]}} f(x,y)\mathrm{d}x\mathrm{d}y = \int_{S_{[\xi,\eta]}} f(\xi,\eta) \det\left[J(\xi,\eta)\right] \mathrm{d}\xi\mathrm{d}\eta \ . \tag{8.62}$$

Numerical quadrature rules take advantage of the relative ease with which these rules may be formulated on standard shapes, triangles, squares, cubes, etc. Thus, the integral on the left of (8.62) will be approximated as

$$\int_{S_{[x,y]}} f(x,y)\mathrm{d}x\mathrm{d}y \approx \sum_{k=1}^{M} f(\xi_k,\eta_k) \det\left[J(\xi_k,\eta_k)\right] W_k \ . \tag{8.63}$$

In the `FAESOR` toolbox, the surface Jacobian $\det\left[J(\xi,\eta)\right]$ is computed for two-dimensional manifold geometric cells by the method `Jacobian_surface`; it is discussed in Section 11.4.

We will introduce three integration rules for the standard triangle, one-point, three-point, and six-point quadrature, but many other rules are available: a number of authors have compiled tables, see for instance Hughes' book [H00]. The 1-point rule will be able to integrate linear polynomials in ξ, η exactly, and the 3-point does the job for up to quadratic polynomials in ξ, η. The six-point rule is good for fourth order polynomials, which may seem an overkill for applications with linear

Table 8.1. Integration rules on the standard triangle; $a = 0.816847572980459$, $b = 0.091576213509771$, $c = 0.108103018168070$, $d = 0.445948490915965$

Rule	Coordinates ξ_j, η_j	Weights W_j	Integrates exactly
1-point	1/3, 1/3	1/2	linear polynomial
3-point	2/3, 1/6	1/6	quadratic polyn.
	1/6, 2/3	1/6	
	1/6, 1/6	1/6	
6-point	a, b	0.054975871827661	quartic polyn.
	b, a	0.054975871827661	
	b, b	0.054975871827661	
	c, d	0.111690794839006	
	d, c	0.111690794839006	
	c, c	0.111690794839006	

triangles, but its worth will be appreciated later. Table 8.1 gives the coordinates of the integration points, and their weights.

Exercise 35. Consider the triangle $\triangle KLM$ with nodes at (x_K, y_K), (x_L, y_L) , and (x_M, y_M). Compute the Jacobian matrix of the map (8.26).

Solution: We will use the formula (8.50). The basis functions are (8.23), (8.21), and (8.22). Thus we have the components of the Jacobian matrix

$$J_{11} = \sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} x_i = (-1)x_K + (+1)x_L + (0)x_M = (x_L - x_K)$$
$$J_{12} = \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} x_i = (-1)x_K + (0)x_L + (+1)x_M = (x_M - x_K)$$
$$J_{21} = \sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} y_i = (-1)y_K + (+1)y_L + (0)y_M = (y_L - y_K)$$
$$J_{22} = \sum_{i=1}^{3} \frac{\partial N_i}{\partial \eta} y_i = (-1)y_K + (0)y_L + (+1)y_M = (y_M - y_K)$$

yielding

$$[J] = \begin{bmatrix} (x_L - x_K) \ , (x_M - x_K) \\ (y_L - y_K) \ , \ (y_M - y_K) \end{bmatrix} .$$

The same result is obtained with the matrix multiplication (8.54), where we set

$$[\mathtt{x}] = \begin{bmatrix} x_K \ , \ y_K \\ x_L \ , \ y_L \\ x_M \ , \ y_M \end{bmatrix} ,$$

and

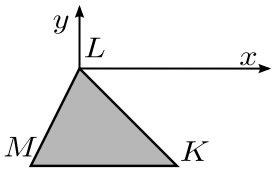
$$[\mathtt{Nder}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} \ , \ \dfrac{\partial N_1}{\partial \eta} \\ \dfrac{\partial N_2}{\partial \xi} \ , \ \dfrac{\partial N_2}{\partial \eta} \\ \dfrac{\partial N_3}{\partial \xi} \ , \ \dfrac{\partial N_3}{\partial \eta} \end{bmatrix} = \begin{bmatrix} -1 \ , -1 \\ 1 \ , \ 0 \\ 0 \ , \ 1 \end{bmatrix} .$$

Note that the Jacobian matrix is a square matrix of the map (8.26). It consists of the two vectors $\boldsymbol{v}$ and $\boldsymbol{w}$. Compare with Exercise 33: It is worthwhile to note the pattern: the tangent vectors are columns of the Jacobian matrix!

Also note that for the triangle T3 the Jacobian matrix is constant (it does not depend on the location of the evaluation point). Furthermore, the (constant) Jacobian is equal to twice the area of the triangle.

Exercise 36.

Compute the moments of inertia of the triangle $\triangle KLM$ with nodes at $(x_K, y_K) = (2, -2)$, $(x_L, y_L) = (0, 0)$, and $(x_M, y_M) = (-1, -2)$ using numerical quadrature with the rules of Table 8.1.



Solution: The moments of inertia are defined as

$$I_{xx} = \int_S y^2 \mathrm{d}A \ , \quad I_{xy} = \int_S xy \mathrm{d}A \ , \quad I_{yy} = \int_S x^2 \mathrm{d}A$$

We will approximate these integrals using numerical quadrature rules as

$$I_{xx} \approx \sum_{k=1}^{M} y^2(\xi_k, \eta_k) \det\left[J(\xi_k, \eta_k)\right] W_k \ ,$$

and so on for the other quantities. The Jacobian matrix may be constructed as in Exercise 35

$$[J(\xi_k, \eta_k)] = [J] = \begin{bmatrix} (x_L - x_K) \ , \ (x_M - x_K) \\ (y_L - y_K) \ , \ (y_M - y_K) \end{bmatrix} = \begin{bmatrix} -2 \ , & -3 \\ 2 \ , & 0 \end{bmatrix}$$

and the Jacobian is the cross product of the columns of the Jacobian matrix, a.k.a. the determinant of the 2 x 2 matrix

$$\det\left[J(\xi_k, \eta_k)\right] = \det\left[J\right] = \begin{bmatrix} -2 \\ 2 \end{bmatrix} \times \begin{bmatrix} -3 \\ 0 \end{bmatrix} = +6$$

The evaluation of the coordinates $x(\xi_k, \eta_k)$ and $y(\xi_k, \eta_k)$ is the tedious part. In order to expedite these calculations we use a bit of Matlab code here. This is an anonymous function to evaluate the basis functions at particular parametric coordinates `xi,eta`

```
N =@(xi,eta) [1-xi-eta;xi;eta]
```

It can be used to evaluate the interpolation (8.25)

$$x(\xi, \eta) = \sum_{i=1}^{3} N_i(\xi, \eta) x_i \ , \quad y(\xi, \eta) = \sum_{i=1}^{3} N_i(\xi, \eta) y_i$$

as we have for instance at the centroid the values of the basis functions

```
>> N(1/3,1/3)
ans =
    0.3333
    0.3333
    0.3333
```

Defining two arrays for the locations of the nodes

$$x = [2; 0; -1]; y = [-2; 0; -2];$$

we can write the integrands as for instance `(N(1/3,1/3)'*y)^2`. Therefore, we can evaluate using the one-point rule

```
>> (N(1/3,1/3)'*y)^2*6*(1/2)
(N(1/3,1/3)'*x)*(N(1/3,1/3)'*y)*6*(1/2)
(N(1/3,1/3)'*x)^2*6*(1/2)
ans =
    5.3333
ans =
   -1.3333
ans =
    0.3333
```

Here

$$\underbrace{\texttt{(N(1/3,1/3)'*y)}}_{y(\xi,\eta)=\sum_{i=1}^{3} N_i(\xi,\eta) y_i}\texttt{\^{}2*}\underbrace{\texttt{6}}_{\det[J]}\texttt{*}\underbrace{\texttt{(1/2)}}_{W_k}$$

and so on. The moments of inertia are computed only approximately with the one-point rule. The next available step up, the three-point quadrature rules already integrates all three moments of inertia exactly:

```
>> (N(2/3,1/6)'*y)^2*6*(1/6)+(N(1/6,2/3)'*y)^2*6*(1/6)+(N(1/6,1/6)'*y)^2*6*(1/6)
(N(2/3,1/6)'*x)*(N(2/3,1/6)'*y)*6*(1/6)+...
    (N(1/6,2/3)'*x)*(N(1/6,2/3)'*y)*6*(1/6)+...
        (N(1/6,1/6)'*x)*(N(1/6,1/6)'*y)*6*(1/6)
(N(2/3,1/6)'*x)^2*6*(1/6)+(N(1/6,2/3)'*x)^2*6*(1/6)+(N(1/6,1/6)'*x)^2*6*(1/6)
ans =
     6
ans =
   -1.5000
ans =
    1.5000
```

That is guaranteed by the quadrature rule: quadratic polynomial integrands are integrated exactly by the three-point rule.

8.12 Conductivity matrix and heat loads

As discussed already in Chapter 6, all problem-dependent code is concentrated in a descendent of the `feblock` class. In particular, the two-dimensional heat diffusion model of this chapter is implemented in the `feblock_diffusion` finite element block class.

The conductivity (8.37) and other matrices are computed by evaluating the contributions from each element separately, storing these contributions element-by-element in a cell array, and then

finally assembling all the element contributions into the overall system matrix. Therefore, the conductivity matrix would be computed element-by-element. The elementwise conductivity matrix of a T3 triangle may be constructed using the mnemonic device of the single-element mesh of Section 2.12.1. As shown in Figure 8.20, the element has three nodes, A, B, and C which are associated with degrees of freedom (temperatures) 1, 2, and 3, and its conductivity matrix has components

$$K_{ji}^{(e)} = \int_{S^{(e)}} (\mathrm{grad} N_{\langle j \rangle})\, \boldsymbol{\kappa} (\mathrm{grad} N_{\langle i \rangle})^T \, \Delta z \, \mathrm{d}S \quad i,j = 1,2,3\,. \tag{8.64}$$

The $[K^{(e)}[$ matrix would be added to the global matrix as indicated in the schematic of Figure 8.20. The symbols indicate this procedure, the so-called ***assembly*** of the element matrix. This assembly process is executed in the `assemble`[4] method of the classes `dense_sysmat` and `sparse_sysmat`.

Remark: In `FAESOR` only the unknown degrees of freedom are given non-zero equation numbers on the global level. Degrees of freedom that are prescribed are not numbered, and rather are assigned an invalid "equation number" zero (0). For instance, if we assume that node A carries the unknown number 13, node C carries the unknown 61, and node B is associated with prescribed temperature, and correspondingly a zero (0) indicates that there is no equation for node B, then the row and column associated with B is ignored during the assembly.

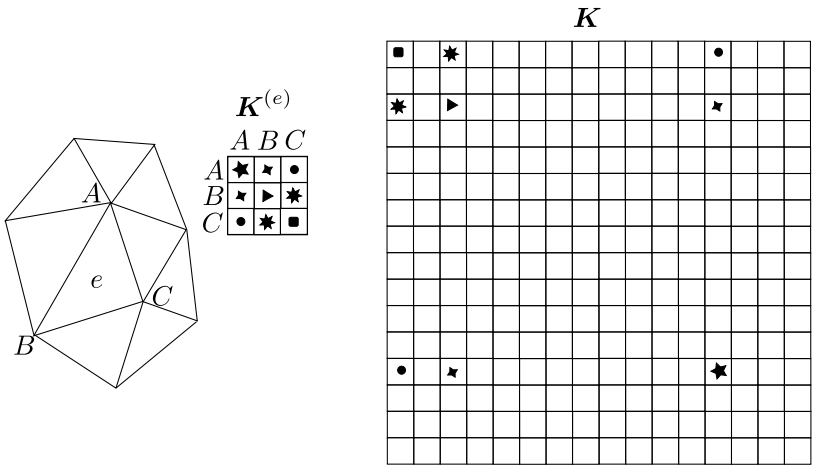


Fig. 8.20. Assembly of the element conductivity matrix

The method `conductivity` returns an object (`ems`) that represents a set of element matrices (class `elematset`). In the present case, each of the matrices represents the conductivity matrix of a single T3 element. The method begins by retrieving some information from the parent class, such as `gcells` (cell array of the geometric cells), `integration_rule`, and the material `mat`.

```
0009 function ems = conductivity[5](self, geom, temp)
0010     gcells = get(self.feblock,'gcells');
0011     nfens = get(gcells,'nfens');
0012     dim = get(geom,'dim');
0013     % Integration rule
0014     integration_rule = get(self.feblock, 'integration_rule');
0015     pc = get(integration_rule, 'param_coords');
0016     w  = get(integration_rule, 'weights');
0017     npts_per_gcell = get(integration_rule, 'npts');
```

[4]Folder: `FAESOR/classes/sysmat/@dense_sysmat`
[5]Folder: `FAESOR/classes/feblock/@feblock_diffusion`

It is of advantage to realize that all the geometric cells in the set `gcells` are of the same type, for instance here all are triangles with three nodes T3. Furthermore, the values of the basis functions (8.21–8.23) and the values of the basis function gradients with respect to the parametric coordinates (8.56) do not depend on which nodes are connected by a finite element and where those nodes are located. Therefore once we have decided on the integration rule to use to evaluate the necessary integrals, we can precompute these quantities at all integration points and then use them whenever needed.

```
0018     % Precompute basis f. values + basis f. gradients wrt parametric coor
0019     for j=1:npts_per_gcell
0020         Ns{j} = bfun(gcells,pc(j,:));
0021         Nders{j} = bfundpar(gcells,pc(j,:));
0022     end
```

Next we retrieve the conductivity matrix of the material

```
0023     % Material
0024     mat = get(self.feblock, 'mater');
0025     kappa = get(get(mat,'property'),'conductivity');
```

and also the array of connectivities . Finally we prepare for the calculation of the element conductivity matrices by pre-allocating the space for the matrices and for the equation numbers that go with them.

```
0026     % Prepare some data: connectivity and the element matrices
0027     conns = get(gcells, 'conn'); % connectivity
0028     Ke = cell(size(conns,1),1);
0029     eqnums = cell(size(conns,1),1);
0030     for i=1:size(conns,1)
0031         eqnums{i} =gather(temp,conns(i,:),'eqnums');
0032         Ke{i} =zeros(nfens);
0033     end
```

We must not forget the locations of the nodes which will be needed to compute the Jacobians and gradients of the basis functions with respect to the space coordinates.

```
0034     xs =get(geom,'values');% retrieve the geometry information
```

The loop over all the geometric cells starts with the retrieval of the connectivity (i.e. the numbers of the nodes which are connected together by the cell), and of the array of the node coordinates, `x` (compare with (8.55)). Then, the loop over all the quadrature points may begin.

```
0035     for i=1:size(conns,1)
0036         conn =conns(i,:);
0037         x=xs(conn,:);
0038         for j=1:npts_per_gcell
0039             J = Jacobian_matrix(gcells,Nders{j},x);
0040             Jac = Jacobian_volume(gcells,conn, Ns{j}, J, x);
0041             Ndersp = Nders{j}/J;
```

The loop over the integration points begins with the computation of the Jacobian matrix, the Jacobian, and the spatial derivatives of the basis functions, `Nspder`. Note that the method used, `Jacobian_volume`, computes a *volume* Jacobian: even though the method works with representation of temperatures as functions of two variables, x, y, the problem that is being solved is still the heat conduction through a three-dimensional solid.

Often for orthotropic materials the axes of orthotropy vary from point-to-point. In that case it makes sense to describe the material properties in local Cartesian coordinates, and then allow the finite element block to define a transformation matrix between the local coordinate directions and

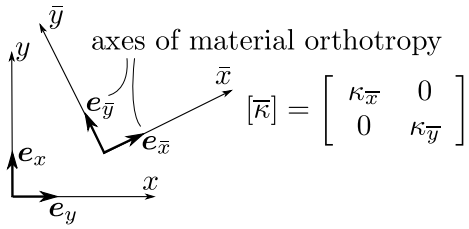


Fig. 8.21. Directions of material orthotropy

the global Cartesian basis: refer to Fig. 8.21. The attribute of the material property object is thus the material conductivity in the *local basis*, $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$

$$[\overline{\kappa}] = \begin{bmatrix} \kappa_{\overline{x}} & 0 \\ 0 & \kappa_{\overline{y}} \end{bmatrix} , \tag{8.65}$$

which is rotated into the global Cartesian basis using the transformation matrix (rotation matrix)

$$[R_m] = \left[[\boldsymbol{e}_{\overline{x}}] \; [\boldsymbol{e}_{\overline{y}}] \right] . \tag{8.66}$$

The columns of $[R_m]$ are the components of the basis vectors $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$ in the global Cartesian coordinates. The material conductivity matrix in the global basis is then expressed using the ordinary transformation rule

$$[\kappa] = [R_m][\overline{\kappa}][R_m]^T . \tag{8.67}$$

The finite element block computes the local material directions using either a user-supplied orientation matrix, or, for isotropic materials, the default identity matrix.

```
0042              Rm = material_directions(self,gcells,pc(j,:),x);
```

Now we exercise the integration rule. Note that the contribution to the element conductivity matrix is computed as a square matrix, since the gradients of the basis functions are arranged as rows of the `Nspder` matrix.

```
0043              Ke{i} = Ke{i} + Ndersp*Rm*(kappa*Jac*w(j))*Rm'*Ndersp' ;
```

Finally, the computed element conductivity matrices `Ke` and the equation numbers for each element `eqnums` are used to create the `ems` object of class `elematset`.

```
0044          end% Loop over quadrature points
0045      end% Loop over elements
0046      ems  = elematset(struct('mat',{Ke}, 'eqnums',{eqnums}));
0047 end
```

Since the topic of the Jacobians has been brought up, we point out how the volume Jacobian is computed for the triangle element T3 in the method `Jacobian_volume`[6]. We ignore everything that does not pertain to the present case, and we focus on line `0024`: the volume Jacobian is computed as the product of the surface Jacobian (determinant of (8.49)) and the "other dimension" (thickness of the slice).

```
0019 function Jac = Jacobian_volume[7](self, conn, N, J, x)
0020      if self.axisymm
0021          xyz =N'*x;
0022          Jac=Jacobian_surface(self,conn,N,J,x)*2*pi*xyz(1);
0023      else
```

[6] Folder: `FAESOR/classes/gcellset/@gcellset_2_manifold`
[7] Folder: `FAESOR/classes/gcellset/@gcellset_2_manifold`

```
0024          Jac=Jacobian_surface(self,conn,N,J,x)*other_dimension(self,conn,N,x);
0025      end
0026  end
```

Exercise 37. Compute the elementwise conductivity matrix of the triangular finite element $\triangle KLM$ from Exercise 36. Assume isotropic homogeneous material.

Solution: The Jacobian matrix of the finite element was computed in Exercise 36

$$[J] = \begin{bmatrix} (x_L - x_K) \ , \ (x_M - x_K) \\ (y_L - y_K) \ , \ (y_M - y_K) \end{bmatrix} = \begin{bmatrix} -2 \ , \ -3 \\ 2 \ , \ 0 \end{bmatrix}$$

Therefore the gradients of the basis functions with respect to x, y are obtained from the formula (8.53) as

$$\left[\mathrm{grad}_{(x,y)} N\right] = \begin{bmatrix} \mathrm{grad} N_K \\ \mathrm{grad} N_L \\ \mathrm{grad} N_M \end{bmatrix} = \begin{bmatrix} -1 \ , \ -1 \\ 1 \ , \ 0 \\ 0 \ , \ 1 \end{bmatrix} \begin{bmatrix} -2 \ , \ -3 \\ 2 \ , \ 0 \end{bmatrix}^{-1} = \begin{bmatrix} 0.33333, & -0.16667 \\ 0, & 0.5 \\ -0.33333, & -0.33333 \end{bmatrix}$$

The elementwise conductivity matrix (8.64) may be computed component by component. For instance,

$$K_{12}^{(e)} = \int_{S^{(e)}} (\mathrm{grad} N_{\langle 1 \rangle}) \, \boldsymbol{\kappa} (\mathrm{grad} N_{\langle 2 \rangle})^T \, \Delta z \, \mathrm{d}S = \int_{S^{(e)}} (\mathrm{grad} N_K) \, \boldsymbol{\kappa} (\mathrm{grad} N_L)^T \, \Delta z \, \mathrm{d}S$$

We can see that all quantities in the integral are in fact independent of x, y and therefore the integral collapses to

$$K_{12}^{(e)} = (\mathrm{grad} N_K) \, \boldsymbol{\kappa} (\mathrm{grad} N_L)^T \, \Delta z \, S^{(e)}$$

with the area of the element $S^{(e)} = \det[J]/2 = 3$, and because the material is isotropic this further simplifies to

$$K_{12}^{(e)} = (\mathrm{grad} N_K) (\mathrm{grad} N_L)^T \, \kappa \Delta z \, S^{(e)}$$

Substituting the numbers from the gradients we get

$$K_{12}^{(e)} = \begin{bmatrix} 0.33333, & -0.16667 \end{bmatrix} \begin{bmatrix} 0, \\ 0.5 \end{bmatrix} \kappa \Delta z \, S^{(e)} = -0.0833 \, \kappa \Delta z \, S^{(e)}$$

A more efficient operation could be devised: we could compute the entire elementwise matrix at the same time. Given the matrix of the gradients of the basis functions, we can write

$$[K^{(e)}] = \int_{S^{(e)}} \begin{bmatrix} \mathrm{grad} N_K \\ \mathrm{grad} N_L \\ \mathrm{grad} N_M \end{bmatrix} \boldsymbol{\kappa} \begin{bmatrix} \mathrm{grad} N_K \\ \mathrm{grad} N_L \\ \mathrm{grad} N_M \end{bmatrix}^T \Delta z \, \mathrm{d}S \tag{8.68}$$

where in this case the integrand is constant and therefore

$$[K^{(e)}] = \begin{bmatrix} \mathrm{grad} N_K \\ \mathrm{grad} N_L \\ \mathrm{grad} N_M \end{bmatrix} \begin{bmatrix} \mathrm{grad} N_K \\ \mathrm{grad} N_L \\ \mathrm{grad} N_M \end{bmatrix}^T \kappa \Delta z \, S^{(e)}$$

In numbers

$$[K^{(e)}] = \begin{bmatrix} 0.33333, -0.16667 \\ 0, \quad 0.5 \\ -0.33333, -0.33333 \end{bmatrix} \begin{bmatrix} 0.33333, \ -0, -0.33333 \\ -0.16667, 0.5, -0.33333 \end{bmatrix} \kappa\Delta z \, S^{(e)}$$

$$= \begin{bmatrix} 0.13889, -0.083333, -0.055556 \\ -0.083333, \quad 0.25, \ -0.16667 \\ -0.055556, \ -0.16667, \quad 0.22222 \end{bmatrix} \kappa\Delta z \, S^{(e)}$$

Exercise 38.

Discuss the rank and the eigenvalues and eigenvectors of the elementwise conductivity matrix from Exercise 37.

Solution: We can write the eigenvalue/eigenvector problem for the elementwise conductivity matrix as

$$[K^{(e)}][T^{(e)}] = \lambda[T^{(e)}]$$

where λ is the eigenvalue, and $[T^{(e)}]$ is the eigenvector. Since all the parameters in the product $\kappa\Delta z \, S^{(e)}$ are positive, we may write

$$[K^{(e)}][T^{(e)}] = \begin{bmatrix} 0.13889, -0.083333, -0.055556 \\ -0.083333, \quad 0.25, \ -0.16667 \\ -0.055556, \ -0.16667, \quad 0.22222 \end{bmatrix} \kappa\Delta z \, S^{(e)} [T^{(e)}] = \lambda[T^{(e)}]$$

and

$$\begin{bmatrix} 0.13889, -0.083333, -0.055556 \\ -0.083333, \quad 0.25, \ -0.16667 \\ -0.055556, \ -0.16667, \quad 0.22222 \end{bmatrix} [T^{(e)}] = \frac{\lambda}{\kappa\Delta z \, S^{(e)}} [T^{(e)}] = \widetilde{\lambda}[T^{(e)}]$$

where we define

$$\widetilde{\lambda} = \frac{\lambda}{\kappa\Delta z \, S^{(e)}}$$

Hence we need to look at the numerical eigenvalues and eigenvectors of the numerical matrix:

```
>> [V,D] =eig([0.138888888888889  -0.083333333333333  -0.055555555555556
  -0.083333333333333   0.250000000000000  -0.166666666666667
  -0.055555555555556  -0.166666666666667   0.222222222222222])
V =
   0.577350269189625   0.810498888215186   0.098783697382797
   0.577350269189626  -0.319700252694335  -0.751304475624793
   0.577350269189627  -0.490798635520848   0.652520778241995
D =
  -0.000000000000000                   0                   0
                   0   0.205401353459334                   0
                   0                   0   0.405709757651778
```

The diagonal of `D` holds the eigenvalues. This means the elementwise conductivity matrix has a zero eigenvalue (`D(1,1)`). The other two eigenvalues are nonzero, from which we can conclude that the rank of the elementwise conductivity matrix is two (the number of nonzero eigenvalues), one less than the number of rows (columns). Consequently, it is singular. The eigenvector associated to the zero eigenvalue is uniform: 0.57735 at each node. This tallies with our intuitive understanding: temperature being the same at all nodes means its gradient is going to be zero everywhere. Therefore

the conductivity matrix does not react to this temperature vector, as it expresses the flow of heat energy by conduction.

The presence of a zero eigenvalue (i.e. the singularity of the element wise conductivity matrix) can be explained further by reference to the meaning of the product $[K^{(e)}][T^{(e)}]$. This is the product of the conductivity matrix with the vector of temperatures at the nodes.

We refer back to the definition of the trial function for the temperature (8.30), and we write it only for a single-element mesh

$$T(x,y) = \sum_{i=1}^{3} N_i(x,y) T_{(i)}$$

Using the definition of the trial function we can compute the gradient of the temperature as

$$\mathrm{grad}T(x,y) = \left[\frac{\partial T(x,y)}{\partial x} \, , \, \frac{\partial T(x,y)}{\partial y} \right] = \left[\frac{\partial \sum_{i=1}^{3} N_i(x,y) T_{(i)}}{\partial x} \, , \, \frac{\partial \sum_{i=1}^{3} N_i(x,y) T_{(i)}}{\partial y} \right]$$

which can be rewritten by realizing that $T_{(i)}$ are not subject to differentiation and by taking the differentiation into the sum

$$\mathrm{grad}T(x,y) = \sum_{i=1}^{3} \left[\frac{\partial N_i(x,y)}{\partial x} \, , \, \frac{\partial N_i(x,y)}{\partial y} \right] T_{(i)}$$

At this point we realize that each of the bracket holds a basis function gradient (see (8.29))

$$\mathrm{grad}N_i(x,y) = \left[\frac{\partial N_i(x,y)}{\partial x} \, , \, \frac{\partial N_i(x,y)}{\partial y} \right]$$

so that we can finally write

$$\mathrm{grad}T(x,y) = \sum_{i=1}^{3} \mathrm{grad}N_i(x,y) T_{(i)} = \sum_{i=1}^{3} T_{(i)} \mathrm{grad}N_i(x,y)$$

This can be also written as a handy matrix expression

$$\mathrm{grad}T(x,y) = \left[T_{(1)}, T_{(2)}, T_{(3)} \right] \begin{bmatrix} [\mathrm{grad}N_1(x,y)] \\ [\mathrm{grad}N_2(x,y)] \\ [\mathrm{grad}N_3(x,y)] \end{bmatrix}$$

or, even more succinctly

$$\mathrm{grad}T(x,y) = [T^{(e)}]^T \left[\mathrm{grad}N^{(e)}(x,y) \right]$$

where $[T^{(e)}]$ is the vector of the three nodal temperatures, and $\left[\mathrm{grad}N^{(e)}(x,y)\right]$ is the matrix of the gradients of the three basis functions of the triangle. Now we recall the connectivity matrix for the single-element mesh (8.68) and we write the product

$$[K^{(e)}][T^{(e)}] = \left(\int_{S^{(e)}} \begin{bmatrix} \mathrm{grad}N_K \\ \mathrm{grad}N_L \\ \mathrm{grad}N_M \end{bmatrix} \boldsymbol{\kappa} \begin{bmatrix} \mathrm{grad}N_K \\ \mathrm{grad}N_L \\ \mathrm{grad}N_M \end{bmatrix}^T \Delta z \, \mathrm{d}S \right) [T^{(e)}]$$

The nodal temperatures may be brought in to yield

$$[K^{(e)}][T^{(e)}] = \int_{S^{(e)}} \begin{bmatrix} \mathrm{grad}N_K \\ \mathrm{grad}N_L \\ \mathrm{grad}N_M \end{bmatrix} \boldsymbol{\kappa} \left(\begin{bmatrix} \mathrm{grad}N_K \\ \mathrm{grad}N_L \\ \mathrm{grad}N_M \end{bmatrix}^T [T^{(e)}] \right) \Delta z \, \mathrm{d}S$$

and we realize

$$\left(\begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix}^T [T^{(e)}] \right) = (\text{grad}T)^T$$

so that the above may be rewritten

$$[K^{(e)}][T^{(e)}] = \int_{S^{(e)}} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix} \boldsymbol{\kappa} (\text{grad}T)^T \, \Delta z \, \text{d}S$$

Now, if all the nodal temperatures are the same, $T_{(1)} = T_{(2)} = T_{(3)}$, the gradient of the temperature is identically zero, $\text{grad}T = [0, 0]$. Therefore, we have

$$[K^{(e)}][T^{(e)}] = \int_{S^{(e)}} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix} \boldsymbol{\kappa} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \, \Delta z \, \text{d}S = \int_{S^{(e)}} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \, \Delta z \, \text{d}S = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Consequently, this is equivalent to the eigenvalue problem with a zero eigenvalue

$$[K^{(e)}][T^{(e)}] = 0 \times [T^{(e)}] = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Therefore, the elementwise conductivity matrix will always have ***rank*** $(n-1)$ where n is the number of nodes of the element. The reason is that for the *uniform distribution of nodal temperatures*, which gives a *zero gradient* of the temperature, the eigenvalue problem always has a solution with a *zero eigenvalue*. Matrices with one zero eigenvalue are *singular* and their rank is consequently reduced by one with respect to the number of rows and columns.

Exercise 39.

Construct the elementswise heat load vector due to internal heat generation of the triangular finite element $\triangle KLM$ from Exercise 36. Assume uniform heat generation rate Q.

Solution: The component of the load vector is defined in (8.39). For Q uniform we see that the elementwise heat load vector on a single-element mnemonic mesh is

$$L_{Q,j} = \int_{S^{(e)}} N_j Q \, \Delta z \, \text{d}S = Q \Delta z \int_{S^{(e)}} N_j \, \text{d}S \; , \quad j = K, L, M \; .$$

The task therefore reduces to the integration of the basis function over the area of the triangle. We will use one-point numerical integration, but the task can be also accomplished by elementary geometry operations. We approximate the integral as

$$\int_{S^{(e)}} N_j \, \text{d}S \approx N_j(\xi_1, \eta_1) \det [J(\xi_1, \eta_1)] \, W_1 \; , \quad j = K, L, M$$

The locations of the quadrature point and the weight are given in Table 8.1 as $\xi_1 = 1/3, \eta_1 = 1/3, W_1 = 1/2$. The Jacobian is independent of the location of the quadrature point for the T3 triangle, and it was computed in Exercise 36. All the basis functions $j = K, L, M$ assume the value of $N_j = 1/3$ at the quadrature point. Therefore we have

$$\int_{S^{(e)}} N_j \, \mathrm{d}S \approx N_j(\xi_1, \eta_1) \det\left[J(\xi_1, \eta_1)\right] W_1 = (1/3) \times \det\left[J(\xi_1, \eta_1)\right] \times (1/2)$$

and considering that $\det\left[J(\xi_1, \eta_1)\right] \times (1/2) = S^{(e)}$ we can express the result as

$$L_{Q,j} = \int_{S^{(e)}} N_j Q \; \Delta z \, \mathrm{d}S = Q \Delta z S^{(e)}/3 \; , \quad j = K, L, M \; .$$

In words, each of the nodes of the triangle will get one third of the total power generated within the element. (Note that $\Delta z S^{(e)}$ is the 3-D volume of the element.)

Exercise 40. Solve for the temperature distribution in a concrete column of circular cross-section $R = 2.5$m immersed in water at 0^o C. The concrete gives off heat at the rate of 4.5 W/m^3. Ignore the foundation and the top of the column: solve with two-dimensional model. At the surface of the column assume that the temperature is that of the surrounding water. Model a 10^o pie slice. Use a single-element mesh. Take $\kappa = 1.8$W/m/^{o}K.

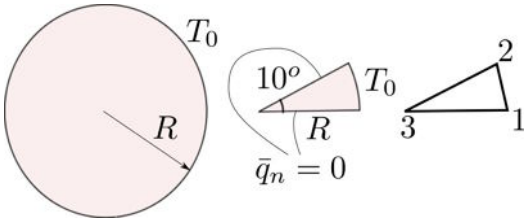


Fig. 8.22. Circular domain of the cross-section of a concrete column immersed in water

Solution: Given the circular cross-section of the column and the symmetry of the material and the boundary conditions, the temperature distribution has an infinite number of planes of symmetry: any plane passing through the center of the circle is a plane of symmetry. The temperature being symmetric with respect to a plane means that the gradient of the temperature through the plane of symmetry is identically zero. Therefore no heat flows through the plane of symmetry. This then determines the boundary condition on the straight edges of the pie slice: they are associated with zero heat flux. The circular arc boundary is associated with prescribed temperature $T_0 = 273.15^o$K. For pedagogical purposes the calculation will be carried out in absolute temperature.

The single-element mesh is constructed as shown. We take the correspondence of the nodes and degrees of freedom as

Node	1	2	3
Degree of freedom (equation) number	2	3	1

The coordinates of the nodes are

Node	x	y
1	2.5	0.0
2	2.462	0.434
3	0.0	0.0

The Jacobian matrix may be constructed easily from the tangent vectors (8.27)

$$[J] = \begin{bmatrix} 2.462 - 2.5 & 0.0 - 2.5 \\ 0.434 - 0.0 & 0.0 - 0.0 \end{bmatrix} = \begin{bmatrix} -0.038 & -2.5 \\ 0.434 & 0.0 \end{bmatrix}$$

The area of the triangle is

$$S^{(e)} = \det[J]/2 = 0.54265\text{m}^2$$

The gradients of the basis functions are computed using (8.53)

$$\left[\text{grad}_{(x,y)}N\right] = \begin{bmatrix}\text{grad}N_1\\ \text{grad}N_2\\ \text{grad}N_3\end{bmatrix} = \begin{bmatrix}-1\ , -1\\ 1\ , \quad 0\\ 0\ , \quad 1\end{bmatrix}\begin{bmatrix}-0.038 & -2.5\\ 0.434 & 0.0\end{bmatrix}^{-1} = \begin{bmatrix}0.4, & -2.2685\\ 0, & 2.3035\\ -0.4, & -0.0350\end{bmatrix}$$

The conductivity matrix is evaluated as in Exercise 37

$$[K^{(e)}] = \int_{S^{(e)}} \begin{bmatrix}\text{grad}N_1\\ \text{grad}N_2\\ \text{grad}N_3\end{bmatrix} \boldsymbol{\kappa} \begin{bmatrix}\text{grad}N_1\\ \text{grad}N_2\\ \text{grad}N_3\end{bmatrix}^T \Delta z\ \text{d}S = \begin{bmatrix}\text{grad}N_1\\ \text{grad}N_2\\ \text{grad}N_3\end{bmatrix}\begin{bmatrix}\text{grad}N_1\\ \text{grad}N_2\\ \text{grad}N_3\end{bmatrix}^T \kappa\Delta z\ S^{(e)}$$

In numbers (we may take $\Delta z = 1\text{m}$)

$$[K^{(e)}] = \begin{bmatrix}0.4, & -2.2685\\ 0, & 2.3035\\ -0.4, & -0.0350\end{bmatrix}\begin{bmatrix}0.4, & -2.2685\\ 0, & 2.3035\\ -0.4, & -0.0350\end{bmatrix}^T \kappa\Delta z\ S^{(e)}$$

$$= \begin{bmatrix}5.3089, & -5.2284, & -0.0805\\ -5.2284, & 5.3091, & -0.0807\\ -0.0805, & -0.0807, & 0.1612\end{bmatrix} \times 1.8 \times 1 \times 0.54265 = \begin{bmatrix}5.1829, & -5.1041, & -0.07874\\ -5.1041, & 5.1829, & -0.07874\\ -0.07874, & -0.07874, & 0.15748\end{bmatrix}$$

The element equation array is $[(1),(2),(3)] = [2,3,1]$, and therefore the global conductivity matrix is assembled as

$$[K] = [0.15748]$$

The element wise load vector due to internal heat generation is written as

$$[L_Q^{(e)}] = \frac{Q\Delta z\ S^{(e)}}{3}\begin{bmatrix}1\\1\\1\end{bmatrix} = 0.814\begin{bmatrix}1\\1\\1\end{bmatrix}$$

Since the prescribed temperature is different from zero, heat loads due to a central boundary conditions need to be accounted for. The degrees of freedom at the circumference are prescribed as $T_2 = T_0$ and $T_3 = T_0$. The element wise load vector due to prescribed temperatures (essential boundary conditions) is written as

$$[L_{\overline{K}}^{(e)}] = -[K^{(e)}](:,1)T_2 - [K^{(e)}](:,2)T_3 = -\begin{bmatrix}5.1829,\\ -5.1041,\\ -0.0787,\end{bmatrix} T_0 - \begin{bmatrix}-5.1041,\\ 5.1829,\\ -0.0787,\end{bmatrix} T_0 = \begin{bmatrix}-21.5018\\ -21.5018\\ 43.0036\end{bmatrix}$$

where we used the notation $(:,1)$ to mean column 1 of the matrix. The global load vector is assembled from the elementwise contributions using the element equation array as

$$[L] = [0.814 + 43.0036] = 43.8176$$

The solution follows as $T_1 = 43.8176/0.15748 = 278.3^o\text{K}$, which is $T_1 = 278.3 - 273.15 = 5.15^o\text{C}$.

While the printout of the computed temperatures at the nodes certainly provides complete information, the same content is more easily conveyed with a picture. For instance, Figure 8.23 visualizes the temperature on the computational mesh (our single element) by raising nodes by an appropriate amount above the cross-section and then interpolating them using the finite element basis functions to create a surface in three dimensions that represents the variation of temperature. In addition, the surface is colored. In this case dark color represents low-temperature.

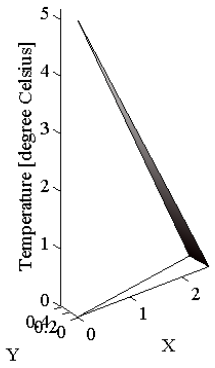


Fig. 8.23. Visualization of the computed temperature field for Exercise 40

8.13 Surface heat transfer matrix and load

In the preceding section we have been dealing with volume integrals. For instance, the conductivity matrix is evaluated over the area S_c, multiplied with Δz. This together yields the volume (compare with Figure 8.24). The surface heat transfer matrix (8.42) and the surface heat transfer load (8.41) (and also the prescribed heat flux load (8.40)) require integration over the bounding surface of the three-dimensional domain. The boundary of the cross-section consists of the curves, $C_{c,3}$ (or $C_{c,2}$), and the integral along the curve multiplying by the thickness Δz produces a surface area. As the volume integrals (which are really surface integrals multiplied by the thickness) are evaluated over the area of the triangles in the mesh, the surface integrals (which are really curved integrals multiplied by the thickness) will be computed over the edges of these triangles.

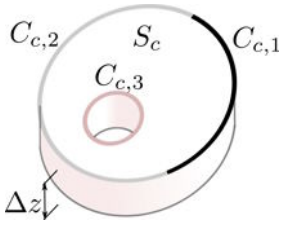


Fig. 8.24. The volume and the boundary of the three-dimensional volume, and of the two-dimensional cross-section which constitutes the computational domain

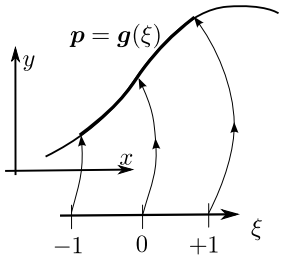


Fig. 8.25. Mapping of the standard interval to a Cartesian space

Clearly we need some understanding of integrals along curves. The goal is to evaluate

$$\int_C f(\boldsymbol{p})\,\mathrm{d}C\ , \tag{8.69}$$

where we will assume that the curve C may be "embedded" in a three-dimensional, two-dimensional, or one-dimensional Euclidean space (i.e. it may be a spatial curve, plane curve, or just an interval on the real line). Correspondingly, the point $\boldsymbol{p}$ on the curve C will have an appropriate number of components, three, two, or one.

To perform the integral, the elementary length $\mathrm{d}C$ is needed. The point $\boldsymbol{p}$ on the curve will be assumed to be the result of the mapping of the standard interval $-1 \le \xi \le +1$ (compare with the 1-D map (4.5), and refer to Fig. 8.25, where the map is two-dimensional)

$$\boldsymbol{p} = \boldsymbol{g}(\xi)\ . \tag{8.70}$$

For two closely spaced points on the curve, $\boldsymbol{p}(\xi)$ and $\boldsymbol{p}(\xi + \mathrm{d}\xi)$, where $\Delta\xi$ is the distance between the two points in the standard interval, the second point may be obtained from the first using the first two terms of the Taylor series as

$$\boldsymbol{p}(\xi + \mathrm{d}\xi) = \boldsymbol{p}(\xi) + \frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi}\mathrm{d}\xi\ \ , \qquad 0 \le \varepsilon \le 1. \tag{8.71}$$

The two points may be connected with a vector approximately tracking the curve (see Fig. 8.26),

$$\boldsymbol{p}(\xi + \mathrm{d}\xi) - \boldsymbol{p}(\xi) = \frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi}\mathrm{d}\xi\ ,$$

whose length (squared) is

$$(\mathrm{d}C)^2 = \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi}\mathrm{d}\xi\right) \cdot \left(\frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi}\mathrm{d}\xi\right) = \left|\left|\frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi}\right|\right|^2 (\mathrm{d}\xi)^2\ .$$

Skipping over the details, we may conclude that for infinitesimally short intervals

$$\frac{\partial \boldsymbol{p}(\xi + \varepsilon \mathrm{d}\xi)}{\partial \xi} \to \frac{\partial \boldsymbol{p}(\xi)}{\partial \xi}$$

and the following relationship is obtained

$$\mathrm{d}C = \left|\left|\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi}\right|\right| \mathrm{d}\xi\ , \tag{8.72}$$

where $\dfrac{\partial \boldsymbol{p}(\xi)}{\partial \xi}$ is the vector ***tangent*** to the curve at ξ, and $\left|\left|\dfrac{\partial \boldsymbol{p}(\xi)}{\partial \xi}\right|\right|$ is the Jacobian to be used in the change-of-variables operation for the curve integral. The above developments should be compared with Eq. (4.6) and the discussion preceding it: indeed, the current result is only a slight generalization.

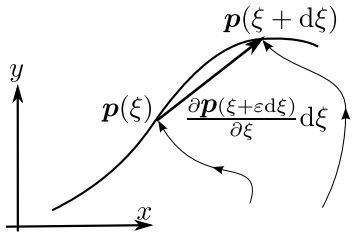


Fig. 8.26. Length of a curve

Exercise 41. Evaluate the integral $\int_C \mathrm{d}C$ along the curve generated by the map

$$\boldsymbol{p} = \boldsymbol{g}(\xi) = \sum_{i=1}^{2} N_i(\xi)\boldsymbol{x}_i \ , \quad -1 \le \xi \le +1 \ , \tag{8.73}$$

where the N_i's are given by (4.7).

Solution: The integrand is $f(\boldsymbol{p}) = 1$ so that the meaning of the integral is the length of the curve. The two endpoints are $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and because the functions (4.7) are linear, the point defined by (8.73) lies along a straight line connecting the two endpoints. In other words, the generated curve is a straight line segment.

The tangent vector is computed as

$$\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} = \sum_{i=1}^{2} \frac{N_i(\xi)}{\partial \xi}\boldsymbol{x}_i = \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{2} \ ,$$

and the Jacobian is

$$J = \left|\left| \frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} \right|\right| = \left|\left| \frac{\boldsymbol{x}_2 - \boldsymbol{x}_1}{2} \right|\right| = \frac{h}{2}$$

where $h = ||\boldsymbol{x}_2 - \boldsymbol{x}_1||$ is the length of the segment between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$.

The integral is evaluated as

$$\int_C \mathrm{d}C = \int_{-1}^{+1} J\mathrm{d}\xi = J\int_{-1}^{+1} \mathrm{d}\xi = \frac{h}{2}\int_{-1}^{+1} \mathrm{d}\xi = \frac{h}{2} \times 2 = h$$

So the length of the curve is h, as expected.

Evaluating the basis functions (8.21–8.23) along the edges of the standard triangle, we may observe that the basis function associated with the opposite vertex is identically zero. For instance the basis function N_1 is identically zero along the edge $2, 3$. The other two basis functions along the edge, in the present example those would be N_2 and N_3, vary linearly along the edge $2, 3$. Therefore, we may consider each edge of the triangle T3 to be equivalent to the finite element L2. We have formulated the L2 element as an isoparametric element in Exercise 30, except that the element domain was one-dimensional. The concept remains the same if the element nodes are in fact located in two or three dimensions. So we can write the interpolation of the geometric coordinates

$$x = \sum_{i=1}^{2} N_i(\xi)x_i \ , \quad y = \sum_{i=1}^{2} N_i(\xi)y_i \ ,$$

where node j is at location (x_j, y_j). Note that this is the map (8.73) considered an Exercise 41, so that we can retroactively recognize that in that exercise we have been working with an L2 finite element.

Interpolating in this way also the temperature as expected for an isoparametric element

$$T = \sum_{i=1}^{2} N_i(\xi)T_{\langle i \rangle} = \sum_{i=1}^{2} N_{\langle i \rangle}(\xi)T_i \ .$$

means that the temperature varies linearly between the two nodes.

Consequently, integrating an expression along the edge of the triangle T3 that connects nodes i, j yields exactly the same result as integrating along the line element L2 that connects nodes i, j. Therefore we will discretize the boundary of the two-dimensional computational domain with L2 elements, and we will evaluate the elementwise finite element for an L2 element.

Exercise 42.

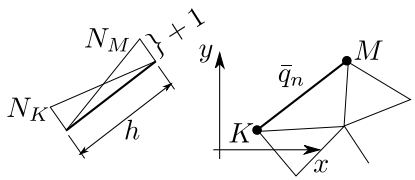


Fig. 8.27. L2 finite element of the boundary of the domain with prescribed heat flux

Compute the elementwise load vector due to prescribed heat flux on the boundary (8.40). Assume uniform prescribed heat flux $\bar{q}_n$.

Solution: The elementwise quantity will be evaluated on the element KM (see Figure 8.27). The mesh of the interior of the domain is also shown, but only for illustrative purposes, it is not used. Using the mnemonic mesh of the single L2 finite element KM the task is to evaluate

$$L_{q2,j} = -\int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \, \overline{q}_n \, \Delta z \, \mathrm{d}C \, ,$$

which may be simplified as

$$L_{q2,j} = -\overline{q}_n \, \Delta z \int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \, \mathrm{d}C \, ,$$

given that $\overline{q}_n \, \Delta z$ does not vary along the length of the element. Using the mapping from the standard interval to the physical space we can write the integral

$$\int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \, \mathrm{d}C = \int_{-1}^{+1} N_j J \mathrm{d}\xi$$

We have determined that the Jacobian $J = h/2$ in Exercise 41, and so we have for instance for $j = K$

$$\int_{-1}^{+1} N_K J \mathrm{d}\xi = (h/2) \int_{-1}^{+1} N_K \mathrm{d}\xi = (h/2) \int_{-1}^{+1} \frac{\xi - 1}{-2} \mathrm{d}\xi = (h/2) \left[\frac{\xi^2/2 - \xi}{-2} \right]_{-1}^{+1} = (h/2)$$

and analogously

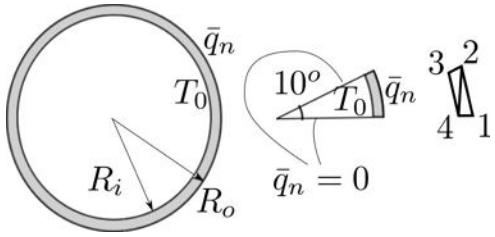
$$\int_{-1}^{+1} N_M J \mathrm{d}\xi = (h/2)$$

So that for uniform prescribed heat flux we get the elementwise heat load vector

$$[L_{q2}]^{(e)} = -\frac{\overline{q}_n \, \Delta z h}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{8.74}$$

Recall that the direction of the heat flux through the boundary is accounted for by the sign of $\overline{q}_n$: positive means heat flowing out from the domain, negative means heat flowing into the domain.

Exercise 43. Solve for the temperature distribution in a concrete pipe of circular cross-section internal radius $R_i = 0.5$m and external radius $R_o = 0.55$m. The pipe holds water at 17^o C, so that we can assume that the internal surface of the pipe is at this temperature. The concrete pipe loses heat at the external cylindrical surface at the rate of the heat flux 113 W/m^2. Ignore the ends of the pipe: solve with two-dimensional model. Model a 10^o pie slice. Use a two-element mesh. Take $\kappa = 1.8$W/m/^{o}K.



Solution: The boundary conditions on the straight edges of the pie slice are determined similarly to Exercise 40: they are associated with zero heat flux. The interior circular arc boundary is associated with prescribed temperature $T_0 = 17^o$C. The calculation will be carried out in ^{o}Celsius.

The two-element mesh is constructed as shown: element 1 with nodes $1, 2, 3$, element 2 with nodes $3, 4, 1$. We take the correspondence of the nodes and degrees of freedom as

Node	1	2	3	4
Degree of freedom (equation) number	4	2	1	3

The coordinates of the four nodes will be arranged as rows of the following matrix:

$$\mathtt{x} = \begin{bmatrix} 0.5, & 0 \\ 0.55, & 0 \\ 0.54164, & 0.095506 \\ 0.4924, & 0.086824 \end{bmatrix}$$

In this exercise we use equation (8.54) to compute the Jacobian matrix. For element 1 we get

$$[J] = \begin{bmatrix} 0.5, & 0.55, & 0.54164 \\ 0, & 0, & 0.095506 \end{bmatrix} \begin{bmatrix} -1, & -1 \\ 1, & 0 \\ 0, & 1 \end{bmatrix} = \begin{bmatrix} 0.05, & 0.041644 \\ 0, & 0.095506 \end{bmatrix}$$

The Jacobian is $\det[J] = 0.004775$. The gradients of the basis functions are computed from (8.53)

$$\left[\text{grad}_{(x,y)} N\right] = \begin{bmatrix} -1, & -1 \\ 1, & 0 \\ 0, & 1 \end{bmatrix} \begin{bmatrix} 20, & -8.7207 \\ 0, & 10.47 \end{bmatrix} = \begin{bmatrix} -20, & -1.7498 \\ 20, & -8.7207 \\ 0, & 10.47 \end{bmatrix}$$

The conductivity matrix of element 1 therefore results from (see Exercise 37)

$$[K^{(e)}] = \int_{S^{(e)}} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix} \boldsymbol{\kappa} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix}^T \Delta z \, \text{d}S = \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix} \begin{bmatrix} \text{grad}N_K \\ \text{grad}N_L \\ \text{grad}N_M \end{bmatrix}^T \kappa \Delta z \, S^{(e)}$$

as (we take arbitrarily $\Delta z = 1$m; also $S^{(e)} = \det[J]/2$)

$$[K^{(e)}] = \begin{bmatrix} 1.7323, & -1.6535, & -0.07874 \\ -1.6535, & 2.046, & -0.39243 \\ -0.07874, & -0.39243, & 0.47117 \end{bmatrix}$$

Using the element equation array for element 1 $[4, 2, 1]$ we assemble the partial result for the conductivity matrix

$$[K] = \begin{bmatrix} 0.47117, & -0.39243 \\ -0.39243, & 2.046 \end{bmatrix}$$

Similarly, for element 2 we obtain the elementwise conductivity matrix

$$[K^{(e)}] = \begin{bmatrix} 1.5748, & -1.6535, & 0.07874 \\ -1.6535, & 2.2506, & -0.59703 \\ 0.07874, & -0.59703, & 0.51829 \end{bmatrix}$$

With the element equation array for element 2 $[1, 3, 4]$ we assemble the final global conductivity matrix

$$[K] = \begin{bmatrix} 2.046, & -0.39243 \\ -0.39243, & 2.046 \end{bmatrix}$$

For the heat load on the exterior surface of the pipe we apply a surface mesh in the form of a single L2 element, with nodes 2,3. The elementwise heat load vector follows from (8.74)

$$[L_{q2}]^{(e)} = -\frac{\overline{q}_n \, \Delta zh}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -113 \times 1 \times ||\boldsymbol{x}_3 - \boldsymbol{x}_2||/2 \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2.7083647 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The element equation array for this surface element is $[2, 1]$, so that the partial heat load vector is

$$[L] = -5.416729 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Finally, we compute elementwise load vectors due to prescribed temperatures (8.38). For the T3 element 1 the temperature is prescribed at the first node as $T_0 = 17^o$C so that we get

$$[L_{\overline{K}}]^{(e)} = -\begin{bmatrix} 1.7323 \\ -1.6535 \\ -0.07874 \end{bmatrix} \times 17 = \begin{bmatrix} -29.449 \\ 28.11 \\ 1.3386 \end{bmatrix}$$

and for element 2 the temperature is prescribed at the second and third node as $T_0 = 17^o$C so that we get

$$[L_{\overline{K}}]^{(e)} = -\begin{bmatrix} -1.6535 \\ 2.2506 \\ -0.59703 \end{bmatrix} \times 17 - \begin{bmatrix} 0.07874 \\ -0.59703 \\ 0.51829 \end{bmatrix} \times 17 = \begin{bmatrix} 26.772 \\ -28.11 \\ 1.3386 \end{bmatrix}$$

All the load vectors are assembled using the element equation arrays specified above as

$$[L] = \begin{bmatrix} 22.693 \\ 22.693 \end{bmatrix}$$

The temperatures (free degrees of freedom) result as

$$[T] = \begin{bmatrix} 13.72415 \\ 13.72415 \end{bmatrix}$$

These are the temperatures at the nodes 2, 3 on the exterior surface of the pipe. Figure 8.28 shows the computed temperatures using a three-dimensional surface.

Exercise 44.

Compute the elementwise ambient-temperature surface heat transfer load vector (8.41) for an L2 finite element with nodes KM. Assume that neither the surface heat transfer coefficient, nor the ambient temperature vary along the boundary.

Solution: The mesh is shown in Figure 8.27. The boundary $C_{c,3}$ across which the Newton's condition operates for the mnemonic mesh is the L2 element KM. The goal is to compute the load vector

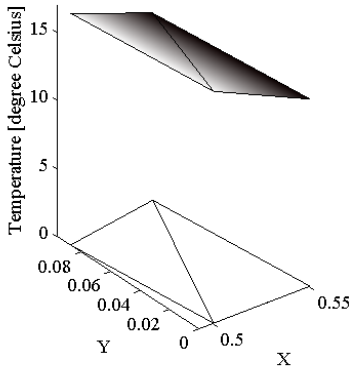


Fig. 8.28. Visualization of the computed temperature field for Exercise 43

$$L_{q3,j} = \int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \; hT_a \; \Delta z \; \mathrm{d}C \; , \quad j = K, M \; .$$

As all h, Δz and T_a are constant with respect to the integration, the integral reduces to

$$L_{q3,j} = hT_a \; \Delta z \int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \; \mathrm{d}C \; , \quad j = K, M \; .$$

We have seen this integral before in Exercise 42: The integral of the basis function of the L2 element over its length evaluates to

$$\int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_j \; \mathrm{d}C = (h^{(e)}/2) \; ,$$

where $h^{(e)} = ||\boldsymbol{x}_M - \boldsymbol{x}_K||$ is the length of the element. The elementwise ambient-temperature load vector is therefore written as

$$[L_{q3}]^{(e)} = \frac{hT_a \Delta z h^{(e)}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{8.75}$$

Exercise 45.

Compute the elementwise surface heat transfer matrix (8.42) for an L2 finite element with nodes KM. Assume that the surface heat transfer coefficient does not vary along the boundary.

Solution: The mesh is shown in Figure 8.27. The boundary $C_{c,3}$ across which the Newton's condition operates for the mnemonic mesh is the L2 element KM. The goal is to compute the matrix

$$H_{ji} = \int_{\boldsymbol{x}_K}^{\boldsymbol{x}_M} N_i \; hN_j \; \Delta z \; \mathrm{d}C \; , \quad i, j = K, M \; ,$$

We have seen an integral like this before: recall the computation of the elementwise mass matrix in Section 3.5. We just identify $h\Delta z$ with the parameter μ in the mass matrix expression. Since h and Δz are constant with respect to the integration, we can immediately write down the elementwise surface heat transfer matrix for uniform surface heat transfer coefficient as

$$[H]^{(e)} = \frac{h\Delta z h^{(e)}}{6} \begin{bmatrix} 2, & 1 \\ 1, & 2 \end{bmatrix} \tag{8.76}$$

where $h^{(e)} = ||\boldsymbol{x}_M - \boldsymbol{x}_K||$ is the length of the element.

Exercise 46.

Consider again the concrete pipe from Exercise 43 with the difference is that instead of the prescribed heat flux a Newton's boundary condition is applied on the outer cylindrical surface. The ambient temperature is -10^o C, and the surface heat transfer coefficient is $h_o = 5.2\text{W/m}^2/^o\text{K}$.

Solution: The conductivity matrix and the load due to prescribed temperatures were computed in Exercise 43. Therefore it remains to compute the heat load due to the ambient temperature, and the surface heat transfer matrix.

The heat load due to ambient temperature is computed for the L2 finite element that connects the nodes 2 and 3:

$$[L_{q3}]^{(e)} = \frac{hT_a \Delta z h^{(e)}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = -2.4927 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

The elementwise load vector is assembled using the element equation array $[2, 1]$ to give the global load vector as

$$[L] = \begin{bmatrix} 22.693 \\ 22.693 \end{bmatrix}$$

The surface heat transfer matrix is evaluated from (8.76)

$$[H]^{(e)} = \frac{h \Delta z h^{(e)}}{6} \begin{bmatrix} 2, & 1 \\ 1, & 2 \end{bmatrix} = 0.2493 \begin{bmatrix} 2, & 1 \\ 1, & 2 \end{bmatrix}$$

The global surface heat transfer matrix is assembled from the single elementwise contribution as

$$[H] = 0.2493 \begin{bmatrix} 2, & 1 \\ 1, & 2 \end{bmatrix}$$

The temperatures (free degrees of freedom) result as

$$[T] = ([K] + [H])^{-1} [L] = \begin{bmatrix} 13.4630 \\ 13.4630 \end{bmatrix}$$

These are the temperatures at the nodes 2, 3 on the exterior surface of the pipe.

We conclude this section with programming aspects that concern the surface integration. First a few words about the computation of the tangent vectors. Soon we will encounter one-dimensional elements with more than two nodes. For instance, the element L3 has three nodes. For general elements with n nodes, the computer implementation computes the tangent as

$$\frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} = \texttt{x'*Nder} \ , \tag{8.77}$$

using the following two matrices ,

$$[\texttt{x}] = \begin{bmatrix} x_1 \ , \ y_1 \\ x_2 \ , \ y_2 \\ \ldots, \ \ldots \\ x_n \ , \ y_n \end{bmatrix} \ , \tag{8.78}$$

where the number of columns is equal to the number of spatial dimensions, 1, 2 (which is assumed in (8.78)), or 3, and [Nder] collects in each row the gradient of the basis function with respect to the parametric coordinate

$$[\texttt{Nder}] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} \\ \dfrac{\partial N_2}{\partial \xi} \\ \cdots \\ \dfrac{\partial N_n}{\partial \xi} \end{bmatrix}, \tag{8.79}$$

These matrices should be compared with those defined for the triangle T3, Eqs. (8.55) and (8.56). The only difference is the number of space dimensions, the number of basis functions, and the number of parametric dimensions; all of these are taken into account by the Matlab code automatically. This computation is the same for one-, two-, and three-dimensional manifold geometric cell, and therefore it is implemented in the method Jacobian_matrix of the class gcellset: Compare formula (8.77) with line 0014.

```
0013 function J = Jacobian_matrix[8](self, Nder, x)
0014     J = x' * Nder;% Compute the Jacobian matrix
0015 end
```

Now that we have worked out how to integrate along a curve, we may describe the computation of the surface terms. The method surface_transfer computes the heat transfer matrix (8.42) by integrating over the appropriate part of the surface. The surface needs to be discretized by compatible geometric cells: when the domain (volume) is covered by three-node triangles, the boundary (surface) is covered by two-node line segments. Most of the preparation steps are straightforward, and are therefore omitted. The loop over the geometric cells computes the heat transfer matrix for each element. The Jacobian is computed in a way that is appropriate for the geometric cells (yet another example of dynamic method dispatch): the method Jacobian_surface.

```
0009 function ems = surface_transfer[9](self, geom, temp)
...
0034     for i=1:size(conns,1)
0035         conn =conns(i,:);
0036         x=xs(conn,:);
0037         for j=1:npts_per_gcell
0038             J = Jacobian_matrix(gcells,Nders{j},x);
0039             Jac = Jacobian_surface(gcells,conn, Ns{j}, J, x);
0040             Ke{i} = Ke{i} + ((h*Jac*w(j))*Ns{j})*Ns{j}';
0041         end
0042     end
...
```

The surface Jacobian is computed for the L2 (line) element following the same principle as before for the volume Jacobian for the element T3. Focusing just on this case, we see that the surface Jacobian is computed as the product of the curve Jacobian and the "other dimension" (thickness Δz)– see line 0024.

```
0019 function Jac = Jacobian_surface[10](self, conn, N, J, x)
0020     if self.axisymm
```

[8] Folder: FAESOR/classes/gcellset/@gcellset
[9] Folder: FAESOR/classes/feblock/@feblock_diffusion
[10] Folder: FAESOR/classes/gcellset/@gcellset_1_manifold

```
0021          xyz =N'*x;
0022          Jac= Jacobian_curve(self, conn, N, J, x)*2*pi*xyz(1);
0023      else
0024          Jac= Jacobian_curve(self,conn,N,J,x)*other_dimension(self,conn,N,x);
0025      end
0026  end
```

Computation of the surface transfer loads could duplicate the work done in the previous method (the same kind of integral), but in order to avoid this duplication, we use the following trick: for each element we compute the element heat surface transfer matrix and multiply by the vector of ambient temperatures (whenever they are nonzero); that gives us the vector of element loads. The method `surface_transfer_loads` is therefore quite straightforward: compute the element surface heat transfer matrices (line 0010), then loop over the geometric cells and retrieve the ambient temperatures from the field `amb`. Provided this vector is nonzero, the product `He*pT` is stored in the element vector object.

```
0009 function evs = surface_transfer_loads[11](self, geom, temp, amb)
0010      ems = surface_transfer(self, geom, temp);
0011      gcells = get(self.feblock,'gcells');
0012      conns = get(gcells, 'conn'); % connectivity
0013      Fe = cell(size(conns,1),1);
0014      eqnums = cell(size(conns,1),1);
0015      for i=1:size(conns,1)
0016          eqnums{i} =gather(temp,conns(i,:),'eqnums');
0017          Fe{i} =zeros(length(eqnums{i}),1);
0018      end
0019      mat=get(ems,'mat');
0020      for i=1:size(conns,1)
0021          conn = conns(i,:); % connectivity
0022          pT = gather(amb, conn, 'prescribed_values');
0023          if norm (pT) ~= 0
0024              Fe{i} =mat{i}*pT;
0025          end
0026      end
0027      evs  = elevecset(struct('vec',{Fe}, 'eqnums',{eqnums}));
0028  end
```

[11] Folder: `FAESOR/classes/feblock/@feblock_diffusion`

9

Steady-state Heat Conduction Solutions

The ordinary differential equations that result from the discretization in space, Eqs. (8.44), lead to steady-state solutions when $\partial T_i(t)/\partial t = 0$, and $\partial \overline{T}_i(t)/\partial t = 0$. The latter condition is necessary, while the former follows when all the transients in the solution decay (in infinite time, in general).

9.1 Steady-state heat conduction equation

Substituting the vanishing temperature rates into (8.44), we obtain

$$\sum_{i=1}^{N_\mathrm{f}} K_{ji} T_i + \sum_{i=1}^{N_\mathrm{f}} H_{ji} T_i = L_{\overline{K},j} + L_{\overline{H},j} + L_{Q,j} + L_{q2,j} + L_{q3,j} \qquad j = 1, ..., N_\mathrm{f} \ . \tag{9.1}$$

which is a system of linear equations for the unknown nodal temperatures. The nodal temperatures are now just numbers, not functions of time. Let us look at a few examples of steady-state heat conduction.

9.2 Thick-walled tube

The first example is a thick-walled rectangular tube, with the outside temperature being prescribed as zero, and the interior surface (perfectly) insulated. The material is isotropic. As shown in Fig. 9.1, the planes of symmetry may be used to reduce the size of the problem. Therefore, only one quarter is discretized, and perfect insulation is applied at the symmetry planes (no heat flows through the symmetry planes). There is a distributed heat source in the material (for instance, such as heat released by curing cement paste).

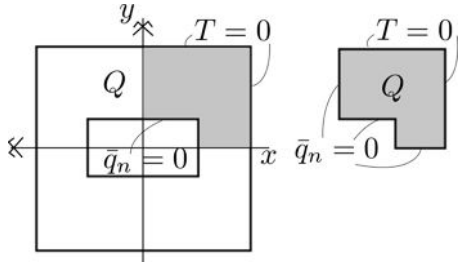


Fig. 9.1. Heat diffusion in a thick-walled rectangular tube

The Matlab script is `lshape1` [1]. The first few lines define some ancillary variables, and then the two-dimensional mesh generator of triangle meshes is invoked. The generator is thoroughly described in the user's guide `targe2_users_guide.pdf`[2], but we will say a few words about its Matlab interface. The first argument is a cell array, each element a string (character array), with one command for the mesh generator. Thus, the first six strings define the curves that bound the domain (straight-line segments), the line `0011` defines a subregion (piece of area to be covered with triangles), and the last line defines the mesh size. The second argument to `targe2_mesher` is the thickness of the slab (default value of 1.0).

```
0001 kappa=[0.2 0; 0 0.2]; % conductivity matrix
0002 Q=0.01100; % uniform heat source
0003 num_integ_pts=1; % 1-point quadrature
0004 [fens,gcells] = targe2_mesher({...
0005         ['curve 1 line 20 0 48 0'],...
0006         ['curve 2 line 48 0 48 48'],...
0007         ['curve 3 line 48 48 0 48'],...
0008         ['curve 4 line 0 48 0 13'],...
0009         ['curve 5 line 0 13 20 13'],...
0010         ['curve 6 line 20 13 20 0'],...
0011         'subregion 1 property 1 boundary 1 2 3 4 5 6',...
0012         ['m-ctl-point constant 3.5']
0013         }, 1.0);
```

Next, the property object appropriate for the heat diffusion model is created, `property_diffusion`, and supplied the material conductivity matrix $\boldsymbol{\kappa}$, and the heat source Q. The material object acts as a mediator between the property object and the finite element block. The finite element block of class `feblock_diffusion` is created, with attributes: the material, the array of geometric cells, and an integration rule (class `tri_rule` is used for triangles).

```
0014 prop=property_diffusion(
        struct('conductivity',kappa,'source',Q));
0015 mater=mater_diffusion (struct('property',prop));
0016 feb = feblock_diffusion (struct ('mater',mater,...
0017     'gcells',gcells,...
0018     'integration_rule',tri_rule(num_integ_pts)));
```

Two fields are created: `geom` represents the geometry (i.e. the locations of the nodes), and it is therefore initialized from the finite element node array, `fens`; and `theta` represents the temperatures at the nodes, and it is initially undefined, except for the number of nodes `nfens`.

```
0019 geom =field(struct('name',['geom'],'dim',2,'fens',fens));
0020 theta=field(struct('name',['theta'],'dim',1,'nfens',...
0021     get(geom,'nfens')));
```

The essential boundary conditions are next applied to the temperature field. The meshing function `fenode_select` is used to select nodes from the `fens` array based on their location: nodes which fall into given bounding boxes are selected ($[x_{lo}\ x_{hi}\ y_{lo}\ y_{hi}]$ =`[48 48 0 48]` and so on for the other box); to avoid problems with testing whether of point is inside or outside a box of zero "thickness", the boxes are for the purpose of the "in"-test inflated by 0.01. The array `prescribed` is filled with ones to indicate that all degrees of freedom are to be prescribed, the components to be prescribed are passed as an empty array (line `0026`), which simply means all components are affected. The values to which the temperatures are being prescribed are all zeros. The data defining the essential boundary conditions are set in the field (`set_ebc`), and then applied (method `apply_ebc` on line `0029`). The free node parameters are then assigned global equation numbers with the method `numbereqns`.

[1] Folder: `FAESOR/examples/heat_diffusion`
[2] Folder: `FAESOR/meshing/targe2`

```
0022 fenids=[fenode_select(fens,struct('box',[48 48 0 48],...
0023     'inflate', 0.01)),...
0024     fenode_select(fens,struct('box',[0 48 48 48],...
0025     'inflate', 0.01))];
0026 prescribed=ones(length(fenids),1);
0027 comp=[];
0028 val=zeros(length(fenids),1);
0029 theta = set_ebc(theta, fenids, prescribed, comp, val);
0030 theta = apply_ebc (theta);
0031 theta = numbereqns (theta);
```

The conductivity matrix is sparse (the linear system to be solved is going to be moderately large, and the efficiency afforded by a sparse matrix is not to be sneezed at), and it is assembled from element conductivity matrices in line 0032. The heat load vector is assembled from element load vectors, and the solution of the linear system of equations is scattered into the `theta` field.

```
0032 K = start (sparse_sysmat, get(theta, 'neqns'));
0033 K = assemble (K, conductivity(feb, geom, theta));
0034 F = start (sysvec, get(theta, 'neqns'));
0035 F = assemble (F, source_loads(feb, geom, theta));
0036 theta = scatter_sysvec(theta, get(K,'mat')\get(F,'vec'));
```

The last fragment of code takes care of the graphic presentation of the results. The field `colorfield` holds one color (a triple of floating-point numbers) per node, and those colors are obtained from the temperature field by mapping node temperatures to colors (line 0042) using the `map_data` method of the `data_colormap` class. The geometric cells of individual finite elements are plotted twice. Once as a raised colored surface (line 0046), and the second time as a wireframe in the x, y plane (line 0048). The resulting graphic is shown in Fig. 9.2. It represents the temperature as the surface raised to a certain height which corresponds to the temperature at a given point. The temperature is also encoded by color.

```
0038 gv=graphic_viewer;
0039 gv=reset (gv,[]);
0040 T=get(theta,'values');
0041 dcm=data_colormap(struct('range',[min(T),max(T)],
           'colormap',jet));
0042 colorfield=field(struct('name',['colorfield'],'data',...
0043     map_data(dcm, T)));
0044 geomT=field(struct ('name', ['geomT'], ...
0045     'data',[get(geom,'values'), get(theta,'values')]));
0046 draw(gcells, gv, struct ('x',geomT, 'u',0*geomT,...
0047         'colorfield',colorfield, 'shrink',0.9));
0048     draw(gcells, gv, struct ('x',geom, 'u',0*geom, ...
0049         'facecolor','none'));
0050 view(2)
```

9.3 Orthotropic insert

The next example introduces nonzero essential boundary conditions, and orthotropic material properties. A square block of isotropic material is insulated on the vertical edges, and two different temperatures are applied on the horizontal edges. There is a square insert of orthotropic material within the larger square. The orientation of the material axes is indicated in Fig. 9.3. Physically the insert could be made of parallel fibers (for instance carbon), embedded in a polymer matrix. The fibers conduct heat well, while in the transverse direction the polymer matrix hampers heat conduction. The problem is solved with script `squareinsquarec`[3]. The domain consists of two materials,

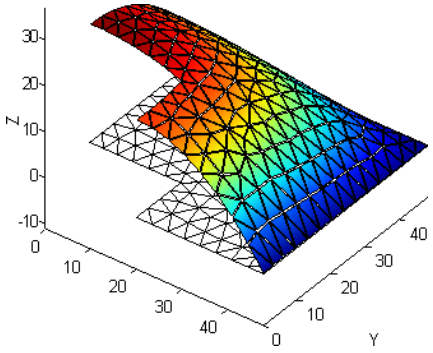


Fig. 9.2. Heat diffusion in a thick-walled rectangular tube: graphic presentation of results

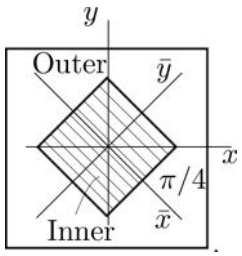


Fig. 9.3. Heat diffusion in inhomogeneous domain with orthotropic material properties

and consequently we define two material conductivity matrices: the inner material has strongly orthotropic properties; the outer material is isotropic. The rotation matrix that defines the local material properties of the insert is set up in line 0005.

```
0001 kappainner=[2.25 0; 0 0.06]; % orthotropic conduct. matrix
0002 kappaouter=[0.25 0; 0 0.25]; % isotropic conduct. matrix
0003 alpha =-45;% local material orientation angle
0004 ca=cos(2*pi/360*alpha); sa=sin(2*pi/360*alpha);
0005 Rm = [ca, -sa;sa, ca];% local material directions
```

The mesh generator defines the eight boundary segments and two subregions: note that the two subregions are assigned different numerical identifiers (1 and 2) to distinguish elements belonging to different subregions.

```
0007 [fens,gcells, groups] = targe2_mesher({...
0008         ['curve 1 line -48 -48 48 -48'],...
0009         ['curve 2 line 48 -48 48 48'],...
0010         ['curve 3 line 48 48 -48 48'],...
0011         ['curve 4 line -48 48 -48 -48'],...
0012         ['curve 5 line 0 -31 31 0'],...
0013         ['curve 6 line 31 0 0 31'],...
0014         ['curve 7 line 0 31 -31 0'],...
0015         ['curve 8 line -31 0 0 -31'],...
0016         ['subregion 1  property 1 ' ...
0017         '   boundary 1 2 3 4 -8 -7 -6 -5'],...
0018         ['subregion 2  property 2 '...
0019         '   boundary 5 6 7 8'],...
0020         ['m-ctl-point constant 4.75']
0021         }, 1.0);
```

[3]Folder: `FAESOR/examples/heat_diffusion`

The inner subregion consists of the geometric cell set `subset(gcells,groups{2})` (`groups{2}` is a list of indexes of the cells that belong to the subregion 2, and the method `subset` extracts from `gcells` the connectivities from the geometric cell set in the list `groups{2}`). Note that the local material directions matrix `Rm` is being supplied to the finite element block constructor (line 0027).

```
0022 propinner = property_diffusion(struct('conductivity',kappainner,...
0023     'source',0));
0024 materinner=mater_diffusion(struct('property',propinner));
0025 febinner = feblock_diffusion(struct ('mater',materinner,...
0026     'gcells',subset(gcells,groups{2}),...
0027     'integration_rule',tri_rule(num_integ_pts),'Rm',Rm));
0028 propouter = property_diffusion(struct('conductivity',kappaouter,...
0029     'source',0));
0030 materouter=mater_diffusion(struct('property',propouter));
0031 febouter = feblock_diffusion(struct ('mater',materouter,...
0032     'gcells',subset(gcells,groups{1}),...
0033     'integration_rule',tri_rule(num_integ_pts)));
```

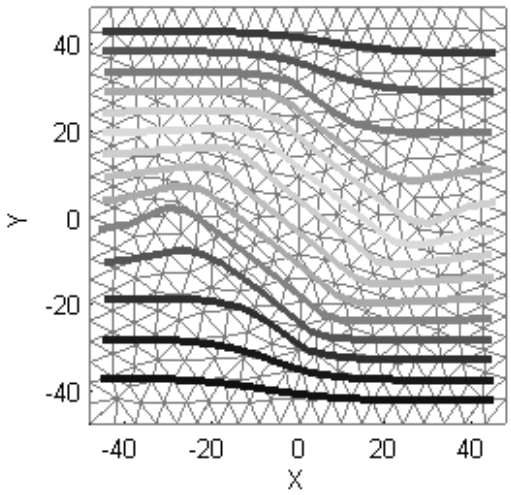


Fig. 9.4. Heat diffusion in inhomogeneous domain with orthotropic material properties: temperature distribution. Notice the distorting effect of the insert

The boundary conditions are straightforward, but notice that the two horizontal edges are being assigned different, nonzero, temperatures.

```
0036 fenids=[fenode_select(fens,struct('box',[-48 48 -48 -48],
0037     'inflate', 0.01))];
0038 prescribed=ones(length(fenids),1);
0039 comp=[];
0040 val=zeros(length(fenids),1)+20;% cold
0041 theta = set_ebc(theta, fenids, prescribed, comp, val);
0042 fenids=[fenode_select(fens,struct('box',[-48 48 48 48],
0043     'inflate', 0.01))];
0044 prescribed=ones(length(fenids),1);
0045 comp=[];
0046 val=zeros(length(fenids),1)+57;% hot
0047 theta = set_ebc(theta, fenids, prescribed, comp, val);
0048 theta = apply_ebc (theta);
```

When assembling the conductivity matrix, the contributions from the two blocks are assembled separately. The thermal loads corresponding to nonzero essential boundary conditions (conductivity only: recall that this is steady-state) are assembled only for the outer subregion block, since there are no boundary conditions on the boundary of the inner block.

```
0050 K = start (sparse_sysmat, get(theta, 'neqns'));
0051 K = assemble (K, conductivity(febinner, geom, theta));
0052 K = assemble (K, conductivity(febouter, geom, theta));
0053 F = start (sysvec, get(theta, 'neqns'));
0054 F = assemble (F,nz_ebc_loads_conductivity(febouter, geom, theta));
```

The results are presented in Fig. 9.4 using level curves of temperature (using the function `tricontour`[4]). The distorting effect of the insert is noteworthy: in one direction the insert conducts heat readily (along the fibers the gradient is relatively small), in the perpendicular direction it is an insulator (across the fibers the gradient of temperature is large). The level curves are they useful visual means of checking the correctness of the application of boundary conditions. For instance, we can readily check that the vertical boundaries are insulated: if the level curves run into a boundary at a right angle, no heat flows through the boundary.

9.4 The T4 NAFEMS Benchmark

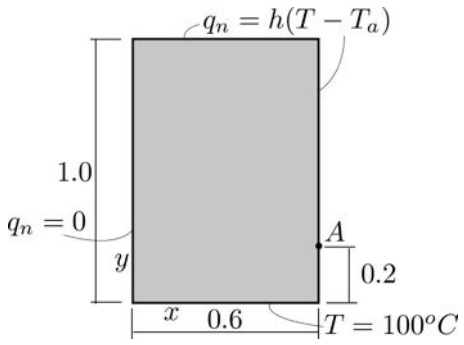


Fig. 9.5. The T4 NAFEMS benchmark geometry and boundary conditions.

This problem is one of the NAFEMS (National Agency for Finite Element Methods and Standards (UK)) benchmark tests for thermal analyses. The boundary conditions are formulated for a rectangle 0.6 meter wide by 1 meter high, with a fixed temperature of 100°C on the lower boundary, perfect insulator on the left boundary, and a heat transfer at 750W/(m^2°C) on the other two boundaries (see Fig. 9.5). The material in the region has a thermal conductivity of 52W/(m°C). The problem is to calculate the steady-state temperature distribution. A complete description of this problem is given in the paper by Cameron, Casey, and Simpson [CCS94].

The `FAESOR` solution is the Matlab script `t4nafems`[5]. Note well that two blocks are being created: the first for the triangular elements in the interior of the domain, and the second, `edgefeb`, for the edge elements (line segment elements with two nodes) along the two boundary edges of the domain with the convective boundary condition.

```
0019 edgefeb = feblock_diffusion (struct ('mater',mater,...
0020     'gcells',subset(edge_gcells,[edge_groups{[2, 3, 4]}]),...
0021     'integration_rule',gauss_rule(1,num_integ_pts),...
```

[4]Folder: `FAESOR/util`
[5]Folder: `FAESOR/examples/heat_diffusion`

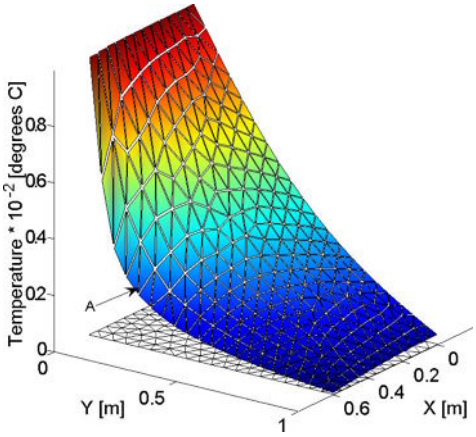


Fig. 9.6. Temperature distribution for the T4 NAFEMS benchmark.

```
0022     'surface_transfer', h));
...
```

Next, we create a field to represent the prescribed ambient temperature along the boundary. The interior values are never used, only the ones on the boundary. They happen to be all equal to zero, but we will not ignore them in the interest of clarity.

```
0026 amb = clone(theta, ['amb']);
0027 fenids=[
0028     fenode_select(fens,struct('box',[0.6 0.6 0 1],...
0029     'inflate', 0.01)),...
0030     fenode_select(fens,struct('box',[0 1 1 1],...
0031     'inflate', 0.01))]   ;
0032 prescribed=ones(length(fenids),1);
0033 comp=[];
0034 val=zeros(length(fenids),1)+0.0;
0035 amb = set_ebc(amb, fenids, prescribed, comp, val);
0036 amb = apply_ebc (amb);
```

The essential boundary condition on the temperature field is applied.

```
0037 fenids=[
0038     fenode_select(fens,struct('box',[0. 0.6 0 0],...
0039     'inflate', 0.01))]    ;
0040 prescribed=ones(length(fenids),1);
0041 comp=[];
0042 val=zeros(length(fenids),1)+100.0;
0043 theta = set_ebc(theta, fenids, prescribed, comp, val);
0044 theta = apply_ebc (theta);
```

The system matrix and the system load vector are assembled, including the surface heat transfer contribution (line 0047), and the surface heat transfer load (line 0051). Note that these are computed on the edge element block `edgefeb`. The contribution of the nonzero prescribed temperature is also added in.

```
0046 K = start (sparse_sysmat, get(theta, 'neqns'));
0047 K = assemble (K, conductivity(feb, geom, theta));
0048 K = assemble (K, surface_transfer(edgefeb, geom, theta));
0049 F = start (sysvec, get(theta, 'neqns'));
0050 F = assemble(F, source_loads(feb, geom, theta));
```

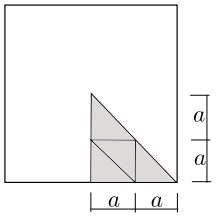
```
0051 F = assemble(F, nz_ebc_loads_conductivity(feb, geom, theta));
0052 F = assemble(F, nz_ebc_loads_surface_transfer(feb, geom, theta));
0053 F = assemble(F, surface_transfer_loads(edgefeb,geom,theta,amb));
```

After the solution, the temperature at the node $x = 0.6$, $y = 0.2$ (label A in Fig. 9.6), is retrieved with `gather` and printed. The calculated value of 18.1969°C agrees well with the reference solution of 18.3°C. The singularity near the corner where the two kinds of boundary conditions meet (prescribed temperature with convective surface heat transfer) is clearly visible (very large slope, that in the limit tends to infinity).

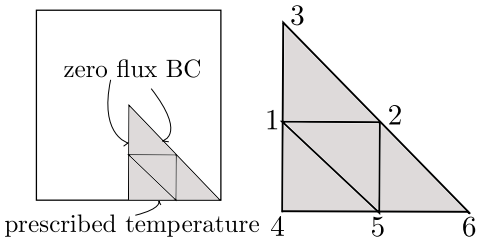
```
0054 theta = scatter_sysvec(theta, get(K,'mat')\get(F,'vec'));
0055 gather(theta,fenode_select(fens,...
0056  struct('box',[0.6 0.6 0.2 0.2],'inflate',0.01)),'values')
```

Exercise 47.

Solve the steady-state temperature distribution using the model of four finite elements of 1/8th of the square domain using two planes of symmetry (see figure below). The temperature around the circumference is prescribed as 20°C. Heat is generated in the interior of the rate of 15W/m^3. Assume homogeneous isotropic material.



Solution: On the the symmetry planes we apply the zero-flux boundary conditions. On the remaining side of the triangle the temperature is prescribed.



The nodes with the free degrees of freedom are numbered first, then the nodes with the prescribed temperatures. Therefore, in this case we can take the equation numbers to be the same as the node numbers.

The finite elements can be numbered so that the total effort is minimized. All elements have the same shape and the same dimensions, and we can take advantage of it by using the numbering of the nodes that always starts from the right angle:

Element	Node connectivity
1	4,5,1
2	5,6,2
3	1,2,3
4	2,1,5

Translation or rotation of the element does not affect the conductivity matrix. Since our numbering ensures that when we superimpose the triangles based on the numbering of the nodes the triangles coincide, the conductivity matrix may be computed just once and assembled four times.

For example, we may compute connectivity matrix of element 3. Note that we can arbitrarily choose the origin of the coordinate system since nothing in the expression for the conductivity matrix depends on the coordinates directly. For simplicity we may take

```
syms a real
x= [0,0;a,0;0,a];
```

The Jacobian matrix may be computed as in (8.54)

```
Nder= [-1,-1;1,0;0,1];
J=x'*Nder
J =
[ a, 0]
[ 0, a]
```

and the Jacobian is seen to be

```
>> det(J )
ans =
a^2
```

From formula (8.53) we compute the Cartesian gradient of the basis functions as

```
Ndersp=Nder/J
Ndersp =
[ -1/a, -1/a]
[  1/a,    0]
[    0,  1/a]
```

The same expression is available by inspection (run-over-rise for the slopes!).

The one-point quadrature rule applied to the conductivity matrix (8.64) will simplify (see Exercise 37) to the product of the basis function gradient matrices multiplied with the area of the element, the uniform thermal conductivity, and the slice thickness (which we call Δz), which we may write by abusing the Matlab notation

$$K^{(3)} = \int_{\triangleright_3} (\mathrm{grad}N)\, \boldsymbol{\kappa} (\mathrm{grad}N)^T \, \Delta z \, \mathrm{d}S \; = (a^2)/2\texttt{*Ndersp*Ndersp}'\kappa\Delta z$$

and evaluating therefore

```
>> syms Dz kappa real
K3= (a^2/2*Ndersp*Ndersp'*kappa*Dz)
K3 =
[      Dz*kappa, -(Dz*kappa)/2, -(Dz*kappa)/2]
[ -(Dz*kappa)/2,  (Dz*kappa)/2,              0]
[ -(Dz*kappa)/2,             0,  (Dz*kappa)/2]
```

or

$$K^{(3)} = \kappa\Delta z \begin{bmatrix} 1 & -1/2 & -1/2 \\ -1/2 & 1/2 & 0 \\ -1/2 & 0 & 1/2 \end{bmatrix}$$

As advertised, this is the only elementwise conductivity matrix we need to compute. It just needs to be assembled four times, into different locations in the global matrix – that is understood. For instance, for the first element we assemble:

```
eq= [4,5,1];
K=sym(zeros(6 ));
K(eq,eq)=K(eq,eq)+K3
K =
```

```
[  (Dz*kappa)/2, 0, 0, -(Dz*kappa)/2,             0, 0]
[             0, 0, 0,             0,             0, 0]
[             0, 0, 0,             0,             0, 0]
[ -(Dz*kappa)/2, 0, 0,      Dz*kappa, -(Dz*kappa)/2, 0]
[             0, 0, 0, -(Dz*kappa)/2,  (Dz*kappa)/2, 0]
[             0, 0, 0,             0,             0, 0]
```

The 6×6 global conductivity matrix results by assembling all four elements as

$$[K] = \kappa \Delta z \begin{bmatrix} 2, & -1, & -1/2, & -1/2, & 0, & 0 \\ -1, & 2, & 0, & 0, & -1, & 0 \\ -1/2, & 0, & 1/2, & 0, & 0, & 0 \\ -1/2, & 0, & 0, & 1, & -1/2, & 0 \\ 0, & -1, & 0, & -1/2, & 2, & -1/2 \\ 0, & 0, & 0, & 0, & -1/2, & 1/2 \end{bmatrix} \tag{9.2}$$

which means that the free-free 3×3 conductivity matrix is obtained from the upper left submatrix as

$$[K] = \kappa \Delta z \begin{bmatrix} 2, & -1, & -1/2 \\ -1, & 2, & 0 \\ -1/2, & 0, & 1/2 \end{bmatrix}$$

The 6×6 global conductivity matrix may also be used to construct the thermal loads due to nonzero essential boundary conditions. The prescribed degrees of freedom $T_4 = T_5 = T_6 = 20$ multiply the last three columns, to yield for the first three rows the contribution to the heat load vector

$$[L] = -\kappa \Delta z \begin{bmatrix} -1/2, \\ 0, \\ 0, \end{bmatrix} T_4 - \kappa \Delta z \begin{bmatrix} 0, \\ -1, \\ 0, \end{bmatrix} T_5 - \kappa \Delta z \begin{bmatrix} 0, \\ 0, \\ 0, \end{bmatrix} T_6 = -20 \kappa \Delta z \begin{bmatrix} -1/2, \\ -1, \\ 0, \end{bmatrix}$$

At this point we can verify that if there is no internal rate of heat generation the temperature should be the same everywhere, equal to the temperature on the boundary. We can compute

```
>> K=K(1:3,1:3)
L=-20*Dz*kappa*[-1/2;-1;0]
K\L
K =
[    2*Dz*kappa,  -Dz*kappa, -(Dz*kappa)/2]
[     -Dz*kappa, 2*Dz*kappa,             0]
[ -(Dz*kappa)/2,          0,  (Dz*kappa)/2]
L =
 10*Dz*kappa
 20*Dz*kappa
           0
ans =
 20
 20
 20
```

where we see the correct $T_1 = T_2 = T_3 = 20$.

The contribution of the internal heat generation rate to the global load vector is assembled from elementwise heat load vectors (see Exercise 39) as: node 1 and 2 collect contribution $(1/3)Q(a^2)/2\Delta z$ (one third of heat generation rate times area times thickness) from three elements

$$3 \times (1/3)Q(a^2)/2\Delta z = Q(a^2)/2\Delta z$$

and node 3 collects contribution from a single element. Thus we have the heat load factor due to internal heat generation rate

$$[L] = Q(a^2)/2\Delta z \begin{bmatrix} 1 \\ 1 \\ 1/3 \end{bmatrix}$$

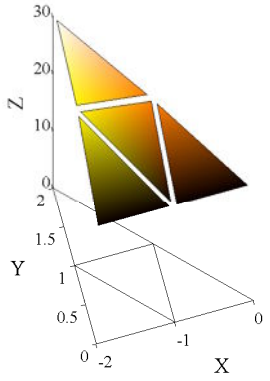
The contribution to the solution of the nodal temperatures due to the internal heat generation rate is therefore computed as

```
L=Q*(a^2)/2*Dz*[1;1;1/3]
K\L
ans =
 (11*Q*a^2)/(12*kappa)
 (17*Q*a^2)/(24*kappa)
   (5*Q*a^2)/(4*kappa)
```

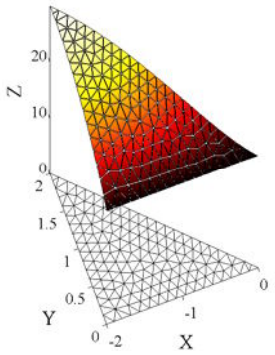
Putting together the contributions to the solution from the essential boundary conditions and from the internal heat generation rate we obtain

$$\begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = \begin{bmatrix} 20 \\ 20 \\ 20 \end{bmatrix} + \frac{Qa^2}{\kappa} \begin{bmatrix} 11/12 \\ 17/24 \\ 5/4 \end{bmatrix}$$

For the numerical values of parameters $Q = 15$, $\kappa = 1.8$, and $a = 1$ we can plot the solution as



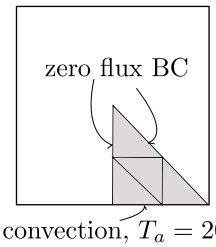
The Matlab script is `pnpSquare`[6] solves this problem using an unstructured grid, and the distribution of temperature is shown below



[6]Folder: `FAESOR/examples/heat_diffusion`

Exercise 48.

Repeat Exercise 47, but instead of essential boundary condition consider Newton's convection surface heat transfer. The ambient temperature around the circumference is prescribed as 20°C. Assume the convection (surface heat transfer) coefficient as a parameter numerically equal twice the value of the thermal conductivity (in consistent physical units). Use the same mesh as in Exercise 47.



Solution: The conductivity matrix (9.2) computed in Exercise 47 can be reused without change. There are six global equations now, and again we take the numbering of the equations to be the same as the numbers of the nodes.

The internal-generation heat load vector is assembled from each triangle by allocating $Q\Delta z S^{(e)}/3$ to each node of the triangular element. Here $S^{(e)} = a^2/2$ is the area of each element. The contributions from the elements are assembled as

$$[L] = Q\Delta z a^2/6 \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \\ 3 \\ 1 \end{bmatrix}$$

On the boundary where surface heats transfer is prescribed we employ two L2 finite elements (nodes 4,5 and 5,6).

The elementwise load vector due to surface heat transfer is the same for both L2 elements. The expression (8.75) developed in Exercise 44 is applicable

$$[L_{q3}] = \frac{hT_a\Delta z h^{(e)}}{2} \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

where $h^{(e)} = a$ is the length of the element, and $h = 2\kappa$. Given the numbering of the equations, the contribution to the global load vector is

$$[L] = \frac{2\kappa T_a \Delta z a}{2} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

The elementwise matrix due to surface heat transfer is the same for both L2 elements. The expression (8.76) developed in Exercise 45 is applicable

$$[H]^{(e)} = \frac{h\Delta z h^{(e)}}{6} \begin{bmatrix} 2, & 1 \\ 1, & 2 \end{bmatrix}$$

The contribution to the global matrix is assembled as

$$[H] = \frac{2\kappa \Delta z a}{6} \begin{bmatrix} 0,0,0,0,0,0 \\ 0,0,0,0,0,0 \\ 0,0,0,0,0,0 \\ 0,0,0,2,1,0 \\ 0,0,0,1,4,1 \\ 0,0,0,0,1,2 \end{bmatrix}$$

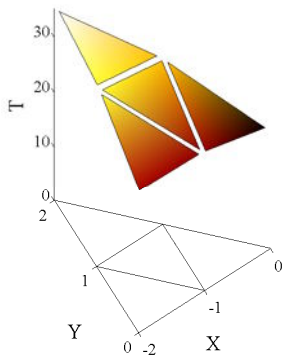
The global system of equations can be finalized as

$$([K] + [H])[T] = Q \Delta z a^2/6 \begin{bmatrix} 3 \\ 3 \\ 1 \\ 1 \\ 3 \\ 1 \end{bmatrix} + \frac{2\kappa T_a \Delta z a}{2} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \end{bmatrix}$$

For the numerical values of parameters $Q = 15$, $\kappa = 1.8$, and $a = 1$ we can plot the solution

$$[T] = \begin{bmatrix} 32.61 \\ 30.81 \\ 35.39 \\ 25.10 \\ 24.84 \\ 21.88 \end{bmatrix}$$

as



10

Transient Heat Conduction Solutions

The ordinary differential equations (8.44) need to be numerically integrated in time as analytical solutions are not possible in general. In this chapter, we will explore a set of methods for numerical integration of the transient response of the heat conduction model.

10.1 Discretization in time for transient heat conduction

Hughes describes a finite difference method, the ***generalized trapezoidal method*** , including its accuracy and stability properties (Chapter 8, Reference [H00]). In order to unclutter the equations we will use the matrix notation, with the following symbols:

$$\boldsymbol{K} = [K_{ji}], \quad j,i = 1, ..., N_{\mathrm{f}} \tag{10.1}$$

for the conductivity matrix,

$$\boldsymbol{H} = [H_{ji}], \quad j,i = 1, ..., N_{\mathrm{f}} \tag{10.2}$$

for the surface heat transfer matrix,

$$\boldsymbol{C} = [C_{ji}], \quad j,i = 1, ..., N_{\mathrm{f}} \tag{10.3}$$

for the capacity matrix, and

$$\boldsymbol{L} = [L_{\overline{C},j} + L_{\overline{K},j} + L_{\overline{H},j} + L_{Q,j} + L_{q2,j} + L_{q3,j}], \quad j = 1, ..., N_{\mathrm{f}}. \tag{10.4}$$

The free temperatures and their rates are collected in column matrices

$$\boldsymbol{T} = [T_j], \quad \dot{\boldsymbol{T}} = \left[\frac{\partial T_i}{\partial t}\right], \quad \text{free } j \ , \tag{10.5}$$

and the prescribed temperatures and their rates

$$\overline{\boldsymbol{T}} = [\overline{T}_j], \quad \dot{\overline{\boldsymbol{T}}} = \left[\frac{\partial \overline{T}_i}{\partial t}\right], \quad \text{prescribed } j \ . \tag{10.6}$$

Therefore, Eq. (8.44) may be recast as

$$\boldsymbol{C}\dot{\boldsymbol{T}} + (\boldsymbol{K} + \boldsymbol{H})\boldsymbol{T} - \boldsymbol{L} = \boldsymbol{0} \ . \tag{10.7}$$

The generalized trapezoidal method proposes to express the relationship between the temperatures and the rates of temperatures at two different time instants, t_n and t_{n+1}, as

$$\theta\dot{\boldsymbol{T}}_{n+1} + (1-\theta)\dot{\boldsymbol{T}}_n = \frac{\boldsymbol{T}_{n+1} - \boldsymbol{T}_n}{\Delta t} \ , \tag{10.8}$$

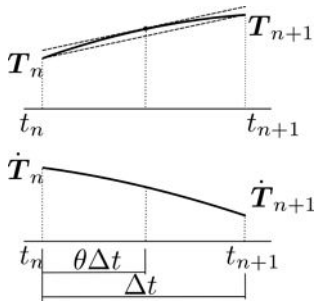


Fig. 10.1. Illustration of the formula (10.8)

where a quantity expressed at time t_n is given a subscript n, and $\Delta t = t_{n+1} - t_n$. The free parameter θ is used to control accuracy and stability of the scheme; see Figure 10.1.

Equation (10.8) is applied to the time stepping of (10.7) by writing it at the two time instants, t_n and t_{n+1}, and then mixing together these two equations. Thus, we add together

$$\theta \left[\boldsymbol{C}\dot{\boldsymbol{T}}_{n+1} + (\boldsymbol{K}+\boldsymbol{H})\boldsymbol{T}_{n+1} - \boldsymbol{L}_{n+1} \right] = \boldsymbol{0} \ , \tag{10.9}$$

and

$$(1-\theta) \left[\boldsymbol{C}\dot{\boldsymbol{T}}_{n} + (\boldsymbol{K}+\boldsymbol{H})\boldsymbol{T}_{n} - \boldsymbol{L}_{n} \right] = \boldsymbol{0} \ , \tag{10.10}$$

and if we assume that Eq. (10.8) applies not only to the free temperatures, but also to the prescribed temperatures, the mixture of rates (left-hand side of (10.8)) may be replaced with the divided difference of the temperatures (right-hand side of (10.8)). The resulting equation refers only to temperatures at two time instants, and may be solved to yield $\boldsymbol{T}_{n+1}$, provided $\boldsymbol{T}_n$ is known.

$$\left[\frac{1}{\Delta t}\boldsymbol{C} + \theta(\boldsymbol{K}+\boldsymbol{H}) \right] \boldsymbol{T}_{n+1} = \left[\frac{1}{\Delta t}\boldsymbol{C} - (1-\theta)(\boldsymbol{K}+\boldsymbol{H}) \right] \boldsymbol{T}_n + \theta \boldsymbol{L}_{n+1} + (1-\theta)\boldsymbol{L}_n \tag{10.11}$$

The essential-boundary-condition loads terms are computed in `FAESOR` by writing

$$\begin{aligned} &\theta[L_{\overline{K},j}]_{n+1} + (1-\theta)[L_{\overline{K},j}]_{n+1} = \\ &\quad - \sum_{i=N_{\mathrm{f}}+1}^{N} \left[\int_{S_c} (\mathrm{grad} N_{\langle j \rangle})\, \boldsymbol{\kappa} (\mathrm{grad} N_{\langle i \rangle})^T \, \Delta z \, \mathrm{d}S \right] (\theta \overline{T}_{i,n+1} + (1-\theta)\overline{T}_{i,n}) \ , \quad i,j = 1, ..., N_{\mathrm{f}}. \end{aligned} \tag{10.12}$$

and

$$\begin{aligned} &\theta[L_{\overline{H},j}]_{n+1} + (1-\theta)[L_{\overline{H},j}]_{n+1} = \\ &\quad - \sum_{i=N_{\mathrm{f}}+1}^{N} \left[\int_{C_{c,3}} N_{\langle j \rangle} \, h N_{\langle i \rangle} \, \Delta z \, \mathrm{d}C \right] (\theta \overline{T}_{i,n+1} + (1-\theta)\overline{T}_{i,n}) \ , \quad i,j = 1, ..., N_{\mathrm{f}}. \end{aligned} \tag{10.13}$$

and for the heat load driven by prescribed temperature rates

$$\begin{aligned} &\theta[L_{\overline{C},j}]_{n+1} + (1-\theta)[L_{\overline{C},j}]_{n+1} = \\ &\quad - \sum_{i=N_{\mathrm{f}}+1}^{N} \left[\int_{S_c} N_{\langle j \rangle} c_V N_{\langle i \rangle} \, \Delta z \, \mathrm{d}S \right] \frac{\overline{T}_{i,n+1} - \overline{T}_{i,n}}{\Delta t} \ , \quad i,j = 1, ..., N_{\mathrm{f}}. \end{aligned} \tag{10.14}$$

where we approximate the prescribed temperature rate rather than use its exact value.

How should we choose the value of θ? The reasoning behind the formula (10.8) could be explained as follows: to get from $\boldsymbol{T}_n$ to $\boldsymbol{T}_{n+1}$ is possible by choosing the correct rate (slope of the straight line

in Figure 10.1). Since we in general don't know at what time instant to compute the correct rate, we guess that the rate is a mixture of the rates at the beginning and the end of the time interval. This would be perfectly reasonable for variations of temperature along a quadratic curve, since then the rate would vary linearly between $\dot{\boldsymbol{T}}_n$ and $\dot{\boldsymbol{T}}_{n+1}$ and taking $\theta = 1/2$ would be exact. Even when the temperature does not vary quadratically $\theta = 1/2$ is close to optimal.

It is sometimes also attractive to make a choice which is non-optimal from the point of view of accuracy. Upon closer inspection of Eq. (10.8) we may conclude that the two choices, $\theta = 0$ and $\theta = 1$, will lead to Euler methods – the ***forward*** (explicit) ***Euler*** for the former, and the ***backward*** (implicit) ***Euler*** for the latter. The value of $\theta = 1/2$ is known as the ***Crank-Nicolson*** method.

The explicit Euler method results in the formula for the advancement of the solution

$$\frac{1}{\Delta t}\boldsymbol{C}\boldsymbol{T}_{n+1} = \left[\frac{1}{\Delta t}\boldsymbol{C} - (\boldsymbol{K} + \boldsymbol{H})\right]\boldsymbol{T}_n + \boldsymbol{L}_n \tag{10.15}$$

which may be computationally very efficient if $\boldsymbol{C}$ is a diagonal matrix. Since this can be easily arranged, the forward Euler method can solve $\boldsymbol{T}_{n+1}$ very quickly in each step in time. Unfortunately, this advantage is to a certain degree canceled by the need to make very many short steps, since the forward Euler method is only conditionally stable (the condition being that the step may be at most a certain length).

The implicit Euler method results in the formula

$$\left[\frac{1}{\Delta t}\boldsymbol{C} + (\boldsymbol{K} + \boldsymbol{H})\right]\boldsymbol{T}_{n+1} = \left[\frac{1}{\Delta t}\boldsymbol{C}\right]\boldsymbol{T}_n + \boldsymbol{L}_{n+1} \tag{10.16}$$

As we can see, the solution of the linear system of equations for $\boldsymbol{T}_{n+1}$ needs to deal with a matrix that is a combination of the capacity, conductivity, and surface heat transfer matrices. Therefore it is not likely to be diagonal, and each solution would tend to be expensive. In order to save time one could factor the matrix $\left[\frac{1}{\Delta t}\boldsymbol{C} + (\boldsymbol{K} + \boldsymbol{H})\right]$ before the time stepping starts and then use forward and backward substitution in each time step. This is going to be more expensive per step than the explicit Euler, but the implicit Euler method can make very large time steps to compensate for this expense as it is unconditionally stable (any time step length would work).

Finally, formula (10.11) with $\theta = 1/2$ is optimally accurate. For this value of the parameter the algorithm is called Crank-Nicolson. It is also unconditionally stable, and one order more accurate (it is a second order method) than either explicit or implicit Euler (which are only first order methods). Unfortunately, sometimes Crank-Nicolson generates oscillations in the computed temperatures. The oscillations eventually go away with reduction in the time step length, but the backward Euler never gives oscillations, and if its lower accuracy is acceptable it is the preferred method for this type of problem.

10.2 The T3 NAFEMS Benchmark

This test is recommended by the UK National Agency for Finite Element Methods and Standards (NAFEMS). The domain shown in Fig. 10.2. One face is held at 0°C, the other face experiences sinusoidal variations in temperature. The temperature at $t = 32$ seconds 0.02 m under the heated face is sought. It is assumed that the plate is very large compared to its thickness, and the problem may therefore be reduced to one dimension, along the thickness. The implementation of transient heat conduction in `FAESOR` is in fact dimension independent, and we simply take care to define the various objects properly for a 1-D problem and the rest is common to all scripts from the heat diffusion examples folder.

The solution is presented in the Matlab script `t3nafems`[1]. First, the various parameters are defined. Note that the backward Euler method ($\theta = 1$) is selected for the time discretization.

[1] Folder: `FAESOR/examples/diffusion`

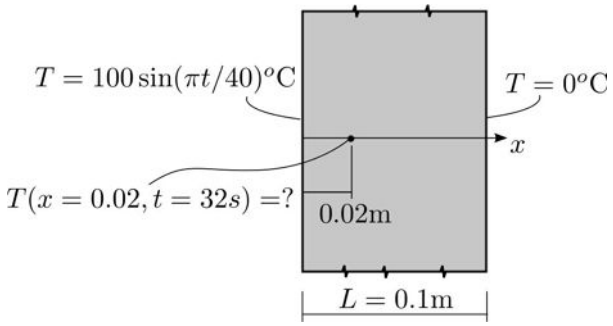


Fig. 10.2. Heat diffusion through a plate (one-dimensional problem), with time-dependent boundary conditions

```
0001 kappa=[35.0]; % conductivity matrix
0002 cm = 440.5;% specific heat per unit mass
0003 rho=7200;% mass density
0004 cv =cm* rho;% specific heat per unit volume
0005 Q=0; % uniform heat source
0006 Tampl=100;
0007 Tamb=0;
0008 Tbar =@(t)(Tampl*sin(pi*t/40)+ Tamb);%hot face temp.
0009 num_integ_pts=2; % quadrature
0010 L=0.1;% thickness
0011 dt=0.5; % time step
0012 tend= 32; % length of the time interval
0013 t=0;
0014 theta = 1.0; % generalized trapezoidal method
0015 online_graphics= ~true;% plot the solution?
0016 n=3*5;% needs to be multiple of five
```

The mesh is created by `L2_block`[2], a simple meshing function which produces a uniformly spaced mesh on the interval $0 \leq x \leq L$. Note that not only the essential boundary conditions are applied to the temperature field, but also the initial condition (which happens to be 0°C) on line `0032`.

```
0017 [fens,gcells] = L2_block(L,n,1.0); % Mesh
0018 prop=property_diffusion(struct('conductivity',kappa,...
0019     'specific_heat',cv,'rho',rho,'source',Q));
0020 mater=mater_diffusion (struct('property',prop));
0021 feb = feblock_diffusion (struct (...
0022     'mater',mater,...
0023     'gcells',gcells,...
0024     'integration_rule',gauss_rule(1,num_integ_pts)));
0025 geom=field(struct('name',['geom'],'dim', 1, 'fens',fens));
0026 tempn = field(struct ('name',['temp'], 'dim', 1,...
0027     'nfens',get(geom,'nfens')));
0028 tempn = set_ebc(tempn, 1, 1, 1, Tbar(t));
0029 tempn = set_ebc(tempn, n+1, 1, 1, Tamb);
0030 tempn = apply_ebc (tempn);
0031 tempn = numbereqns (tempn);
0032 tempn = scatter_sysvec(tempn,gather_sysvec(tempn)*0+Tamb);
```

The conductivity and capacity matrix are time independent; we compute them once, and henceforth work only with the arrays `Km` and `Cm`.

[2]Folder: `FAESOR/meshing`

```
0033 K = start (dense_sysmat, get(tempn, 'neqns'));
0034 K = assemble (K, conductivity(feb, geom, tempn));
0035 Km = get(K,'mat');
0036 C = start (dense_sysmat, get(tempn, 'neqns'));
0037 C = assemble (C, capacity(feb, geom, tempn));
0038 Cm = get(C,'mat');
```

The time stepping begins. First, the temperature boundary conditions are time-dependent, which means they have to be set for each pass through the time loop (i.e. for each time instant).

```
0039 Tfifth = [];
0040 while t<tend+0.1*dt % Time stepping
0041     if online_graphics
...
0046     end
0047     tempn1 = tempn;
0048     tempn1 = set_ebc(tempn1, 1, 1, 1, Tbar(t+dt));
0049     tempn1 = set_ebc(tempn1, n+1, 1, 1, Tamb);
0050     tempn1 = apply_ebc (tempn1);
```

The thermal loads corresponding to nonzero temperatures and temperature rates are applied next. We may compare the fields that are being passed on lines `0052` and `0054` with (10.11) and the discussion below that equation: The Matlab code is a literal transcription of the formulas. Note that we are directly working with objects of the class `field`, using operator overload (adding and multiplying fields).

```
0051     F = start (sysvec, get(tempn, 'neqns'));
0052     F = assemble (F, nz_ebc_loads_conductivity(feb, geom,
0053         theta*tempn1 + (1-theta)*tempn));
0054     F = assemble (F, nz_ebc_loads_capacity(feb, geom, ...
0055         (tempn1-tempn)*(1/dt)));
0056     Tn=gather_sysvec(tempn);
0057     Tfifth = [Tfifth Tn(n/5)];
```

The individual objects in this system of linear equations for `Tn1` are again directly recognizable in formula (10.11).

```
0058     Tn1=(1/dt*Cm+theta*Km)\((1/dt*Cm-(1-theta)*Km)*Tn+...
0059         get(F,'vec'));
0060     tempn = scatter_sysvec(tempn1,Tn1);
0061     t=t+dt;
0062 end
```

The results are summarized in Fig. 10.3. The NAFEMS-reported reference solution is 36.6°C at the time $t = 32$ seconds, and the solid curve shown in the figure was obtained with 200 elements through the thickness with time step $\Delta t = 0.1$ (yielding 36.56°C, that is 99.9% of the reference temperature). The reference solution is further compared with results for 15 elements used with time step $\Delta t = 0.5$ (dotted line) that gave 36.69°C, that is 100.26% of the reference temperature. The agreement is rather good, despite the presence of significant temperature gradients near the variable-temperature surface.

10.3 Transient cooling in a shrink-fitting application

Shrink fitting is a common manufacturing process used to assemble two parts. The two parts are tempered to different temperatures in order for one to shrink and the other one to expand. Figure 10.4 shows of a cross-section of the three-dimensional steel block into which a tungsten insert is to be

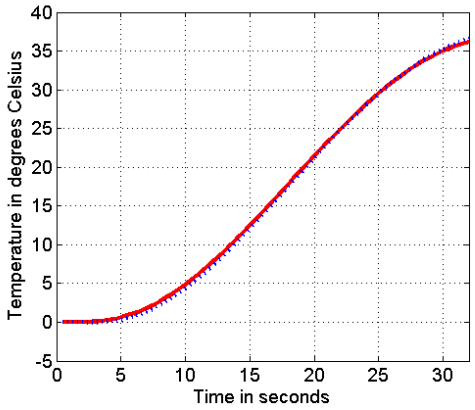


Fig. 10.3. Heat diffusion through a plate (one-dimensional problem): temperature 0.02 m under the heated face

fitted. In our case, the cold part is maintained at $-10°$C prior to the assembly, while the hot part is at 84°C. The temperature of the ambient air is 17°C. The task is to determine how long it will take before the temperature of the hot part drops below 75°C (which is given as a manufacturing constraint).

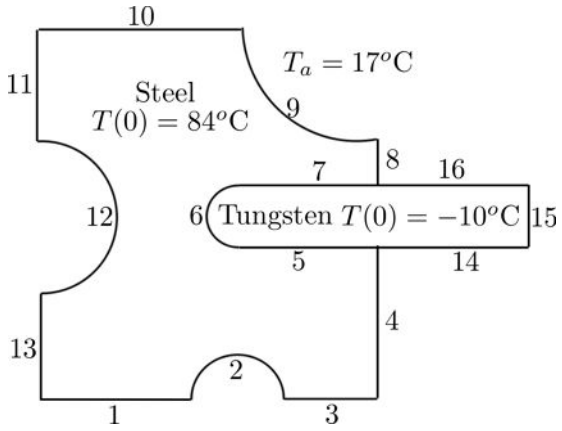


Fig. 10.4. Transient cooling of a shrink-fitted assembly: schematic

The problem is solved by the script `shrinkfit`[3]. Let us begin with the mesh generation: Figure 10.4 shows the two regions (steel and tungsten), and the boundary edges (note the numbers next to the edges). The mesh is relatively coarse considering the thickness of some of the geometry – only around three elements through the thickness of the cold part. Even so the mesh has almost 2300 triangular elements, and the transient solution takes a couple of minutes.

```
0017 [fens,gcells,groups,edge_gcells,edge_groups]=
           targe2_mesher({...
0018     'curve 1 line 0 0 50 0',...
0019     'curve 2 arc 50 0 80 0 center 65 -0.001 ',...
0020     'curve 3 line 80 0 110 0',...
0021     'curve 4 line 110 0 110 50',...
0022     'curve 5 line 110 50 65 50 ',...
```

[3]Folder: `FAESOR/examples/diffusion`

```
0023     'curve 6 arc 65 50 65 70 center 65.001 60  ',...
0024     'curve 7 line 65 70 110 70',...
0025     'curve 8 line 110 70 110 85',...
0026     'curve 9 arc 110 85 65 120 center 110 120 ',...
0027     'curve 10 line 65 120 0 120',...
0028     'curve 11 line 0 120 0 85',...
0029     'curve 12 arc 0 85 0 35 center -0.001 60 rev',...
0030     'curve 13 line 0 35 0 0',...
0031     'curve 14 line 110, 50, 160, 50',...
0032     'curve 15 line 160, 50, 160, 70',...
0033     'curve 16 line 160, 70, 110, 70',...
0034     ['subregion 1  property 1 boundary '...
0035     ' 1 2 3 4 5 6 7 8 9 10 11 12 13'],...
0036     ['subregion 2  property 2 boundary '...
0037     ' -5 -6 -7 14 15 16'],...
0038     ['m-ctl-point constant 3']
0039     }, 1.0);
```

The property, material, and block objects are created for each material (steel, tungsten) separately.

```
0040 prop_steel=...
0041  property_diffusion (struct('conductivity',kappa_steel,...
0042  'specific_heat',cv_steel,'source',0.0));
0043 mater_steel=mater_diffusion(struct('property',prop_steel));
0044 feb_steel=feblock_diffusion(struct('mater',mater_steel,...
0045  'gcells',gcells(groups{1}),...
0046  'integration_rule',tri_rule(num_integ_pts)));
0047 prop_tungsten=...
0048  property_diffusion (struct('conductivity',kappa_tungsten,
0049  'specific_heat',cv_tungsten,'source',0.0));
0050 mater_tungsten=mater_diffusion
           (struct('property',prop_tungsten));
0051 feb_tungsten = feblock_diffusion (
           struct ('mater',mater_tungsten,...
0052  'gcells',gcells(groups{2}),...
0053  'integration_rule',tri_rule(num_integ_pts)));
```

For the edges that separate the metal from the air, elements to be used in the surface heat transfer needs to be generated (finite element block `efeb`). Note that the interior edges are omitted.

```
0054 edge_gcells=edge_gcells([edge_groups{[(1:4) (8:16)]}]);
0055 efeb = feblock_diffusion (struct ('mater',mater_steel,...
0056     'gcells',edge_gcells,...
0057     'integration_rule',gauss_rule(1,num_integ_pts),...
0058     'surface_transfer', h));
```

The ambient temperature is defined in the field `amb`. The temperature is applied at the nodes associated with the boundary edges in the loop (line `0063`).

```
0062 amb = clone(tempn, ['amb']);
0063 for i= 1:length(edge_gcells)
0064     conn = get(edge_gcells(i),'conn');
0065     amb = set_ebc(amb, conn, conn*0+1, [], conn*0+Ta);
0066 end
0067 amb = apply_ebc (amb);
```

The time stepping loop is almost identical to the one in the example in Section 10.2, except the thermal load vector is based on the convective surface heat transfer. We solve the problem twice,

each time with time step `dt=2.5` (set on line 11), first time with `theta= 0.5` (Crank-Nicolson) and `theta= 1.0` (backward Euler).

The evolution of the lowest and highest temperature in the entire assembly is shown in Fig. 10.5: the highest temperature drops below 75°C after around 80 seconds. The two solutions are similar in the later stages, but initially the Crank-Nicolson produces oscillations which raise the temperature of the hot part unphysically above its initial temperature. Hence, the backward Euler may be preferable.

This demonstration brings to the fore the questions Should we trust the computed results? How accurate are the results? If the results are not of adequate accuracy, how do I improve them? We do not address these important issues here, neither in the resolution afforded by the mesh, nor in the selection of the time step. Some pointers to how this could be understood are given in Chapter 12.

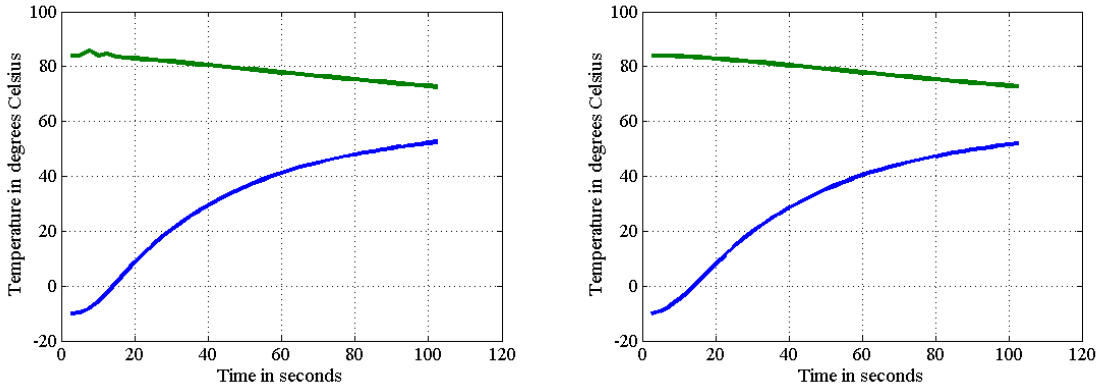
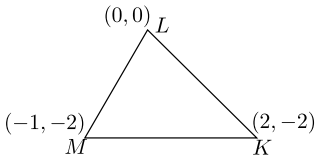


Fig. 10.5. Transient cooling of a shrink-fitted assembly: time evolution of the lowest and highest temperature in the assembly; Crank-Nicolson on the left, backward Euler on the right

---

Exercise 49.

Compute the capacity matrix of the triangular finite element $\triangle KLM$. Assume isotropic homogeneous material. Use sufficiently accurate quadrature rule to evaluate the matrix exactly.



Solution: The single-element capacity matrix is computed from the formula (8.34) for an imaginary mesh that consists of a single element. The capacity matrix links together the three nodes whose basis functions are nonzero over the element. Therefore, the capacity matrix has a dimension of 3×3.

The basis functions are piecewise linear, and the Jacobian of the three-node triangle is constant. Consequently we need to integrate quadratic polynomials. By inspection of Table 8.1 we would estimate that the three point rule would be adequately accurate.

For convenience we will carry out the integration for the entire capacity matrix rather than for the nine individual components. Hence we introduce the matrix of basis function values

$$[N(\xi,\eta)] = \begin{bmatrix} N_K(\xi,\eta) \\ N_L(\xi,\eta) \\ N_M(\xi,\eta) \end{bmatrix}$$

where (ξ,η) are the parametric coordinates at which the function value is taken.

$$\underset{3\times3}{[C^{(e)}]} = \int_{S^{(e)}} c_V [N][N]^T \, \Delta z \, \mathrm{d}S$$

The integral is evaluated with three-point numerical quadrature as

$$\underset{3\times3}{[C^{(e)}]} = \sum_{k=1}^{3} c_V [N(\xi_k, \eta_k)][N(\xi_k, \eta_k)]^T \det[J(\xi_k, \eta_k)] W_k \Delta z$$

where c_V, Δz and the Jacobian $\det[J(\xi_k, \eta_k)]$ are constants. The Jacobian is twice the area of the triangle, $\det[J(\xi_k, \eta_k)] = 2S^{(e)} = 6$ (Exercise 36). Just a few lines of Matlab code will make light of the work:

```
>> N=@(xi,eta) [1-xi-eta;xi;eta ];
detJ=6; W=1/6;
N(2/3,1/6)*N(2/3,1/6)'*detJ*W+...
N(1/6,2/3)*N(1/6,2/3)'*detJ*W+...
N(1/6,1/6)*N(1/6,1/6)'*detJ*W
ans =
    0.5000    0.2500    0.2500
    0.2500    0.5000    0.2500
    0.2500    0.2500    0.5000
```

Thus we finally obtain the consistent capacity matrix of the T3 finite element $\triangle KLM$ as

$$\underset{3\times3}{[C^{(e)}]} = \frac{c_V \Delta z}{4} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

This result can be profitably generalized by keeping the area of the element in the formula instead of the numerical value

$$\underset{3\times3}{[C^{(e)}]} = \frac{c_V \Delta z S^{(e)}}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

Note that $\Delta z S^{(e)}$ is the total volume of the element, and $c_V \Delta z S^{(e)}$ is the total heat capacity of the element volume.

A quick check can be used to ascertain that the result is correct: if the nodal temperature rates are the same at all nodes $\dot{T}_K = \dot{T}_L = \dot{T}_M = A$, we get the total power at the nodes

$$\underset{3\times3}{[C^{(e)}]} \begin{bmatrix} \dot{T}_K \\ \dot{T}_L \\ \dot{T}_M \end{bmatrix} = \frac{c_V \Delta z S^{(e)}}{12} \begin{bmatrix} 2 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} A \\ A \\ A \end{bmatrix} = \frac{c_V \Delta z S^{(e)} A}{3} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

which is one third of the total power stored in the element $c_V \Delta z S^{(e)} A$ per node (recall $c_V \Delta z S^{(e)}$ is the total heat capacity of the element, and A is the uniform rate of temperature throughout the element).

11

Expanding the Library of Element Types

The linear triangle T3 is not particularly accurate, but for the linear heat conduction problem it is quite adequate. Nevertheless, we will introduce a selection of other elements to expand the scope of the approximation methods discussed so far. This is desirable from a couple of different viewpoints. Firstly, with the linear triangle we have been able to construct basis functions which allow for linear variations in temperature to be represented exactly. Hence, if the exact solution leads to a constant gradient of temperature, the approximate solution does not involve any discretization error. Unfortunately, constant gradients of temperature are not commonly encountered in applications. If the basis functions can represent higher-order polynomials, for instance quadratic, the resulting method will be able to represent more complex gradients of temperature: linear, in the case of the quadratic variation of temperature. Taking the liberty to oversimplify somewhat, we can say that *the more complex the temperature variations that are reproduced without error, the higher the overall accuracy of the scheme.*

Secondly, introducing different element types may enable us to play games with different quadrature schemes. One view of the finite element method puts the basis function above all: the elements are there only to integrate all the expressions that involve the basis functions and their derivatives as accurately as possible (exactly?). However, there is a virtue in the complementary view: the basis functions are used essentially to ensure a degree of compatibility (fitting together), and the accuracy of the method can be enhanced by tuning the properties of the individual finite elements. One of the dials that can be used for tuning is the quadrature rule: we shall see how to improve the performance of the elements used for stress analysis by employing integration rules which are on purpose *not* doing their job as accurately as possible.

11.1 Quadratic triangle T6

The triangle T6 makes it possible to design basis functions that can reproduce quadratic variations of the temperature. More precisely, it will do that in terms of the coordinates on the standard triangle. As we shall see, the map from the standard triangle will also allow for quadratic temperature variation in the physical space, but more generally it will lead to rational expressions.

The first task will be to formulate the basis functions on the standard triangle, Fig. 11.1. To be able to write down a polynomial for a particular basis function that is quadratic in ξ, η, six coefficients will be needed. To determine these coefficients, we will make use of the common device of equipping the basis functions with the Kronecker delta property (2.23). Let us start with the basis function $N_2 = a_0 + a_1\xi + a_2\eta + a_3\xi\eta + a_4\xi^2 + a_5\eta^2$. Writing

$$N_2(\xi_k, \eta_k) = \delta_{2k}, \quad \text{for } k = 1, ..., 6 \ ,$$

at all six nodes (see Table 11.1), provides us with six equations from which the six coefficients may be determined. That is however tedious and boring: let us use commonsense and guesswork instead. Looking along the η axis we see three (1, 6, 3) and two (4, 5) nodes respectively align at the same

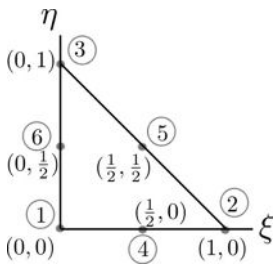


Fig. 11.1. Standard quadratic triangle.

ξ coordinate. Evidently, this makes it possible to design the function N_2 as a Lagrange polynomial that is zero in these two locations, and equal to one at node 2

$$N_2 = \frac{(\xi - 0)(\xi - 1/2)}{(1 - 0)(1 - 1/2)} = \xi(2\xi - 1) \ .$$

Similarly, in the other direction we have for $N_3 = \eta(2\eta - 1)$.

To approach the construction of the other basis functions, we note that both N_2 and N_3 may be written as the normalized product of planes: for N_2 the two planes are $\widehat{p}_2(\xi, \eta) = \xi$ and $\widetilde{p}_2(\xi, \eta) = \xi - 1/2$, and N_2 is written as

$$N_2 = \frac{\widehat{p}_2(\xi, \eta)\widetilde{p}_2(\xi, \eta)}{\widehat{p}_2(1, 0)\widetilde{p}_2(1, 0)} = \xi(2\xi - 1) \ .$$

Similarly for N_3 and N_1: the recipe is to find two planes that go through three nodes and two nodes respectively (but not through the node at which the function is supposed to be equal to one), and normalize their product. For N_1 the planes are $\widehat{p}_1(\xi, \eta) = 1 - \xi - \eta$ (this is the same N_1 as in (8.23)) and $\widetilde{p}_1(\xi, \eta) = 1 - 2\xi - 2\eta$ (compare with Fig. 11.2)

$$N_1 = (1 - \xi - \eta)(1 - 2\xi - 2\eta) \ .$$

Table 11.1. Standard quadratic triangle: locations of the nodes

Coordinate	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
ξ	0	1	0	1/2	1/2	0
η	0	0	1	0	1/2	1/2

For the mid-edge nodes, 4, 5, 6, we find planes that pass through two triples of nodes. For instance, for node 6 (see Fig. 11.2), the two planes are $\widehat{p}_6(\xi, \eta) = 1 - \xi - \eta$ and $\widetilde{p}_6(\xi, \eta) = \eta$ (same as N_3 as in (8.22))

$$N_6 = 4(1 - \xi - \eta)\eta \ .$$

All the basis functions are summarized as

$$[N] = \begin{bmatrix} (\eta + \xi - 1)(2\eta + 2\xi - 1) \\ \xi(2\xi - 1) \\ \eta(2\eta - 1) \\ -4\xi(\eta + \xi - 1) \\ 4\eta\xi \\ -4\eta(\eta + \xi - 1) \end{bmatrix} \tag{11.1}$$

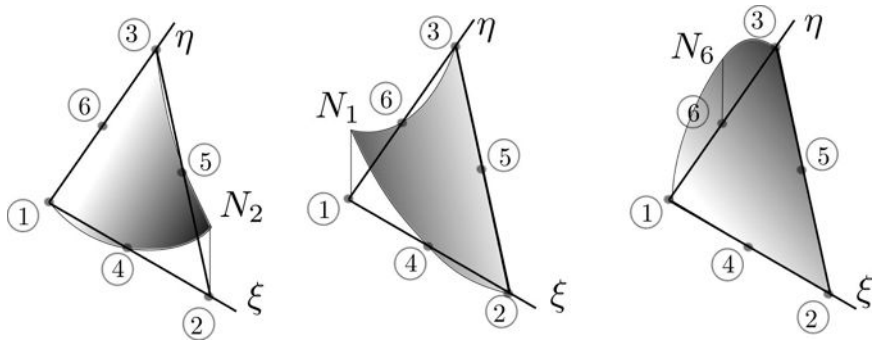


Fig. 11.2. Standard quadratic triangle: Basis functions N_2, N_1, and N_6.

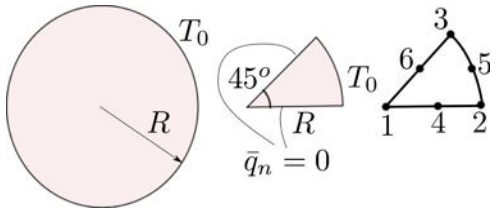


Fig. 11.3. Circular domain of the cross-section of a concrete column immersed in water, quadratic triangle (T6) model

---

Exercise 50. Solve for the temperature distribution in a concrete column of circular cross-section from Exercise 40 using a mesh of a single T6 finite element. Model a 45^o pie slice.

Solution: The mesh is shown in Figure 11.3. The degrees of freedom are numbered as

Node	1	2	3	4	5	6
Degree of freedom (equation) number	1	4	5	2	6	3

This time the calculation will be performed in degrees Celsius. This means that degrees of freedom 4, 5, 6 will be prescribed as $T_4 = T_5 = T_6 = 0^o$C.

The coordinates of the nodes (one per row, x, y in the columns) are given by the matrix

$$[x] = \begin{bmatrix} 0, & 0 \\ 2.5, & 0 \\ 1.7678, & 1.7678 \\ 1.25, & 0 \\ 2.3097, & 0.95671 \\ 0.88388, & 0.88388 \end{bmatrix}$$

The basis functions are given in equation (11.1), and hence the gradient of the basis functions with respect to the parametric coordinates is

$$\left[\text{grad}_{(\xi,\eta)} N\right] = \begin{bmatrix} 4\eta + 4\xi - 3, & 4\eta + 4\xi - 3 \\ 4\xi - 1, & 0 \\ 0, & 4\eta - 1 \\ 4 - 8\xi - 4\eta, & -4\xi \\ 4\eta, & 4\xi \\ -4\eta, & 4 - 4\xi - 8\eta \end{bmatrix}$$

At variance with the T3 finite element the gradients of the basis functions wrt ξ, η are not constant over the element. In this case they vary as linear functions of the parametric coordinates. Therefore,

also the Jacobian matrix is going to be a function of the parametric coordinates. And, the gradients of the basis functions with respect to x, y are going to vary from point to point.

We are ready to start the computation. We will use the three-point quadrature rule from Table 8.1. We define the matrix of the basis functions, and compute symbolically the matrix of the gradients of the basis functions with respect to the parametric coordinates.

```
syms  xi eta kappa real
N= [(eta + xi - 1)*(2*eta + 2*xi - 1)
    xi*(2*xi - 1)
    eta*(2*eta - 1)
    -4*xi*(eta + xi - 1)
    4*eta*xi
    -4*eta*(eta + xi - 1)]
gradNxi=[diff(N,'xi'),diff(N,'eta')]
```

The Jacobian matrix is computed as

```
x= [                   0                   0
    2.500000000000000                   0
    1.767766952966369   1.767766952966369
    1.250000000000000                   0
    2.309698831278217   0.956708580912724
    0.883883476483184   0.883883476483184]
J=x'*gradNxi
```

from the formula (8.54).

The matrix of the gradients of the basis functions with respect to x, y is computed as usual

```
gradN=gradNxi*inv(J)
```

and because of both matrices being functions of the parametric coordinates, especially because the inverse of the Jacobian matrix is needed, it turns out to be a complicated so-called rational polynomial in the parametric coordinates. For instance, one element of the gradient matrix reads

```
>> gradN(1,1)
ans =((861726481700140*xi - 562949953421312*2^(1/2)*xi
+ 281474976710656*2^(1/2))*(4*eta + 4*xi - 3))/(2154316204250350*xi
+ 115956713292207*2^(1/2)*eta - 1407374883553280*2^(1/2)*xi
+ 703687441776640*2^(1/2)) + (4*eta*(140737488355328*2^(1/2)
- 215431620425035)*(4*eta + 4*xi - 3))/(2154316204250350*xi
+ 115956713292207*2^(1/2)*eta - 1407374883553280*2^(1/2)*xi
+ 703687441776640*2^(1/2))
>>
```

The numerical integration is straightforward, but obviously would be very tedious if done by hand. The coordinates of the quadrature points needs to be substituted into the symbolic expressions for `gradN` and `J` using the substitution function `subs`.

```
kappa=1.8; dz=1;
K=zeros(6,6);
xi=2/3; eta=1/6; W=1/6; % location and weight of quadrature point
K =K+subs(kappa*dz*gradN*gradN'*det(J))*W;
xi=1/6; eta=2/3; W=1/6; % location and weight of quadrature point
K =K+subs(kappa*dz*gradN*gradN'*det(J))*W;
xi=1/6; eta=1/6; W=1/6; % location and weight of quadrature point
K =K+subs(kappa*dz*gradN*gradN'*det(J))*W;
```

The resulting elementwise conductivity matrix of the T6 finite element obtains as

$$[K^{(e)}] = \begin{bmatrix} 0.70489, & 0.15571, & 0.15571, & -0.46369, & -0.088922, & -0.46369 \\ 0.15571, & 1.562, & 0.35862, & -0.62766, & -1.3019, & -0.1468 \\ 0.15571, & 0.35862, & 1.562, & -0.1468, & -1.3019, & -0.62766 \\ -0.46369, & -0.62766, & -0.1468, & 4.2903, & -0.58666, & -2.4655 \\ -0.088922, & -1.3019, & -1.3019, & -0.58666, & 3.866, & -0.58666 \\ -0.46369, & -0.1468, & -0.62766, & -2.4655, & -0.58666, & 4.2903 \end{bmatrix} \tag{11.2}$$

The numerical integration above was performed partially with symbolic expressions. To get closer to the "finite element way" of running computations we can rewrite the calculations as follows. We define a *function* to evaluate the matrix of gradients of the basis functions with respect to the parametric coordinates for given values of `xi,eta`:

```
gradNxi =@(xi,eta)([
[ 4*eta + 4*xi - 3, 4*eta + 4*xi - 3]
[         4*xi - 1,                0]
[                0,        4*eta - 1]
[ 4 - 8*xi - 4*eta,            -4*xi]
[           4*eta,             4*xi]
[          -4*eta, 4 - 4*xi - 8*eta]]);
```

Initialize the variables.

```
kappa=1.8; dz=1;
K=zeros(6,6);
```

Now use the above definitions to integrate. We are at the first quadrature point.

```
xi_k=2/3; eta_k=1/6; W_k=1/6; % location and weight of quadrature point
```

The function `gradNxi` called with the arguments `xi_k,eta_k` returns a matrix of numbers.

```
K>> gradNxi(xi_k,eta_k)
ans =
   0.333333333333333   0.333333333333333
   1.666666666666667                   0
                   0  -0.333333333333333
  -2.000000000000000  -2.666666666666667
   0.666666666666667   2.666666666666667
  -0.666666666666667   0.000000000000000
```

Therefore as we compute the Jacobian matrix

```
J=x'*gradNxi(xi_k,eta_k);
```

we get simply a matrix of double-precision numbers.

```
K>> J
J =
   2.617210236530022   2.236607899086455
   0.048550069619693   1.961967231445141
```

Consequently, the matrix `gradN` is also a number matrix, and the elementwise conductivity matrix will also consist only of numbers.

```
gradN=gradNxi(xi_k,eta_k)*inv(J);
K =K+(kappa*dz*gradN*gradN'*det(J))*W_k;
```

The same series of steps is repeated for the other two quadrature points.

```
xi_k=1/6; eta_k=2/3; W_k=1/6; % location and weight of quadrature point
J=x'*gradNxi(xi_k,eta_k);
gradN=gradNxi(xi_k,eta_k)*inv(J);
K =K+(kappa*dz*gradN*gradN'*det(J))*W_k;
xi_k=1/6; eta_k=1/6; W_k=1/6; % location and weight of quadrature point
J=x'*gradNxi(xi_k,eta_k);
gradN=gradNxi(xi_k,eta_k)*inv(J);
K =K+(kappa*dz*gradN*gradN'*det(J))*W_k;
```

The resulting elementwise conductivity matrix is the same as before. If you actually run this code you will notice that the numerical calculation runs about 100 times faster than the symbolic one. Numerical treatment is in this case the clear winner in the efficiency department.

Using the element equation array $[1, 4, 5, 2, 6, 3]$ we assemble the conductivity matrix of the structure as

$$[K] = \begin{bmatrix} 0.70489, & -0.46369, & -0.46369 \\ -0.46369, & 4.2903, & -2.4655 \\ -0.46369, & -2.4655, & 4.2903 \end{bmatrix}$$

We need to compute the elementwise heat load vector due to internal heat generation. We proceed as in Exercise 43

$$[L^{(e)}_{Q,j}] = \int_{\triangleright_e} N_j Q \; \Delta z \; \mathrm{d}S = Q \Delta z \int_{\triangleright_e} N_j \; \mathrm{d}S \; , \quad j = 1, ..., 6 \; .$$

To evaluate the integrals we will again use numerical integration. The function `N` will return the values of the basis functions at particular coordinates so that we can use it to give us the values of the integrand.

```
N= @(xi,eta)([(eta + xi - 1)*(2*eta + 2*xi - 1)
    xi*(2*xi - 1)
    eta*(2*eta - 1)
    -4*xi*(eta + xi - 1)
    4*eta*xi
    -4*eta*(eta + xi - 1)]);
```

Initialize some variables

```
Q=4.5;
F=zeros(6,1);
```

and start integrating. This is the expression of the first quadrature point. Note that this is pretty much a literal transcription of the numerical integration formula.

```
xi_k=2/3; eta_k=1/6; W_k=1/6; % location and weight of quadrature point
J=x'*gradNxi(xi_k,eta_k);
F =F+(Q*dz*N(xi_k,eta_k)*det(J))*W_k;
```

Repeat for the last two quadrature points

```
xi_k=1/6; eta_k=2/3; W_k=1/6; % location and weight of quadrature point
J=x'*gradNxi(xi_k,eta_k);
F =F+(Q*dz*N(xi_k,eta_k)*det(J))*W_k;
xi_k=1/6; eta_k=1/6; W_k=1/6; % location and weight of quadrature point
J=x'*gradNxi(xi_k,eta_k);
F =F+(Q*dz*N(xi_k,eta_k)*det(J))*W_k;
```

and the elementwise source heat load vector results as

$$[L_Q^{(e)}] = \begin{bmatrix} -0.060688 \\ 0.030344 \\ 0.030344 \\ 3.6483 \\ 3.7394 \\ 3.6483 \end{bmatrix} \tag{11.3}$$

which is readily assembled into the structural heat load vector

$$[L] = \begin{bmatrix} -0.060688 \\ 3.6483 \\ 3.6483 \end{bmatrix}$$

The solution of the discrete system reads

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \end{bmatrix} = [K]^{-1}[L] = \begin{bmatrix} 3.8219 \\ 2.9705 \\ 2.9705 \end{bmatrix}$$

Note that T_1 is the temperature at the center of the circle. A graphical representation of the solution is shown in Figure 11.4. On the right we have a genuine smooth surface that corresponds to the representation of temperature delivered by the T6 element; on the left we have the usual graphic that finite element programs use for quadratic elements– for greater efficiency the curved elements are shown as collections of flat polygons. However, this is only the graphical representation, the calculation reflects all the properties of the T6 elements correctly. The high accuracy of the single

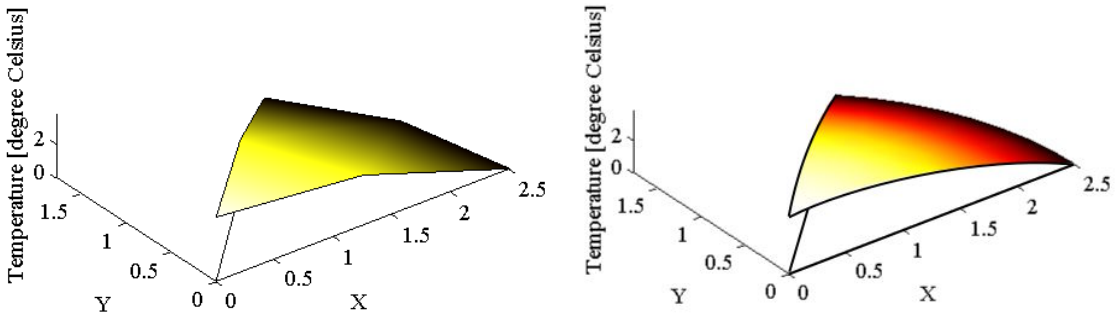


Fig. 11.4. Circular domain of the cross-section of a concrete column immersed in water, quadratic triangle (T6) model

T6 finite element mesh is manifest here. Comparable accuracy is obtained with 16 T3 triangles, and the reference solution at the center computed with 4,500 T3 elements 3.906^{o}C differs by only about two percent.

11.2 Quadratic 1-D element L3

The L3 element is developed from the L2 element by adding one more node inside the element. Therefore in order for the basis functions to satisfy the condition that they are either zero or one at a node, they will have to become quadratic polynomials. Nevertheless, the basis functions for this element are still the Lagrange interpolation functions on the standard interval, as shown in

Figure 11.5. The quadratic Lagrange interpolation polynomial is developed as explained earlier: for instance, N_1 is sought as a product of two linear polynomials which each vanish at the location of the remaining nodes

$$(\xi - 0)(\xi - 1) \, .$$

This expression is then normalized to become $+1$ at $\xi = -1$. Thus the basis functions N_1, N_2, and N_3 on the standard interval read

$$N_1(\xi) = \frac{\xi(\xi - 1)}{2} \, , \quad N_2(\xi) = \frac{\xi(\xi + 1)}{2} \, , \quad N_3(\xi) = (1 - \xi^2) \, . \tag{11.4}$$

The L3 element can use all the machinery developed for the isoparametric elements earlier. We will demonstrate it on the heat conduction problem with examples.

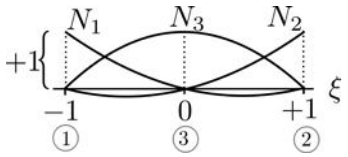


Fig. 11.5. Quadratic basis functions on the standard interval

---

Exercise 51.

Compute the elementwise conductivity matrix for the quadratic L3 finite element. Assume uniform thermal conductivity.

Solution: The conductivity matrix results from the weighted residual equation (8.14) after the substitution of the test and trial functions. In this way we get the expression

$$K_{ji} = S \int_0^L N'_{\langle j \rangle} \kappa_{xx} N'_{\langle i \rangle} \, \mathrm{d}x$$

For the mnemonic mesh of a single L3 finite element with nodes KLM we can compute the elementwise conductivity 3×3 matrix using the matrix expression

$$[K^{(e)}] = S \int_0^L [\mathrm{grad}N] \kappa_{xx} [\mathrm{grad}N]^T \, \mathrm{d}x$$

where

$$[\mathrm{grad}N] = \left[\mathrm{grad}_{(x)} N \right] = \begin{bmatrix} \mathrm{grad}_{(x)} N_K \\ \mathrm{grad}_{(x)} N_L \\ \mathrm{grad}_{(x)} N_M \end{bmatrix} = \begin{bmatrix} \dfrac{\partial N_K}{\partial x} \\ \dfrac{\partial N_L}{\partial x} \\ \dfrac{\partial N_M}{\partial x} \end{bmatrix} \, .$$

With numerical quadrature, which in this case is highly advisable given how complex an analytical solution can get, the conductivity matrix will be written as

$$[K^{(e)}] = \sum_{k=1}^{M} S[\text{grad}N(\xi_k)]\kappa_{xx}[\text{grad}N(\xi_k)]^T J(\xi_k) W_k$$

where we indicate that $[\text{grad}N(\xi_k)]$, which needs to be computed from (8.53), depend on the location of the quadrature point ξ_k.

The definition of the gradients of the basis functions with respect to the parametric coordinates (8.51) reduces to

$$\left[\text{grad}_{(\xi)}N\right] = \begin{bmatrix} \text{grad}_{(\xi)}N_K \\ \text{grad}_{(\xi)}N_L \\ \text{grad}_{(\xi)}N_M \end{bmatrix} = \begin{bmatrix} \xi - 1/2 \\ \xi + 1/2 \\ -2\xi \end{bmatrix} .$$

Importantly, at variance with the L2 and the T3 elements, the above gradients depend on the parametric coordinate. Therefore, they will change from integration point to integration point. The same holds for the Jacobian matrix: The Jacobian matrix is for this one-coordinate model defined as (8.50)

$$[J] = \left[\sum_{i=1}^{3} \frac{\partial N_i}{\partial \xi} x_i \right] .$$

It will also depend on the location of the quadrature point.

To continue we will make the following assumption. We shall take the locations of the end nodes as variables x_K, x_L and the location of the mid-element node will be taking exactly in the *middle of the element* $x_M = (x_K + x_L)/2$.

Here are a few lines of Matlab code to provide to students with a practical recipe: First we define five symbolic variables, the locations of the end nodes, the parametric coordinate ξ , the cross-section S, and the material conductivity κ

```
syms xK xL  xi S kappa real
x=[xK;xL;(xK+xL)/2];
```

We define the matrix `N` of basis function values, and the gradient of the basis functions with respect to the parametric coordinate

```
N=[xi*(xi-1)/2;xi*(xi+1)/2;(1-xi^2)]
gradNxi=[diff(N,'xi')]
N =
 (xi*(xi - 1))/2
 (xi*(xi + 1))/2
        1 - xi^2
gradNxi =
 xi - 1/2
 xi + 1/2
    -2*xi

>> J=x'*gradNxi
J =
xK*(xi - 1/2) + xL*(xi + 1/2) - 2*xi*(xK/2 + xL/2)

>> J=collect(simplify(J),xi)
J =
xL/2 - xK/2

>> gradN=gradNxi*inv(J)
gradN =
```

```
 -(xi - 1/2)/(xK/2 - xL/2)
 -(xi + 1/2)/(xK/2 - xL/2)
      (2*xi)/(xK/2 - xL/2)
```

Now we are ready for the numerical integration. We initialize the matrix to be a 3×3 zero matrix. The integrand is quadratic expression in ξ. The Gauss two-point quadrature is sufficiently accurate to compute such an integral exactly, viz Table 4.3.

Note that we need to substitute the value of the location of the quadrature point for the variable `xi` into the expression `kappa*gradN*gradN'*det(J)`, and that is achieved by calling the `subs` function. Here is initialization and the contribution from the first quadrature point:

```
>> K=zeros(3,3);
xi=-0.577350269189626; W=1; % location and weight of quadrature point
K =K+subs(S*kappa*gradN*gradN'*det(J))*W;
```

And the contribution of the second quadrature point is added in:

```
xi=0.577350269189626; W=1; % location and weight of quadrature point
K =K+subs(S*kappa*gradN*gradN'*det(J))*W;
```

Finally, the resulting expression is quite messy and the `simplify` function cleans it up somewhat,

```
K=simplify(K)
K =
[ -(7*S*kappa)/(3*(xK-xL)),   -(S*kappa)/(3*(xK-xL)),   (8*S*kappa)/(3*(xK-xL))]
[   -(S*kappa)/(3*(xK-xL)), -(7*S*kappa)/(3*(xK-xL)),   (8*S*kappa)/(3*(xK-xL))]
[  (8*S*kappa)/(3*(xK-xL)),  (8*S*kappa)/(3*(xK-xL)), -(16*S*kappa)/(3*(xK-xL))]
```

but we can do even better by hand: The L3 finite element elementwise conductivity for the configuration with the interior node at the average location of the end nodes reads

$$[K^{(e)}] = \frac{S\kappa}{h} \begin{bmatrix} 7/3, & 1/3, & -8/3 \\ 1/3, & 7/3, & -8/3 \\ -8/3, & -8/3, & 16/3 \end{bmatrix} \tag{11.5}$$

where $h = x_L - x_K$ is the length of the element.

Exercise 52.

Modify the assignment of Exercise 50 by including Newton's boundary condition at the circumference. The ambient temperature is $T_a = -1.5^o$ C, and the surface heat transfer coefficient is $h_o = 5.2\text{W/m}^2/^o\text{K}$.

Solution: The finite element mesh is the same as in Exercise 50, except that all nodes carry free degrees of freedom, and we take the numbers of the degrees of freedom to be equal to the numbers of the nodes. The elementwise conductivity matrix and the source heat load vector were computed as (11.2) and (11.3).

The outer circular boundary will be discretized with a single L3 finite element, with nodes 2, 3, 5. We can start with the elementwise ambient-temperature heat load vector. The formula

$$L_{q3,j} = \int_{\boldsymbol{x}_1}^{\boldsymbol{x}_2} N_j \; hT_a \; \Delta z \; \mathrm{d}C \; , \quad j = 1, 2, 3 \; .$$

simplifies for h, Δz and T_a constant with respect to the integration to

$$L_{q3,j} = hT_a \; \Delta z \int_{\boldsymbol{x}_1}^{\boldsymbol{x}_2} N_j \; \mathrm{d}C \; , \quad j = 1, 2, 3 \; .$$

Using a change of variables, we can write

$$L_{q3,j} = hT_a \Delta z \int_{-1}^{+1} N_j J \, \mathrm{d}\xi \ , \quad j = 1, 2, 3 \ .$$

where $J = \left|\left| \frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} \right|\right|$ is the Jacobian of the map $\boldsymbol{p}(\xi) = \sum_{i=1}^{3} N_i(\xi) \boldsymbol{x}_i$. The basis function values and the gradients of the basis functions with respect to ξ are computed as matrices by the two anonymous functions

```
N=@(xi)([xi*(xi-1)/2;xi*(xi+1)/2;(1-xi^2)]);
gradNxi =@(xi)([  xi - 1/2; xi + 1/2;    -2*xi]);
```

With the initialization

```
Ta=-1.5; ho= 5.2; dz=1.0;
x= [2.500000000000000                   0
    1.767766952966369   1.767766952966369
    2.309698831278217   0.956708580912724];
F=zeros(3,1);
```

we are ready to evaluate numerically the integrals of the basis functions along the length of the element. The expression

```
gradNxi(xi_k)'*x
```

evaluates the tangent vector to the curve of the element. For instance, for $\xi = -1$ (i.e. at the node 2) we get

```
>> gradNxi(-1)'*x
ans =
   -0.0145    1.0295
```

which is almost (but not exactly) tangent to the boundary circle. The reason is that the circle is only approximated by the quadratic curve that passes through the nodes 2, 5, 3. The length of the circular arc between the nodes $2, 3$ is

$$(45/180)\pi R = 1.9635$$

which is almost double the Jacobian

```
>> J=norm(gradNxi(-1)'*x)
J =
    1.0296
```

precisely as expected (for a straight line, with the third node in the middle, the Jacobian would be one half the length).

The numerical integration is based on Gauss two-point rule, and each of the expressions is practically a literal transcription of the formula:

```
xi_k=-0.577350269189626; W_k=1; % location and weight of quadrature point
J=norm(gradNxi(xi_k)'*x);
F =F+Ta*ho*dz*N(xi_k)*J*W_k;
xi_k=0.577350269189626; W_k=1; % location and weight of quadrature point
J=norm(gradNxi(xi_k)'*x);
F =F+Ta*ho*dz*N(xi_k)*J*W_k;
```

The resulting heat load vector due to ambient temperature is

$$[L^{(e)}] = \begin{bmatrix} -2.5522 \\ -2.5522 \\ -10.209 \end{bmatrix}$$

We continue next with the elementwise surface heat transfer matrix. The formula

$$H_{ji} = \int_{\boldsymbol{x}_1}^{\boldsymbol{x}_2} N_i \; hN_j \; \Delta z \; \mathrm{d}C \; , \quad j = 1, 2, 3 \; ,$$

simplifies for h, Δz constant with respect to the integration to

$$H_{ji} = h\Delta z \int_{\boldsymbol{x}_1}^{\boldsymbol{x}_2} N_i \; N_j \; \mathrm{d}C \; , \quad j = 1, 2, 3 \; .$$

Using a change of variables, we can write

$$H_{ji} = h\Delta z \int_{-1}^{+1} N_i \; N_j J \; \mathrm{d}\xi \; , \quad j = 1, 2, 3 \; .$$

The anonymous functions defined above are still needed, as is the initialization

```
ho= 5.2; dz=1.0;
H=zeros(3,3);
```

We again use the Gauss two-point rule. Note that

```
>> (N(xi_k)*N(xi_k)')
ans =
    0.2073   -0.0556    0.3036
   -0.0556    0.0149   -0.0813
    0.3036   -0.0813    0.4444
```

is a square matrix.

```
xi_k=-0.577350269189626; W_k=1; % location and weight of quadrature point
J=norm(gradNxi(xi_k)'*x);
H=H+ho*dz*(N(xi_k)*N(xi_k)')*J*W_k;
xi_k=0.577350269189626; W_k=1; % location and weight of quadrature point
J=norm(gradNxi(xi_k)'*x);
H=H+ho*dz*(N(xi_k)*N(xi_k)')*J*W_k;
```

The resulting surface heat transfer matrix of the L3 element is

$$[H^{(e)}] = \begin{bmatrix} 1.1343, & -0.56716, & 1.1343 \\ -0.56716, & 1.1343, & 1.1343 \\ 1.1343, & 1.1343, & 4.5373 \end{bmatrix}$$

The global surface heat transfer matrix is assembled as

$$[H] = \begin{bmatrix} 0, & 0, & 0, & -0, & -0, & -0 \\ 0, & 1.1343, & -0.56716, & -0, & 1.1343, & -0 \\ 0, & -0.56716, & 1.1343, & -0, & 1.1343, & -0 \\ -0, & -0, & -0, & 0, & -0, & -0 \\ -0, & 1.1343, & 1.1343, & -0, & 4.5373, & -0 \\ -0, & -0, & -0, & -0, & -0, & 0 \end{bmatrix}$$

and the sum of the global conductivity and surface heat transfer matrices reads

$$[K] + [H] = \begin{bmatrix} 0.7049, & 0.1557, & 0.1557, & -0.4637, & -0.0889, & -0.4637 \\ 0.1557, & 2.6963, & -0.2085, & -0.6277, & -0.1675, & -0.1468 \\ 0.1557, & -0.2085, & 2.6963, & -0.1468, & -0.1675, & -0.6277 \\ -0.4637, & -0.6277, & -0.1468, & 4.2903, & -0.5867, & -2.4655 \\ -0.0889, & -0.1675, & -0.1675, & -0.5867, & 8.4032, & -0.5867 \\ -0.4637, & -0.1468, & -0.6277, & -2.4655, & -0.5867, & 4.2903 \end{bmatrix}$$

The total heat load vector is assembled from the source heat loads and the ambient-temperature heat loads

$$[F] = \begin{bmatrix} -0.0607 \\ -2.5219 \\ -2.5219 \\ 3.6483 \\ -6.4695 \\ 3.6483 \end{bmatrix}$$

The nodal temperatures result as

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = ([K] + [H])^{-1}[F] = \begin{bmatrix} 3.4133 \\ -0.46255 \\ -0.46255 \\ 2.5426 \\ -0.39718 \\ 2.5426 \end{bmatrix} \quad \text{in } {}^{o}\text{C}$$

The solution is visualized in Figure 11.6.

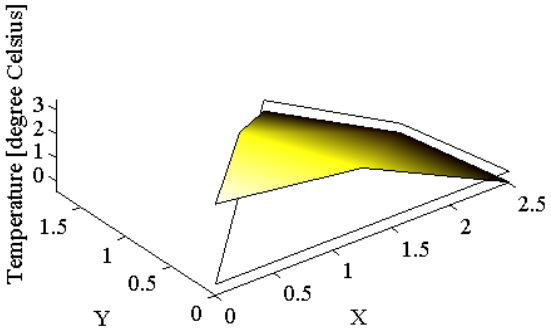


Fig. 11.6. Circular domain of the cross-section of a concrete column with Newton's boundary condition along the circumference, quadratic triangle (T6) model

---

Exercise 53.

Computes the elementwise capacity matrix for the quadratic L3 element. Assume uniform specific heat (homogeneous material). Consider a well-shaped element: the interior node is exactly in the middle of the element.

Solution: Imagine a mesh with a single element (see Exercise 51). The components of its capacity matrix read

$$C_{ji} = S \int_{L^{(e)}} N_{\langle j \rangle} c_V N_{\langle i \rangle} \, \mathrm{d}x$$

where S is the cross-sectional area, and $L^{(e)}$ is the length of the element. The capacity matrix is 3×3 and therefore all of its elements can be computed by evaluating the matrix integral

$$[C^{(e)}_{3\times 3}] = S \int_{L^{(e)}} c_V [N][N]^T \, \mathrm{d}x$$

where

$$[N(\xi)] = \begin{bmatrix} N_K(\xi) \\ N_L(\xi) \\ N_M(\xi) \end{bmatrix}$$

where (ξ) are the parametric coordinates at which the function value is taken.

The integral may be evaluated with three-point Gaussian quadrature rule. The reasoning behind this choice is the character of the integrand: each of the basis functions is quadratic in the parametric coordinate, and therefore their product is quartic. Even when the Jacobian is constant (as it is when the interior node is exactly in the middle of the element), the integrand will be fourth-order in x, but the selected integration rule can handle quintic polynomials.

The (constant) Jacobian was computed in Exercise 51:

$$\det[J(\xi)] = \frac{x_L - x_K}{2}$$

and so we can evaluate

$$\underset{3\times3}{[C^{(e)}]} = S \sum_{k=1}^{3} c_V [N(\xi_k)][N(\xi_k)]^T \det[J(\xi_k)] W_k$$

using the coordinates and weights of Table 4.3

```
>> xi=-0.774596669241483;
C1 =subs(N*N'*det(J))*0.555555555555556;
xi=0.774596669241483;
C2 =subs(N*N'*det(J))*0.555555555555556;
xi=0;
C3 =subs(N*N'*det(J))*0.888888888888889;
C=simplify(C1+C2+C3)
C =
[ (2*xL)/15 - (2*xK)/15,         xK/30 - xL/30,         xL/15 - xK/15]
[         xK/30 - xL/30, (2*xL)/15 - (2*xK)/15,         xL/15 - xK/15]
[         xL/15 - xK/15,         xL/15 - xK/15, (8*xL)/15 - (8*xK)/15]
```

which can be simplified by extracting the common factor $(x_L - x_K)$ to derive the consistent capacity matrix of the L3 finite element KLM as

$$[C^{(e)}] = \frac{(x_L - x_K) c_V S}{15} \begin{bmatrix} 2, & -\frac{1}{2}, & 1 \\ -\frac{1}{2}, & 2, & 1 \\ 1, & 1, & 8 \end{bmatrix}$$

Exercise 54.

Computes the elementwise heat load vector for internal heat generation for the quadratic L3 finite talent. Assume uniform heat generation density Q.

Solution: Solution: Imagine a mesh with a single element (see Exercise 51). The components of the elementwise heat load vector for internal heat generation read

$$L_j = S \int_{L^{(e)}} N_{\langle j \rangle} Q \, \mathrm{d}x$$

where S is the cross-sectional area, and $L^{(e)}$ is the length of the element. The heat load vector matrix is 3×1 and therefore all of its components can be computed by evaluating the matrix integral

$$\underset{3\times1}{[L]} = S \int_{L^{(e)}} [N] Q \, \mathrm{d}x$$

where $[N(\xi)]$ was defined in Exercise 53.

The integral may be evaluated with two-point Gaussian quadrature rule because each of the basis functions is quadratic in the parametric coordinate, and the Jacobian is constant.

The Jacobian was computed in Exercise 51:

$$\det[J(\xi)] = \frac{x_L - x_K}{2}$$

and so we can evaluate

$$\underset{3\times 1}{[L]} = S \sum_{k=1}^{3} [N(\xi_k)] Q \det[J(\xi_k)] W_k$$

using the coordinates and weights of Table 4.3

```
>> syms xK xL xM xi real
N=[xi*(xi-1)/2;xi*(xi+1)/2;(1-xi^2)];
detJ=(xL-xK)/2;
xi=-0.577350269189626;
L1 =subs(N*detJ)*1;
xi=0.577350269189626;
L2 =subs(N*detJ)*1;
L=simplify(subs(L1+L2))
L =
         xL/6 - xK/6
         xL/6 - xK/6
 (2*xL)/3 - (2*xK)/3
```

which can be simplified by extracting the common factor $(x_L - x_K)$ to derive the internal-heat load vector of the L3 finite element KLM as

$$[L] = \frac{(x_L - x_K) Q S}{6} \begin{bmatrix} 1 \\ 1 \\ 4 \end{bmatrix}$$

11.3 Point element P1

Browsing the `classes/gcell` folder, one may notice the `gcell_X_manifold` class folders with `X`= 0, 1, 2, 3. All `FAESOR` geometric cells are of certain so-called ***manifold dimension***: solids are of manifold dimension 3, surfaces are of dimension 2, while curves and points are of dimensions 1 and 0. Since we commonly solve heat diffusion (and other problems) with functions that are defined in 3-D domains (solids), 2-D domains (surfaces), and 1-D domains (curves), we also have to deal with integration over the boundaries of these domains; these are, correspondingly, surfaces, curves, and points.

When the heat diffusion model was formulated in two-dimensional domains in Chapter 8, the discrete domain consisted of triangles (elements T3), and the discrete boundary consisted of line segments (elements L2). Analogously, when the heat diffusion is solved in a one-dimensional domain (interval of the real line) which is covered by elements L2, the boundary consists of two points: hence the need for elements of type P1.

Evaluating the integrals of the surface heat transfer matrix (8.42) and the surface heat transfer load (8.41) (and also the prescribed heat flux load (8.40)) over the boundary of an interval on the real line simply means taking the values of the integrands at the end points. In terms of a quadrature formula applied at the boundary point a (analogous to (4.6)),

$$f(a) \approx f(\xi_1) J(\xi_1) W_1 \ ,$$

which is going to give the expected results with $\xi_1 = 0$, $f(\xi_1) = f(a)$, $W_1 = 1$, and the Jacobian $J(\xi_1) = 1$. The quadrature rule with these properties is the `point_rule`, and a sample script to use this type of evaluation of boundary integrals for one-dimensional heat diffusion problems is `transcool`[1].

With the introduction of the element P1, a closure is achieved: the same code will now work for heat diffusion problems solved on one-dimensional, two-dimensional, and three-dimensional domains.

11.4 Integrating over m-dimensional domains

The uniform treatment of the manifold dimension of the domain allows us to produce dimension-independent code. Therefore, integration of any (typically, scalar) function over any domain or subdomain is carried out by a single method of the class `feblock`. Consider as an example the geometry of a cylinder, the volume tiled with tetrahedra, the bounding surface covered with triangles, the edges of the cylindrical faces approximated with straight two-node segments, and one node at each vertex of the mesh. We may integrate over the volume of the 3-D mesh to find an approximation of the volume of the original cylinder; or over the length of a single edge to approximate the circumference; or over the area of one circular face to find an approximation of the cross-sectional area; or to count all the nodes on the cylindrical surface when we integrate over all the vertices of the triangles on that surface.

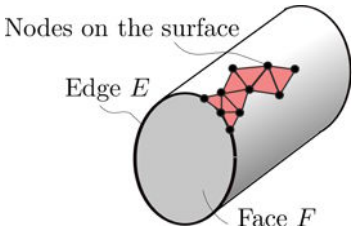


Fig. 11.7. Geometry of a cylinder.

The method `measure` of the class `feblock` evaluates the integral

$$\int_{V_m} f(\boldsymbol{x}) \, \mathrm{d}V_m \ , \tag{11.6}$$

where V_m is the volume of an m-dimensional manifold ($m = 0, 1, 2, 3$). There are a number of uses to which the method could be applied: as an example consider the calculation of the moments of inertia, or calculation of the centroid.

The method takes as arguments the geometry field (evidently, the volume of any discrete manifold is going to depend on the locations of its vertices), and a function handle.

```
0014 function result = measure[2] (self, geom, fh, varargin)
0015     gcells =self.gcells;
```

The simplest function that can be supplied is $f(\boldsymbol{x}) = +1$, which yields the volume of the manifold as the result of the integration. Otherwise, the function can supply for instance the location-dependent mass density when the body is inhomogeneous. The dimension of the manifold may be deduced from the manifold dimension of the geometric cells, or it could be supplied in the optional `varargin` argument.

[1] Folder: `FAESOR/examples/diffusion`
[2] Folder: `FAESOR/classes/feblock/@feblock`

```
0016     if nargin >=4
0017         m =varargin{1};% Either the manifold dimension was supplied
0018     else
0019         m= get(gcells(1),'dim');% ...Or it is implied
0020     end
```

Evaluate the integral by looping over all geometric cells. First step: precompute the basis function values and the gradients of the basis functions with respect to the parametric coordinates. Within the loop collect the geometry from the supplied field.

```
...
0032     result = 0;% Initialize the result
0033     % Now loop over all gcells in the block
0034     conns = get(gcells, 'conn'); % connectivity
0035     for i=1:ngcells
0036         conn =conns(i,:);
0037         x = gather(geom, conn, 'values', 'noreshape');
0038         % Loop over all integration points
0039         for j=1:npts_per_gcell
```

The method `Jacobian_mdim` is dispatched dynamically, to be treated differently in dependence on the dimension of the manifold. The result of `Jacobian_mdim` may depend on the location of the point in the parametric or spatial coordinates.

```
0040             J = Jacobian_matrix (gcells, Nders{j}, x);
0041             Jac =Jacobian_mdim(gcells, conn, Ns{j}, J, x, m);
```

The array of basis functions is computed so that the spatial location may be evaluated, and supplied to the function `fh`. The argument `N'*x` is the spatial location of the integration point (it is the interpolation of the locations of the nodes!). The result is accumulated with numerical quadrature.

```
0042             result = result + fh(Ns{j}'*x)*Jac*w(j);
0043         end
0044     end
0045 end
```

As an example, here is the `Jacobian_mdim` method for a two-dimensional manifold (a surface). This may be a real surface (a zero-thickness sheet); or it may be equipped with a thickness, or it could be revolved around an axis, and therefore be a representative of a 3-D volume. The argument `m` is used to distinguish between these possibilities.

```
0019 function Jac = Jacobian_mdim[3](self, conn, N, J, x, m)
0020     switch (m)
0021         case 3
0022             Jac = Jacobian_volume(self, conn, N, J, x);
0023         case 2
0024             Jac = Jacobian_surface(self, conn, N, J, x);
0025         otherwise
0026             error('Wrong dimension');
0027     end
0028 end
```

Let us assume for the moment it is a surface without a thickness (`m=2`), in which case the method `Jacobian_surface` of the class `gcellset_2_manifold` is invoked to do the work. The number of space dimensions `sdim` of the space in which the manifold is embedded could be 2 (the manifold is just a piece of the Euclidean plane), or 3 (the manifold is then a piece of a 3-D surface). The number of tangent vectors must be 2 (compare with (8.59), and refer to Fig. 11.8): they are

[3] Folder: `FAESOR/classes/gcellset/@gcellset_2_manifold`

$$\frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \xi}, \quad \text{and} \quad \frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \eta}.$$

For a surface that is part of a plane, the Jacobian is the determinant of the square Jacobian matrix (line 0016). For a surface that is embedded in a 3-D space, the Jacobian is the length (norm) of the cross product vector of the two tangents (refer to Fig. 8.19). Here, the cross product is expressed through a skew-symmetric matrix (line 0018).

```
0012 function Jac = Jacobian_surface[4](self, conn, N, J, x)
0013     [sdim, ntan] = size(J);
0014     if     ntan==2 % 2-D gcell
0015         if sdim==ntan
0016             Jac = J(1,1)*J(2,2)-J(2,1)*J(1,2);% this is det(J);% Compute the Jacobian
0017         else
0018             Jac = norm(skewmat(J(:,1))*J(:,2));
0019         end
0020     else
0021         error('Got an incorrect size of tangents');
0022     end
0023 end
```

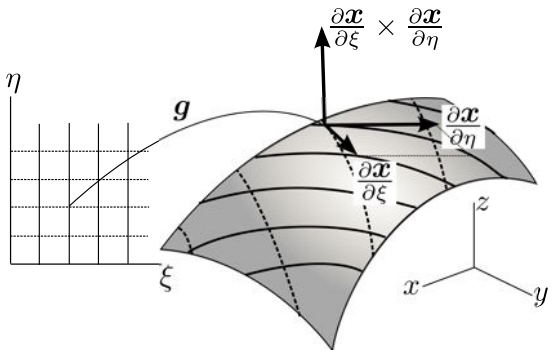


Fig. 11.8. Surface with the coordinate curves and tangents

Finally, we give an example of the use of the `measure` method: The Matlab script `test_measure`[5] computes the volume and the surface area of a rectangular block tiled with tetrahedra T4. Note that the boundary of the 3-D domain is extracted with the meshing function `mesh_bdry`

```
0001 a=2.5*pi;
0002 b=2.95;
0003 c=6.1313;
0004 [fens,gcells] = t4block(a,b,c, 5, 4, 7);
0005 bg=mesh_bdry(gcells);
0006 geom=field(struct('name',['geom'],'dim',3,'fens',fens));
0007 feb = feblock (struct ('mater',[], 'gcells',gcells,...
0008     'integration_rule', tet_rule (1)));
0009 disp([' The volume is = '...
0010     num2str(measure(feb,geom,inline('1'))) ...
0011     ', to be compared with ' num2str(a*b*c)])
0012 feb = feblock (struct ('mater',[], 'gcells',bg,...
0013     'integration_rule', tri_rule (1)));
```

[4] Folder: `FAESOR/classes/gcellset/@gcellset_2_manifold`
[5] Folder: `FAESOR/examples/miscellaneous`

```
0014 disp([' The surface is = ' ...
0015     num2str(measure(feb,geom,inline('1'))) ...
0016     ', to be compared with ' num2str(2*(a*b+a*c+b*c))])
```

It might seem tempting to evaluate all the objects used in the computational methods of this book, the conductivity matrix, the mass matrix, the load terms, and so on, with a generalization of the `measure` method. Unfortunately, Matlab passes arguments by value, which means that to accumulate as the result, for instance, a 20000×20000 stiffness matrix would be prohibitively expensive and wasteful. (The correct result would be produced, if that's any consolation.)

11.5 Tetrahedron T4

The tetrahedron with four nodes at the corners (element T4) is a straightforward extension of the triangle T3. The standard tetrahedron is shown in Fig. 11.9. The basis functions in the parametric coordinates are designed to be linear functions of ξ, η, ζ, and there are four corners at which to use the Kronecker delta property. It is straightforward to deduce that

$$N_1(\xi,\eta,\zeta) = 1 - \xi - \eta - \zeta \ , \ N_2(\xi,\eta,\zeta) = \xi \ , \ N_3(\xi,\eta,\zeta) = \eta \ , \ N_4(\xi,\eta,\zeta) = \zeta \ . \tag{11.7}$$

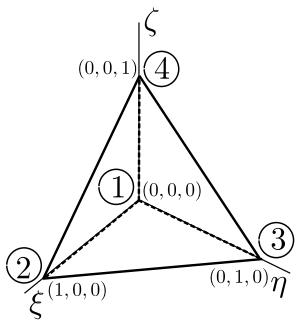


Fig. 11.9. Standard tetrahedron

Table 11.2 defines two integration rules for tetrahedra [H00]. The one-point rule is adequate for conductivity matrix evaluation, while the four-point rule could handle the capacity matrix terms.

Table 11.2. Numerical integration rules on the standard tetrahedron; $a = 0.1381966$, $b = 0.5854102$.

Rule	Coordinates ξ_j, η_j, ζ_j	Weights W_j	Integrates exactly
1-point	1/4, 1/4, 1/4	1/6	linear polynomial
4-point	a, a, a	1/24	quadratic polynomial
	b, a, a	1/24	
	a, b, a	1/24	
	a, a, b	1/24	

The four basis functions of the tetrahedron each vanish along the opposite face (basis function N_i on the face opposite node i and so on). The remaining three vary along this face exactly as if it was a triangle T3. The situation is entirely analogous to the one discussed in Section 8.13 for the triangle T3 and the line segment L2. Therefore, evaluation of the surface heat transfer contributions for a mesh of T4 volume elements is performed by extracting the faces of the tetrahedra as the geometric cells of type T3, and integrating over those cells.

Example: The script `helixcooled`[6] illustrates a solution with a full 3-D geometry discretized with the T4 tetrahedra. The problem is to determine steady state surface temperature for a helical spring with variable cross-section– see Fig. 11.10. The thick end is maintained at constant temperature, and on the rest of the surface we assume convection cooling.

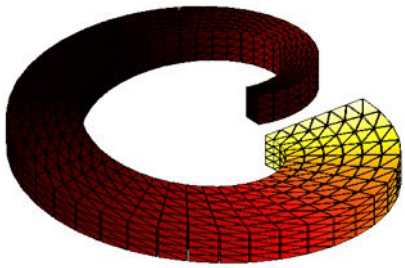


Fig. 11.10. The cooling of a helical spring.

The mesh is a simple regular block tiled with tetrahedra, but it is then shaped by moving nodes to different locations using the meshing function `transform_apply`, first by changing its cross-section, and then by shifting all nodes in the y-direction. Finally, the shape is twisted into a helix using `transform_2_helix`.

```
0008 [fens,gcells] = t4block(Angle,Width,Height, 50, 6, 4);
0009 Radius = 1.2;
0010 fens=transform_apply(fens,...
                    @(x,data)(x.*[1,(1-x(1)/Angle/1.2),1]),[]);
0011 fens=transform_apply(fens,@(x,data)(x+ [0,Radius,0]),[]);
0012 climbPerRevolution= 1.3;
0013 fens = transform_2_helix(fens,climbPerRevolution);
```

The surface mesh consists of triangles T3, and is extracted from the tetrahedral mesh using the meshing function `mesh_bdry`. The surface mesh is drawn with the `drawmesh` utility.

```
0014 bgcells=mesh_bdry(gcells);
0015 drawmesh({fens,bgcells},'gcells','facecolor','red')
```

Next, the finite element blocks for the tetrahedral elements in the volume and the triangular elements on the surface are created. Note that the two blocks use different quadrature rules, `tet_rule` for the tetrahedra, and `tri_rule` for the triangles; both use just one integration point.

```
0017 feb = feblock_diffusion (struct ('mater',mater,...
0018     'gcells',gcells,...
0019     'integration_rule',tet_rule(num_integ_pts)));
0020 bfeb = feblock_diffusion (struct ('mater',mater,...
0021     'gcells',bgcells,...
0022     'integration_rule',tri_rule(num_integ_pts),...
0023     'surface_transfer', h));
```

From this point on, the script does not depend on the element types, be it the calculation of the system matrices, or graphics output.

11.6 Simplex elements

The point P1, the segment L2, the triangle T3, and the tetrahedron T4, are all examples of the so-called simplex elements. By definition, an n-dimensional simplex is the convex hull of $n+1$ points

[6] Folder: `FAESOR/examples/diffusion`

(vertices) in the n-dimensional space. Tiling domains with simplex elements is attractive, because a number of mathematical properties guarantees the success of automatic tools for mesh generation. This is to be contrasted with the generation of quadrilaterals in two dimensions, and of bricks (shapes bounded by six quadrilateral faces) in three dimensions: not an easy task– mesh generators often fail to produce good-quality meshes, or often they just fail to produce any mesh.

While the simplex elements perform adequately in the heat conduction models, in other types of analyses their inherent simplicity tends to work against them. For instance, as we shall see in linear elasticity the response of meshes composed of simplex elements is quite poorly represented – they are "too stiff".

Exercise 55. Generalize the calculation of the shape functions from the expression (8.28) to apply to the four-node tetrahedron and to the two-node line segment.

Solution: The basis functions for the T4 tetrahedron $KLMP$ are linear functions with four coefficients, for instance

$$N_K(x,y,z) = a_K x + b_K y + c_K z + d_K$$

The interpolation conditions are written analogously to (8.28)

$$N_K(x_J,y_J,z_J) = a_K x_J + b_K y_J + c_K z_J + d_K = \delta_{KJ} \ ,$$

which results in

$$\underbrace{\begin{bmatrix} x_K,\ y_K,\ z_K,\ 1 \\ x_L,\ y_L,\ z_L,\ 1 \\ x_M,\ y_M,\ z_M,\ 1 \\ x_P,\ y_P,\ z_P,\ 1 \end{bmatrix}}_{\boldsymbol{X}} \underbrace{\begin{bmatrix} a_K,\ a_L,\ a_M,\ a_P \\ b_K,\ b_L,\ b_M,\ b_P \\ c_K,\ c_L,\ c_M,\ c_P \\ d_K,\ d_L,\ d_M,\ d_P \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} 1,\ 0,\ 0,\ 0 \\ 0,\ 1,\ 0,\ 0 \\ 0,\ 0,\ 1,\ 0 \\ 0,\ 0,\ 0,\ 1 \end{bmatrix}}_{\boldsymbol{1}}$$

Consequently, the coefficients of the basis functions result as

$$\boldsymbol{A} = \boldsymbol{X}^{-1}$$

and in particular the matrix of basis function gradients with respect to x, y, z are the first three columns of $\boldsymbol{A}^T$.

For the two-node L2 line segment element KL the basis functions are

$$N_K(x) = a_K x + b_K \ , \quad N_L(x) = a_L x + b_L \ .$$

The analogy of the interpolation conditions for the tetrahedron now read

$$\underbrace{\begin{bmatrix} x_K,\ 1 \\ x_L,\ 1 \end{bmatrix}}_{\boldsymbol{X}} \underbrace{\begin{bmatrix} a_K,\ a_L \\ b_K,\ b_L \end{bmatrix}}_{\boldsymbol{A}} = \underbrace{\begin{bmatrix} 1,\ 0 \\ 0,\ 1 \end{bmatrix}}_{\boldsymbol{1}}$$

Consequently, the coefficients of the basis functions result as

$$\boldsymbol{A} = \boldsymbol{X}^{-1} = \frac{1}{x_L - x_K} \begin{bmatrix} -1, & +1 \\ x_L, & -x_K \end{bmatrix} \ .$$

The basis functions are therefore seen to be

$$N_K(x) = \frac{-1}{x_L - x_K} x + \frac{x_L}{x_L - x_K} = \frac{x_L - x}{x_L - x_K} \ , \quad N_L(x) = \frac{1}{x_L - x_K} x + \frac{-x_K}{x_L - x_K} = \frac{x - x_K}{x_L - x_K} \ .$$

The gradients of the basis functions are the first column of the $\boldsymbol{A}^T$

$$\text{grad}N_K = \frac{-1}{x_L - x_K}\ , \quad \text{grad}N_L(x) = \frac{1}{x_L - x_K}$$

The same reasoning applies to the final simplex in the family: the point P0. The basis function for the point element is

$$N_K(x) = a_K$$

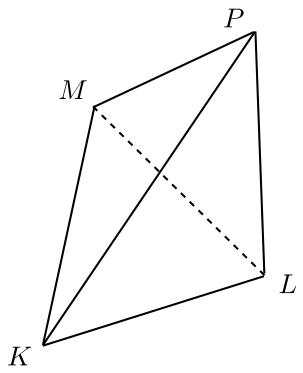
and the interpolation condition is

$$N_K(x_K) = 1$$

Hence it follows that $N_K(x) = 1$ for the point P0.

Exercise 56.

Compute the Jacobian of the tetrahedron T4 finite element with nodes K, L, M and P. Establish the relationship of the Jacobian to the volume of the parallelepiped that shares edges with the tetrahedron. Solution:



The coordinates of the tetrahedron nodes are arranged in a matrix as

$$[\mathtt{x}] = \begin{bmatrix} x_K\ ,\ y_K\ ,\ z_K \\ x_L\ ,\ y_L\ ,\ z_L \\ x_M\ ,\ y_M\ ,\ z_M \\ x_P\ ,\ y_P\ ,\ z_P \end{bmatrix}\ ,$$

similarly as for the triangle in (8.55). The Jacobian matrix is computed from the general formula (8.54) where the matrix of basis function gradients with respect to the parametric coordinates is

$$[\mathtt{Nder}] = \begin{bmatrix} \dfrac{\partial N_K}{\partial \xi}\ ,\ \dfrac{\partial N_K}{\partial \eta}\ ,\ \dfrac{\partial N_K}{\partial \zeta} \\ \dfrac{\partial N_L}{\partial \xi}\ ,\ \dfrac{\partial N_L}{\partial \eta}\ ,\ \dfrac{\partial N_L}{\partial \zeta} \\ \dfrac{\partial N_M}{\partial \xi}\ ,\ \dfrac{\partial N_M}{\partial \eta}\ ,\ \dfrac{\partial N_M}{\partial \zeta} \\ \dfrac{\partial N_P}{\partial \xi}\ ,\ \dfrac{\partial N_P}{\partial \eta}\ ,\ \dfrac{\partial N_P}{\partial \zeta} \end{bmatrix} = \begin{bmatrix} -1\ ,\ -1\ ,\ -1 \\ 1\ ,\ 0\ ,\ 0 \\ 0\ ,\ 1\ ,\ 0 \\ 0\ ,\ 0\ ,\ 1 \end{bmatrix}\ .$$

The matrix product yields the (constant) Jacobian matrix

$$[J] = \begin{bmatrix} x_L - x_K \ , x_M - x_K \ , x_P - x_K \\ y_L - y_K \ , \ y_M - y_K \ , \ y_P - y_K \\ z_L - z_K \ , \ z_M - z_K \ , \ z_P - z_K \end{bmatrix}$$

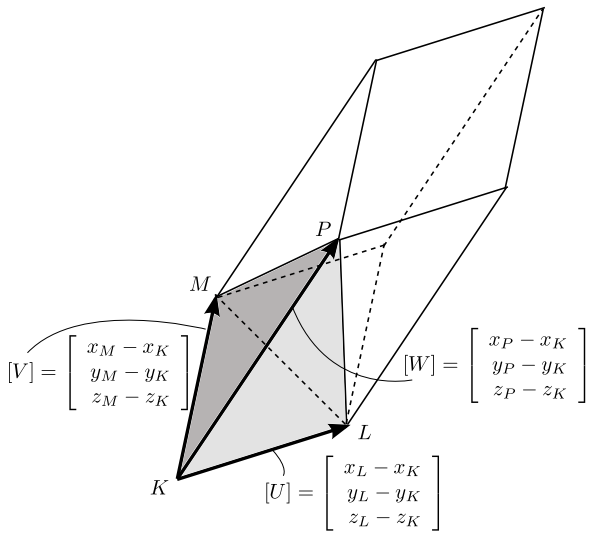
The Jacobian is the determinant of this matrix, and the determinant has an illuminating geometrical meaning. First rewrite the Jacobian matrix as

$$[J] = \big[\, [U] \ , [V] \ , [W] \,\big]$$

where the three vectors that constitute the columns of the Jacobian matrix are defined as the differences between the locations of the nodes

$$[U] = \begin{bmatrix} x_L - x_K \\ y_L - y_K \\ z_L - z_K \end{bmatrix}, \quad [V] = \begin{bmatrix} x_M - x_K \\ y_M - y_K \\ z_M - z_K \end{bmatrix}, \quad [W] = \begin{bmatrix} x_P - x_K \\ y_P - y_K \\ z_P - z_K \end{bmatrix}$$

It is easily demonstrated by computation that the volume of the parallelepiped is the triple vector



product

$$V_{pp} = [U] \cdot [V] \times [W]$$

which is also the determinant of the Jacobian matrix. Hence, we have the relationship between the Jacobian of the tetrahedron and the volume of the parallelepiped generated by the three edges of the tetrahedron that meet at the vertex K

$$\det[J] = V_{pp}$$

The volume of the parallelepiped (and the Jacobian) do not depend on the numbering of the tetrahedron. Any triple of connected edges of the tetrahedron would give the same result. The triple vector product is a signed quantity however, and we have to consider the numbering of the nodes: for example these numberings will give positive volume–$KLMP$, $LPMK$, $MPKL$, $PLKM$. The rule is that the three vectors must form a right-handed triple, similar to the basis vectors of the right-handed Cartesian coordinate system.

Finally, elementary geometry considerations lead to the conclusion that the volume of the tetrahedron is $V_{tet} = V_{pp}/6$ which also allows us to write

$V_{tet} = \det[J]/6$.

11.7 Quadrilateral Q4

Quadrilateral elements address the excessive stiffness of simplex elements by coupling together a larger number of nodes, which in the end leads to basis functions which are more than just linear polynomials.

The element Q4 has four nodes, and its standard shape is a square. This square is to be understood as the Cartesian product of two standard intervals (Fig. 8.25). Therefore, the basis functions of Q4 may also be formed as products of the basis functions on the standard interval L2. Assuming the numbering of the nodes as shown in Fig. 11.11, the basis function N_1 may be written as the product of the basis function on the interval $-1 \le \xi \le +1$ and the basis function on the interval $-1 \le \eta \le +1$ where both functions correspond to the left-hand side endpoint ($\xi = -1,\ \eta = -1$)

$$N_1(\xi,\eta) = \frac{\xi - 1}{-1 - 1} \times \frac{\eta - 1}{-1 - 1} = \frac{(\xi - 1)(\eta - 1)}{4} \ . \tag{11.8}$$

Similarly, for the remaining three functions we have

$$N_2(\xi,\eta) = \frac{(\xi + 1)(\eta - 1)}{-4} \ , \ N_3(\xi,\eta) = \frac{(\xi + 1)(\eta + 1)}{4} \ , \tag{11.9}$$

and

$$N_4(\xi,\eta) = \frac{(\xi - 1)(\eta + 1)}{-4} \ . \tag{11.10}$$

As all basis functions are linear in ξ and η, the shape that they represent when raised as a surface above the standard square is a hyperbolic paraboloid.

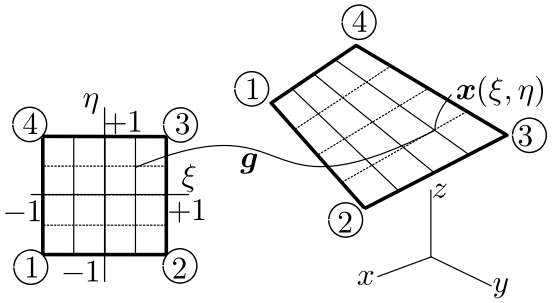


Fig. 11.11. Mapping the standard square to a general quadrilateral

Since the standard square is a Cartesian product of the standard intervals for which one-dimensional Gauss integration rules are a common choice, a two-dimensional Gauss integration rule is commonly adopted for Q4. It consists of a Cartesian product of one-dimensional Gauss rules. The class `gauss_rule` implements two-dimensional (and three-dimensional) rules which are products of one-dimensional tables. Thanks to the utility `gaussquad` by Peter J. Acklam (included with `FAESOR`), one-dimensional tables of any order may be calculated on demand and used for higher dimensions. For the four-node quadrilateral, a 2×2 Gauss quadrature is appropriate for conductivity matrices; a one-point rule is insufficient to build up the proper rank of the element matrices, while higher-order rules are a waste of time. The requirements for the minimum number of integration points from the point of view of stability (regularity of the stiffness matrices) are discussed in Section 18.2.

11.8 Hexahedron H8

To extend the quadrilateral to three dimensions is quite straightforward: instead of a Cartesian product of two intervals on the standard square, we consider the Cartesian product of three intervals on the standard cube (Fig. 11.12). This element is discussed in more detail later, in Section 15.5.

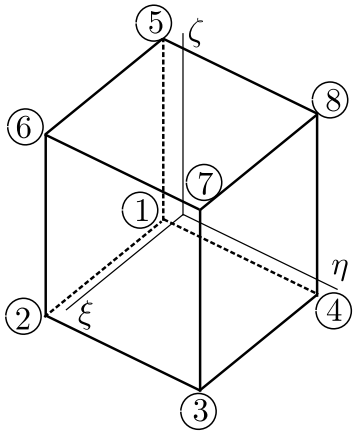


Fig. 11.12. Numbering of the nodes of the hexahedron H8. Node 1 is at $\xi = -1, \eta = -1, \zeta = -1$, node 2 is at $\xi = +1, \eta = -1, \zeta = -1$,..., and node 8 is at $\xi = -1, \eta = +1, \zeta = +1$

Exercise 57.

Extend the one-dimensional Gaussian integration rule for the standard interval to work for two-dimensional integration on the standard square.

Solution: See Figure 11.11 for the standard bi-unit square ($-1 \le \xi \le +1$ and $-1 \le \eta \le +1$). The integration over the square may be split into two one-dimensional integrations as

$$\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\xi\mathrm{d}\eta = \int_{-1}^{+1}\left(\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\eta\right)\mathrm{d}\xi$$

or,

$$\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\xi\mathrm{d}\eta = \int_{-1}^{+1}\left(\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\xi\right)\mathrm{d}\eta$$

The one-dimensional Gaussian rule may be applied to the outer integral

$$\int_{-1}^{+1}\left(\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\xi\right)\mathrm{d}\eta \approx \sum_{k=1}^{M}\left(\int_{-1}^{+1} f(\xi,\eta_k)\,\mathrm{d}\xi\right)W_k$$

Now each of the integrals $\int_{-1}^{+1} f(\xi,\eta_k)\,\mathrm{d}\xi$ may be evaluated with a numerical rule, possibly a different one from the one used above. Normally there is no reason to choose a different numerical rule, and therefore we can write

$$\int_{-1}^{+1} f(\xi,\eta_k)\,\mathrm{d}\xi \approx \sum_{j=1}^{M} f(\xi_j,\eta_k)W_j$$

Substituting of the latter approximation into the former yields

$$\int_{-1}^{+1}\left(\int_{-1}^{+1} f(\xi,\eta)\,\mathrm{d}\xi\right)\mathrm{d}\eta \approx \sum_{k=1}^{M}\sum_{j=1}^{M} f(\xi_j,\eta_k)W_jW_k$$

Two-and three-dimensional Gaussian rules may be written as one-dimensional tables, exactly as for one-dimensional Gaussian rules, by listing the quadrature points and their weights as follows: the coordinates of the quadrature points are composed as tensor products of the coordinates of the one-dimensional rules, and the corresponding weights are the products of the weights of the one-dimensional rules. For instance, one-point rule in one dimension

One-point one-dimensional rule		
j	Coordinates ξ_j	Weights W_j
1	0	2

yields a one-point rule in two dimensions,

One-point two-dimensional rule		
j	Coordinates ξ_j, η_j	Weights W_j
1	$0, 0$	$2 \times 2 = 4$

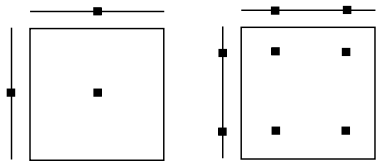
and two-point rule in one dimension

Two-point one-dimensional rule		
j	Coordinates ξ_j	Weights W_j
1	$-\sqrt{1/3}$	1
2	$+\sqrt{1/3}$	1

gives the four-point rule in two dimensions

Four-point two-dimensional rule		
j	Coordinates ξ_j, η_j	Weights W_j
1	$-\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
2	$-\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$
3	$+\sqrt{1/3}, -\sqrt{1/3}$	$1 \times 1 = 1$
4	$+\sqrt{1/3}, +\sqrt{1/3}$	$1 \times 1 = 1$

The figure below illustrates the above tables graphically. Higher-order Gaussian integration rules



would be derived entirely analogously.

Exercise 58.

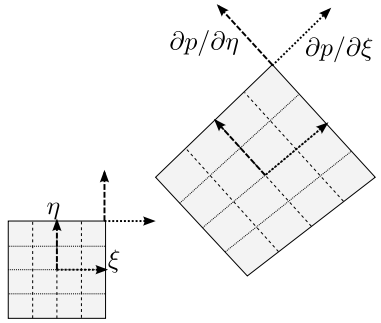
Illustrate the possibility of finding a negative Jacobian in severely distorted quadrilaterals.

Solution:

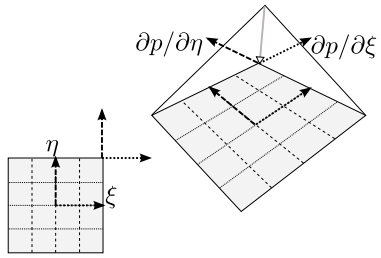
The Jacobian is the determinant of the Jacobian matrix (8.50). As discussed below equation (8.60), the Jacobian may be computed as the cross product of the two vectors

$$\det[J(\xi,\eta)] = \frac{\partial \boldsymbol{p}(\xi,\eta)}{\partial \xi} \times \frac{\partial \boldsymbol{p}(\xi,\eta)}{\partial \eta}$$

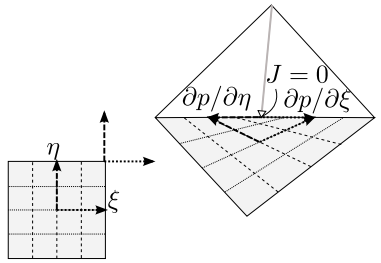
Note that $\partial \boldsymbol{p}(\xi,\eta)/\partial\xi$ is shown as dotted vector, and $\partial \boldsymbol{p}(\xi,\eta)/\partial\eta$ is displayed with a dashed vector. In the general quadrilateral these vectors change from point to point as shown in this figure. Also shown in the bottom left part of the figure is the standard shape (square) in the parametric coordinates.



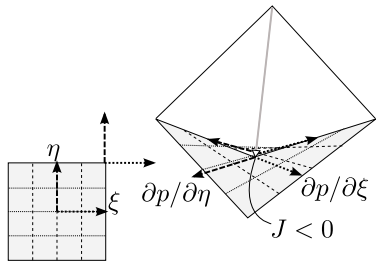
Since the two pairs of vectors shown in the figure are of slightly different lengths and subtend a slightly different angle, but Jacobian at the two indicated points will be different. Now imagine the topmost corner of the quadrilateral is moved towards the bottommost corner. (The original shape of the quadrilateral is shown for reference.) Clearly the shift caused the Jacobians to change, since both the lengths of the vectors and the angle between them changed.



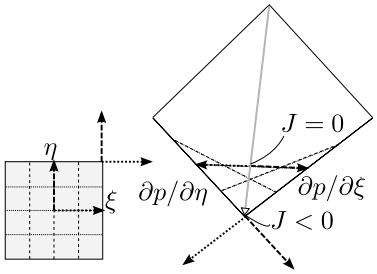
If the downward motion of the corner continues, eventually it will end up on the straight line that connects the leftmost and rightmost corners. At that point $\partial \boldsymbol{p}(\xi,\eta)/\partial\xi$ and $\partial \boldsymbol{p}(\xi,\eta)/\partial\eta$ become collinear, and the cross product by definition vanishes (becomes zero).



As the originally topmost corner is moved further down, the Jacobian at that corner it comes negative. At the center of the element the Jacobian is still positive, but considerably smaller in magnitude.



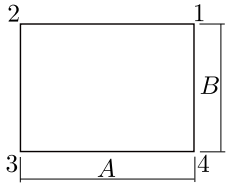
Finally, as the originally topmost corner coincides with the bottommost one the Jacobian at the center of the element becomes zero and the element is effectively folded over itself so that at each point in the physical coordinates where the Jacobian is positive in the element there is also another point in the element where the Jacobian is of the same magnitude but negative. The element effectively vanishes (it has zero area)!



Exercise 59.

Compute the elementwise conductivity matrix of a rectangular Q4 finite element using a 2×2 point Gauss quadrature rule. Assume homogeneous isotropic thermal conductivity. Assess the rank of the resulting conductivity matrix. Assume the nodes are located at

$$[p_1] = \left[\, A,\, B \,\right] \;, \quad [p_2] = \left[\, 0,\, B \,\right] \;, \quad [p_3] = \left[\, 0,\, 0 \,\right] \;, \quad [p_4] = \left[\, A,\, 0 \,\right] \;,$$



Solution: The conductivity matrix is computed from the general expression (8.37) for the mnemonic mesh shown above,

$$K_{ji} = \int_{S^{(e)}} (\mathrm{grad} N_{\langle j \rangle})\, \boldsymbol{\kappa} (\mathrm{grad} N_{\langle i \rangle})^T \, \Delta z \, \mathrm{d}S \;, \qquad i,j = 1, ..., 4 \;.$$

where $S^{(e)}$ is the area of the element. With the four-point Gauss rule (call for Exercise 57) this is approximated as

$$K_{ji} = \sum_{k=1}^{4} \mathrm{grad} N_{\langle j \rangle}(\xi_k, \eta_k)\, \boldsymbol{\kappa}\, \mathrm{grad} N^T_{\langle i \rangle}(\xi_k, \eta_k)\, \Delta z \;\; \det[J(\xi_k, \eta_k)] W_k \;, \qquad i,j = 1, ..., 4 \;.$$

where $\xi_k = \pm\sqrt{3}/3, \eta_k = \pm\sqrt{3}/3, W_k = 1$.

The basis functions (11.8)–(11.10) yield the matrix of gradients of the basis functions with respect to the parametric coordinates

```
>> syms xi eta real
N=[(xi-1)*(eta-1)/4;(xi+1)*(eta-1)/-4;(xi+1)*(eta+1)/4;(xi-1)*(eta+1)/-4]
Nder=[diff(N,'xi'),diff(N,'eta')]
N =
  ((eta - 1)*(xi - 1))/4
 -((eta - 1)*(xi + 1))/4
  ((eta + 1)*(xi + 1))/4
 -((eta + 1)*(xi - 1))/4
Nder =
[   eta/4 - 1/4,   xi/4 - 1/4]
[   1/4 - eta/4, - xi/4 - 1/4]
[   eta/4 + 1/4,   xi/4 + 1/4]
[ - eta/4 - 1/4,   1/4 - xi/4]
```

and we compute the Jacobian matrix and the Jacobian as

```
>> syms A B real
x=[A,B;0,B;0,0;A,0];
J=simple (x'*Nder)
J =
[ -A/2,    0]
[    0, -B/2]
>> syms A B real
x=[A,B;0,B;0,0;A,0];
J=simple (x'*Nder)
detJ=det(J)
J =
[ -A/2,    0]
[    0, -B/2]
detJ =
(A*B)/4
```

We may note that the Jacobian is constant, which is what we would expect since the standard square is mapped into a rectangle.

The basis function gradients with respect to x, y may be evaluated from the general formula (8.53)

```
>> Ndersp=Nder/J
Ndersp =
[ -(eta - 1)/(2*A), -(xi - 1)/(2*B)]
[  (eta - 1)/(2*A),  (xi + 1)/(2*B)]
[ -(eta + 1)/(2*A), -(xi + 1)/(2*B)]
[  (eta + 1)/(2*A),  (xi - 1)/(2*B)]
```

from which we glean that the gradients vary from point to point within the element.

The conductivity matrix can be now evaluated. First we simplify by noting that $\kappa, \Delta z, \det[J(\xi_k, \eta_k)] = (AB)/4$ and $W_k = 1$ are all constants and therefore may be taken out of the numerical quadrature sum. The four-point integration yields

```
>> xi=-0.577350269189626; eta=-0.577350269189626;
K1=subs(Ndersp)*subs(Ndersp)';
xi=-0.577350269189626; eta=0.577350269189626;
K2=subs(Ndersp)*subs(Ndersp)';
xi=0.577350269189626; eta=-0.577350269189626;
K3=subs(Ndersp)*subs(Ndersp)';
xi=0.577350269189626; eta=0.577350269189626;
K4=subs(Ndersp)*subs(Ndersp)';
K=simple (K1+K2+K3+K4);
```

with the intermediate result

$$\mathtt{K} = \begin{bmatrix} 4/(3A^2)+4/(3B^2), & 2/(3B^2)-4/(3A^2), & -2/(3A^2)-2/(3B^2), & 2/(3A^2)-4/(3B^2) \\ 2/(3B^2)-4/(3A^2), & 4/(3A^2)+4/(3B^2), & 2/(3A^2)-4/(3B^2), & -2/(3A^2)-2/(3B^2) \\ -2/(3A^2)-2/(3B^2), & 2/(3A^2)-4/(3B^2), & 4/(3A^2)+4/(3B^2), & 2/(3B^2)-4/(3A^2) \\ 2/(3A^2)-4/(3B^2), & -2/(3A^2)-2/(3B^2), & 2/(3B^2)-4/(3A^2), & 4/(3A^2)+4/(3B^2) \end{bmatrix}$$

so that

$$[K^{(e)}] = \frac{\kappa \Delta z \; AB}{4} \mathtt{K}$$

The Matlab symbolic toolbox can tell us the rank of the symbolic matrix `K`:

```
>> rank(K)
ans =
3
```

The rank of the conductivity matrix was discussed for the T3 (triangle) element in Exercise 38. It was discussed there that the expected rank would be three, and the present result is consistent.

The eigenvalue problem

$$[K^{(e)}][T^{(e)}] = \lambda[T^{(e)}]$$

or, rather

$$\frac{\kappa \Delta z\ AB}{4}\mathtt{K}[T^{(e)}] = \lambda[T^{(e)}] \quad \rightarrow \quad \mathtt{K}\,[T^{(e)}] = \frac{4}{\lambda}[T^{(e)}]\ ,$$

is solved by the symbolic Matlab toolkit to yield

```
>> [V,D]=eig(K)
V =
[ 1,  1, -1, -1]
[ 1, -1, -1,  1]
[ 1, -1,  1, -1]
[ 1,  1,  1,  1]
D =
[ 0,     0,     0,                            0]
[ 0, 4/A^2,     0,                            0]
[ 0,     0, 4/B^2,                            0]
[ 0,     0,     0, (4*A^2 + 4*B^2)/(3*A^2*B^2)]
```

We can see that the eigenvalue `D(1,1)` is zero, and the corresponding eigenvector in the first column of the matrix `V` represents a uniform temperature throughout the element.

Exercise 60.

Compute the elementwise conductivity matrix of a rectangular Q4 finite element using a one-point Gauss quadrature rule. Assume homogeneous isotropic thermal conductivity. Assess the rank of the resulting conductivity matrix. Refer to Exercise 59 for the single-element mesh.

Solution: The conductivity matrix is computed from the general expression (8.37) for the mnemonic mesh shown above,

$$K_{ji} = \int_{S^{(e)}} (\mathrm{grad}N_{\langle j\rangle})\ \boldsymbol{\kappa} (\mathrm{grad}N_{\langle i\rangle})^T\ \Delta z\ \mathrm{d}S\ , \qquad i,j = 1,...,4\ .$$

where $S^{(e)}$ is the area of the element. With the one-point Gauss rule (call for Exercise 57) this is approximated as

$$K_{ji} = \mathrm{grad}N_{\langle j\rangle}(\xi_1,\eta_1)\ \boldsymbol{\kappa}\ \mathrm{grad}N^T_{\langle i\rangle}(\xi_1,\eta_1)\ \Delta z\ \det[J(\xi_1,\eta_1)]W_1\ , \qquad i,j = 1,...,4\ .$$

where $\xi_1 = 0, \eta_1 = 0, W_1 = 4$. Note that the point $\xi_1 = 0, \eta_1 = 0$ maps to the geometrical center of the rectangular element. The basis functions (11.8)–(11.10) yield the matrix of gradients of the basis functions with respect to the parametric coordinates and we compute the Jacobian matrix and the Jacobian as discussed in Exercise 59.

The basis function gradients with respect to x, y may be evaluated from the general formula (8.53), but for the centroid of the rectangular element they are also easily computed from the elementary rise-over-run formula. For instance,

$$\frac{\partial N_1}{\partial x} = \frac{1/2}{A}, \quad \frac{\partial N_1}{\partial y} = \frac{1/2}{B}$$

and

$$\frac{\partial N_2}{\partial x} = \frac{-1/2}{A}, \quad \frac{\partial N_2}{\partial y} = \frac{1/2}{B} \ .$$

This is easily verified as we obtain

```
>> Ndersp=Nder/J
Ndersp =
[ -(eta - 1)/(2*A), -(xi - 1)/(2*B)]
[  (eta - 1)/(2*A),  (xi + 1)/(2*B)]
[ -(eta + 1)/(2*A), -(xi + 1)/(2*B)]
[  (eta + 1)/(2*A),  (xi - 1)/(2*B)]
```

from which we should take that the gradients vary from point to point within the element. At the center the gradients are

```
>> xi=0; eta=0;
subs(Ndersp)
ans =
[  1/(2*A),  1/(2*B)]
[ -1/(2*A),  1/(2*B)]
[ -1/(2*A), -1/(2*B)]
[  1/(2*A), -1/(2*B)]
```

in agreement with our elementary computations.

The conductivity matrix can be now evaluated. The one-point integration yields

```
xi=0; eta=0;
syms kappa Dz real
K=kappa*subs(Ndersp)*subs(Ndersp)'*Dz*detJ*4;
K1=simple (K/kappa/Dz*4)
K1 =
[   A/B + B/A,   A/B - B/A, - A/B - B/A,   B/A - A/B]
[   A/B - B/A,   A/B + B/A,   B/A - A/B, - A/B - B/A]
[ - A/B - B/A,   B/A - A/B,   A/B + B/A,   A/B - B/A]
[   B/A - A/B, - A/B - B/A,   A/B - B/A,   A/B + B/A]
```

so that

$$[K] = \frac{\kappa \Delta z}{4} \begin{bmatrix} A/B + B/A, & A/B - B/A, & -A/B - B/A, & B/A - A/B \\ A/B - B/A, & A/B + B/A, & B/A - A/B, & -A/B - B/A \\ -A/B - B/A, & B/A - A/B, & A/B + B/A, & A/B - B/A \\ B/A - A/B, & -A/B - B/A, & A/B - B/A, & A/B + B/A \end{bmatrix} = \frac{\kappa \Delta z}{4} \texttt{K1}$$

The Matlab symbolic toolbox can tell us the rank of the symbolic matrix `K1`:

```
>> rank(K1)
ans =
2
```

The rank of the conductivity matrix was discussed for the T3 (triangle) element in Exercise 38. It was discussed there that the expected rank here would be three, not two. In other words, we would expect one zero eigenvalue, but apparently there are two. A detailed analysis follows: We solve the eigenvalue problem

$$[K^{(e)}][T^{(e)}] = \lambda [T^{(e)}]$$

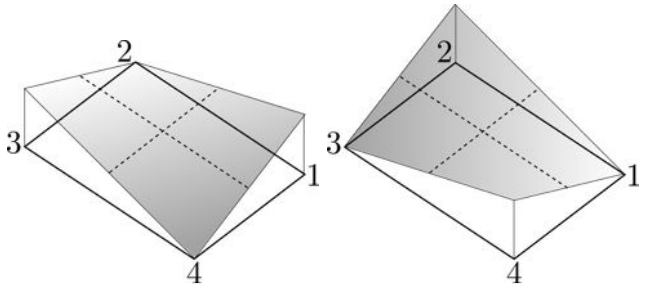
or rather

$$\frac{\kappa\Delta z}{4}\texttt{K1}[T^{(e)}] = \lambda[T^{(e)}] \quad \rightarrow \quad \texttt{K1}[T^{(e)}] = \frac{4}{\kappa\Delta z}\lambda[T^{(e)}] \, .$$

The symbolic Matlab solution is

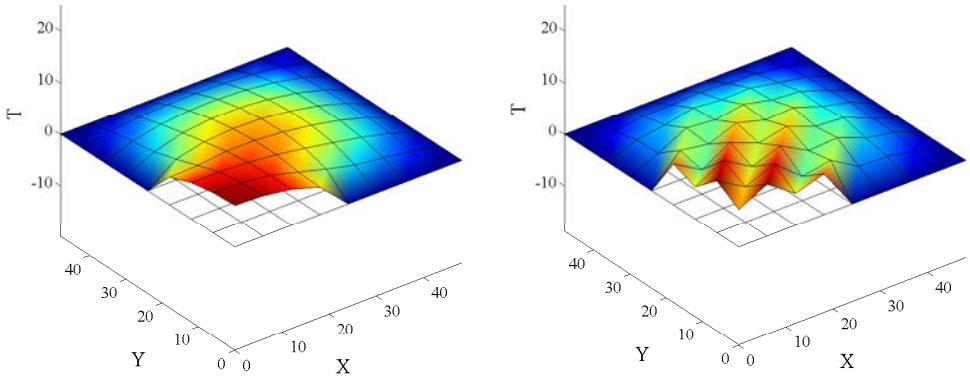
```
>> [V,D]=eig(K1)
V =
[ 1, 0,  1, -1]
[ 0, 1, -1, -1]
[ 1, 0, -1,  1]
[ 0, 1,  1,  1]
D =
[ 0, 0,       0,       0]
[ 0, 0,       0,       0]
[ 0, 0, (4*B)/A,       0]
[ 0, 0,       0, (4*A)/B]
```

The graphical representation of the first two modes that correspond to zero eigenvalues helps explain what's going on. The modes are variations of the temperature across the element, shown as a surface raised above the element to the third dimension. The slope of the temperature surface corresponds to the gradient of the temperature. We can see that the gradient of the temperature is nonzero everywhere, except at the midpoint of the element where the two dashed lines that lie in the temperature surface intersect. Those two lines have zero slope, and therefore at the midpoint the gradient of the temperature is zero. But that is exactly where the conductivity matrix was integrated. The integration of the conductivity matrix may be thought of as imposition of constraints on the variations of temperature across the element that are allowed. In particular, the correct conductivity matrix should allow for a uniform temperature distribution, which would correspond to zero temperature gradient everywhere. Clearly imposing such a constraint in the quadrilateral at a single quadrature point is not sufficient, since there are possible distributions of temperature which are not uniform but which do give a zero temperature gradient at the quadrature point.



The problem with a deficient-rank elementwise conductivity matrix is that even when the conductivity matrix of the entire structure is assembled, the deficiency of the individual element matrices may show up. Either the global conductivity matrix is genuinely singular, or it is close to singular.

The figure below shows a particular steady-state heat conduction case with nonzero internal heat generation rate and a combined essential and natural boundary condition. On the left the solution was obtained with the Q4 quadrilateral used with the 2×2 Gauss quadrature, on the right the same mesh is used with a one-point Gauss quadrature. For the latter scheme the individual elementwise conductivity matrices are singular, and the global conductivity matrix is close to singular. The singular matrix is prone to instabilities of the solution, and we can compare the patterns of zero-eigenvalue modes for the individual elements shown above with the oscillations that developed in the solution below.



11.9 Extracting the mesh boundary

The shapes of the geometric cells in the `FAESOR` toolbox are linked together through the "taking the boundary" operation (symbol ∂ in Fig. 11.13). This capability is crucial because some operations need to be performed over the volume of the mesh, while others should be evaluated over the surface. Also, the volume and surface integrals may need to be computed for models of different number of space dimensions. The general utility `mesh_bdry`[7] may be used to extract the boundary from a mesh or a mesh subset. To support these operations, the geometric cells have the responsibility of computing the connectivity of their boundary and supplying the handle of the constructor of the appropriate boundary geometric cell with their `get` method.

Figure 11.13 summarizes how the various types of geometric cells fit together. Some of these types have been discussed already, some make their appearance later in the book.

[7] Folder: `FAESOR/meshing`

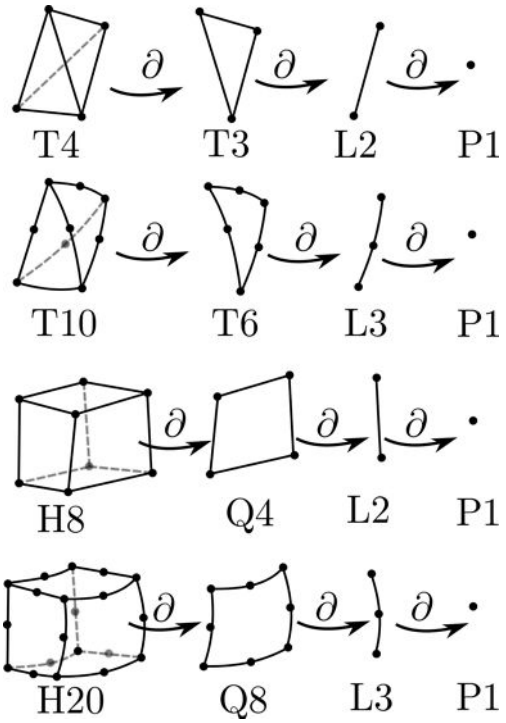


Fig. 11.13. The families of geometric cells and their boundaries. The symbol ∂ means "extract the boundary". Note that only one cell of the boundary set is shown. For instance, T4 boundary consists of four T3 cells, the boundary of one T3 consists of three L2 cells, and the boundary of one L2 consists of two P1's.

12

Discretization Error, Error Control, and Convergence

In this chapter we will address the error of the finite element approximation. In particular, we will inspect the so-called discretization error, which is the part of the error that is due to the introduction of the finite element basis functions. Also, we will discuss ways of controlling the error.

We begin by outlining how to estimate interpolation errors. The finite element solution in general does not interpolate the exact solution– meaning that the finite element nodal value is different from the value of the exact solution at the node, but it turns out that the interpolation errors are related to the actual errors in the numerical solution. Even though we will not address this relationship, it will prove beneficial to understand the behavior of the different types of errors on the simpler case of the interpolation errors.

12.1 Motivating example

First we look at an example. We pick a problem that has an analytical solution so that one can define true errors for the quantity v as

$$E_{v,h} = v_{\text{ex}} - v_h$$

as the difference between the true solution v_{ex} and the computed solution v_h, where by the notation $\cdot_h$ we mean that the computed solution was obtained with a finite element mesh and h indicates some measure of the properties of the mesh (for instance, the mesh size).

The variable can be anything of interest in the solution of the problem. For instance, for a heat conduction problem the variable v could stand for the temperature, or the heat flux, or perhaps the temperature gradient, or total amount of heat power in the volume of the domain.

In the example below we consider heat conduction through a wall, and we observe the error in the temperature and the error in the heat flux.

Exercise 61.

Wall is exposed to given temperature $T = 0^o$C at both faces. Heat is generated internally at the uniform rate Q. Use the mesh of seven equal-length L2 finite elements. Compute the the exact error of the temperature and the heat flux. Set $L = 6$m, $\kappa = 4\text{W/m/}^o\text{K}$, $Q = .1\text{W/m}^3$.

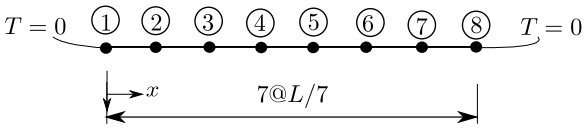


Fig. 12.1. Wall with prescribed temperatures at the boundary and uniform internal heat generation. Finite element mesh of seven L2 elements.

Solution: The numbering of the nodes is shown in Figure 12.1. The free degrees of freedom 1,...,6 are assigned to the nodes 2,...,8. The elements are numbered left or right. Therefore, the elementwise conductivity matrices and source heat load vectors are assembled as

$$[K] = \frac{\kappa}{L/7}\begin{bmatrix} 2, & -1, & 0, & 0, & 0, & 0 \\ -1, & 2, & -1, & 0, & 0, & 0 \\ 0, & -1, & 2, & -1, & 0, & 0 \\ 0, & 0, & -1, & 2, & -1, & 0 \\ 0, & 0, & 0, & -1, & 2, & -1 \\ 0, & 0, & 0, & 0, & -1, & 2 \end{bmatrix}$$

and

$$[F] = \frac{QL}{7}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

The solution is

$$[T] = \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{bmatrix} = \frac{QL^2}{49\kappa}\begin{bmatrix} 3 \\ 5 \\ 6 \\ 6 \\ 5 \\ 3 \end{bmatrix}$$

The two degrees of freedom $T_7 = T_8 = 0^o$C are known, and so we can plot the finite element solution using the trial function expansion

$$T(x) = \sum_{i=1}^{8} N_{\langle i \rangle}(x) T_i$$

and compare with the analytical solution. In Figure 12.2 (on the left) the dotted curve is the analytical solution, the finite element solution is the solid curve. The error is there, but perhaps difficult to see at this resolution. It *is* visible that the finite element solution that agrees with the exact solution at the nodes (it doesn't happen in general, only in 1-D!). The character of the error of the temperature is clearly brought out in Figure 12.2 (on the right) which displays the error

$$E_{T,h}(x) = T_{\text{ex}}(x) - T_h(x)$$

On each of the seven L2 elements the temperature error is represented by a little parabolic arc. The error is zero at the nodes.

The heat flux is computed using the definition

$$q = -\kappa T'$$

The gradient of the temperature is computed using the definition of the trial function

$$T'(x) = \sum_{i=1}^{8} N'_{\langle i \rangle}(x) T_i$$

It can be computed conveniently element-by-element taking advantage of the fact that over each element only to basis functions are nonzero. We write for the L2 element that connects nodes K, M

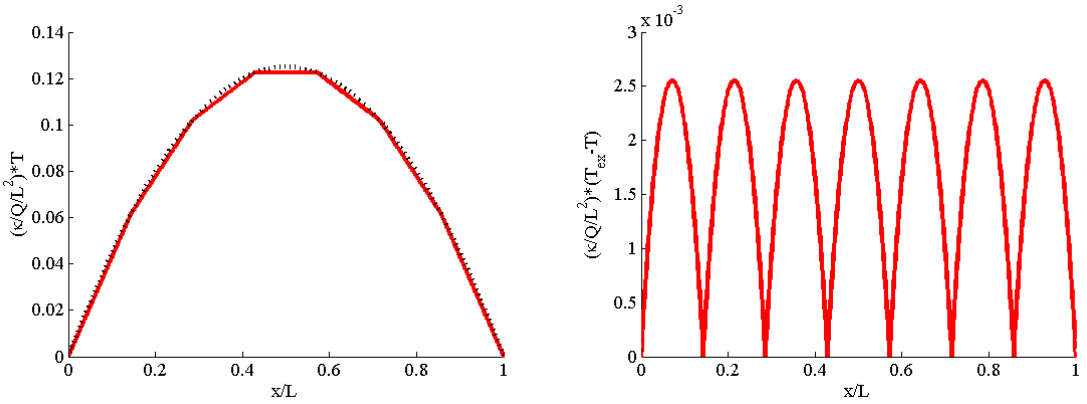


Fig. 12.2. Wall with prescribed temperatures at the boundary and uniform internal heat generation. The temperature distribution on the left, dotted curve exact, solid curve approximate. The error of the temperature on the right.

$$T'(x) = N'_K(x)T_{(K)} + N'_M(x)T_{(M)}$$

where we easily work out $N'_K(x) = -1/h$ and $N'_M(x) = +1/h$, with h being the length of the element, so that

$$T'(x) = \frac{T_{(M)} - T_{(K)}}{h}$$

and the heat flux is within the element K, M

$$q = -\kappa \frac{T_{(M)} - T_{(K)}}{h}$$

Note that within each element the heat flux is uniform, which goes hand-in-hand with the linear variation of the temperature within the element. For instance, for element 1 we get ($K = 1$, $M = 2$, $(K) = 7$, $(M) = 1$)

$$q = -\kappa \frac{T_{(M)} - T_{(K)}}{h} = -\kappa \frac{\frac{QL^2}{49\kappa} \times 3 - 0}{L/7} = -\frac{3QL}{7} = -0.2571$$

Compare with Figure 12.3. The horizontal lines represent the uniform heat flux within each element. The true heat flux is indicated with the continuous dotted line. The difference between the discontinuous curve of the finite element heat fluxes and the continuous line of the true heat flux is the error of the heat flux. In Figure 12.3 on the right the error in the heat flux is shown and we see that it is represented by a linear function within each element.

In the next example we will consider a higher order finite element: the quadratic L3 line element with three nodes. It will be applied to the problem of static deflection of the prestressed wire.

Exercise 62.

Compute the deflection and the transverse force of the prestressed wire under the transverse load with a triangular distribution (Figure 12.4). $P = 4$N, $q_0 = 0.1$N/m, $L = 6$m. Use the mesh of three L3 finite elements

Element	Nodes
1	1,2,5

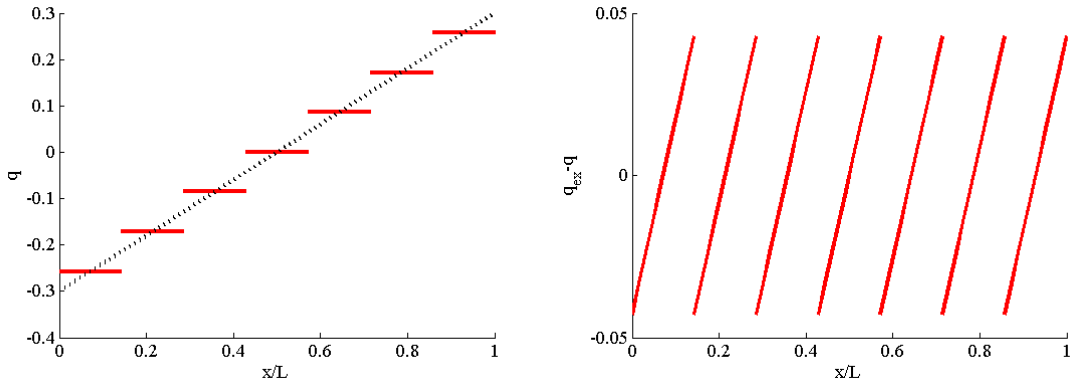


Fig. 12.3. Wall with prescribed temperatures at the boundary and uniform internal heat generation. The heat flux on the left, dotted curve exact, solid curve approximate. Error of the heat flux on the right.

2	2,3,6
3	3,4,7

where the interior nodes are at the midpoint of each element. There are five unknown degrees of freedom, which are assigned to nodes as

Node	1	2	3	4	5	6	7
Degree of freedom	6	1	2	7	3	4	5

and $w_6 = w(0) = 0$ and $w_7 = w(L) = 0$ at the pinned ends.

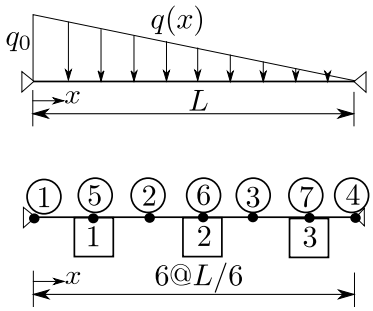


Fig. 12.4. Prestressed wire with linearly varying transverse load. The finite element mesh of three L3 Finite elements of equal length.

Solution: We have investigated the elementwise load vector for the L2 element and the linear variation of the transverse distributed load in Exercise 20. Here we will repeat this calculation using the Gauss two-point integration rule.

To empower the students with computer-aided symbolic algebra skills we present the complete calculation in Matlab here: First we define the symbolic variables, `xK`, `xL` are the locations of the end nodes, `xi` is the parametric coordinate, `qK` and `qL` are the values of the linearly varying load at the nodes.

```
syms xK xL  xi qK qL  real
```

The basis functions and the gradient of the basis functions with respect to the parametric coordinate of the L3 element are

```
N=[xi*(xi-1)/2; xi*(xi+1)/2; (1-xi^2)];
gradNxi=[diff(N,'xi')];
```

The array of the locations of the nodes is defined as

```
x=[xK;xL;(xK+xL)/2];
```

where we note that the third node was placed at the midpoint of the element. We compute the Jacobian, and we note that it is constant and equal to one half the length of the element

```
>> J=x'*gradNxi
J=collect(simplify(J),xi)
J =
xK*(xi - 1/2) + xL*(xi + 1/2) - 2*xi*(xK/2 + xL/2)
J =
xL/2 - xK/2
```

i.e. $J = L/6$.

The distributed transverse load varies linearly between the nodes. In the parametric coordinates we can express it as

```
q=(1-xi)/2*qK+(1+xi)/2*qL;
```

using the well-known Lagrange interpolation functions. At this point we are ready to execute the numerical integration. Note that the integrand is a function of the parametric coordinate, which is why we use `subs` to substitute the location of the quadrature point.

```
L=zeros(3,1);
xi=-0.577350269189626; W=1; % location and weight of quadrature point
L=L+subs(N*q*det(J))*W;
xi=0.577350269189626; W=1; % location and weight of quadrature point
L=L+subs(N*q*det(J))*W;
simple(L)
ans =
        -(qK*(xK - xL))/6
        -(qL*(xK - xL))/6
 -((qK + qL)*(xK - xL))/3
```

And so we obtain the elementwise load vector of the L3 finite element for linearly varying transverse load

$$[L^{(e)}] = \frac{h}{6}\begin{bmatrix} q_K \\ q_L \\ 2(q_K + q_L) \end{bmatrix}$$

where for convenience we set $h = x_L - x_K$.

For instance, for element 1 we obtain $h = L/3$, $q_K = q_0$ and $q_L = (2/3)q_0$ and the elementwise load vector

$$[L^{(e)}] = \frac{L/3}{6}\begin{bmatrix} q_0 \\ (2/3)q_0 \\ 2(q_0 + (2/3)q_0) \end{bmatrix} = \frac{Lq_0}{18}\begin{bmatrix} 1 \\ 2/3 \\ 10/3 \end{bmatrix}$$

These elementwise load vectors are assembled as usual

$$[L] = \begin{bmatrix} 0.044444 \\ 0.022222 \\ 0.11111 \\ 0.066667 \\ 0.022222 \end{bmatrix}$$

Taking advantage of the analogy between the wire model and the model of heat conduction we modify easily the elementwise conductivity matrix of the L3 element (11.5)

$$[K^{(e)}] = \frac{P}{h}\begin{bmatrix} 7/3, & 1/3, & -8/3 \\ 1/3, & 7/3, & -8/3 \\ -8/3, & -8/3, & 16/3 \end{bmatrix}$$

Note however that this matrix is valid only for the interior node at the midpoint of the element. The global stiffness matrix is assembled as

$$[K] = \begin{bmatrix} 9.3333, & 0.66667, & -5.3333, & -5.3333, & 0 \\ 0.66667, & 9.3333, & 0, & -5.3333, & -5.3333 \\ -5.3333, & 0, & 10.667, & 0, & 0 \\ -5.3333, & -5.3333, & 0, & 10.667, & 0 \\ 0, & -5.3333, & 0, & 0, & 10.667 \end{bmatrix}$$

and solution follows

$$[w] = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{bmatrix} = \begin{bmatrix} 0.055556 \\ 0.044444 \\ 0.038194 \\ 0.05625 \\ 0.024306 \end{bmatrix}$$

The two degrees of freedom $w_6 = w_7 = 0$ are known. The finite element solution

$$w(x) = \sum_{i=1}^{7} N_{\langle i \rangle}(x) w_i$$

is compared with the analytical solution in Figure 12.5. The difference is hardly visible, as the element is highly accurate.

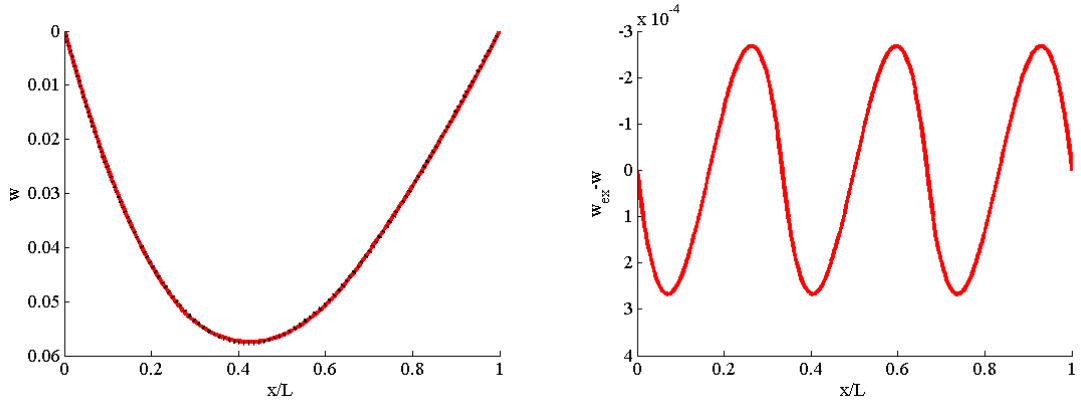


Fig. 12.5. Prestressed wire with linearly varying transverse load. The deflection on the left, dotted curve exact, solid curve approximate. Error of the deflection on the right.

Plotting the true displacement error as the difference between the analytical deflection and finite element solution brings out the features of the error (Figure 12.5 on the right). Interestingly we can see that the deflection error appears to be the same in all the elements, and its variation seems to be a cubic curve. We expect the L3 element to be able to match quadratic deflection curves, so in fact the first polynomial term that it cannot match is cubic!

The transverse force is computed as $S = Pw'$ which is best evaluated over each element separately: For instance, for the first element

$$S = Pw' = P\left(N_1(x)w_{(1)} + N_2(x)w_{(2)} + N_5(x)w_{(5)}\right)' =$$

$$P\left(N_1'(x)w_{(1)} + N_2'(x)w_{(2)} + N_5'(x)w_{(5)}\right) = P\left(N_1'(x)w_6 + N_2'(x)w_1 + N_5'(x)w_3\right)$$

The derivatives of the basis functions need to be computed from (8.53). Fortunately the Jacobian was shown above to be constant $J = L/6$, and since the gradients with respect to the parametric coordinate are linear functions

```
gradNxi =
 xi - 1/2
 xi + 1/2
    -2*xi
```

the gradients with respect to x are also going to be linear along each element. For instance

$$N_1' = (\xi - 1/2)/(L/6)\,, \quad N_2' = (\xi + 1/2)/(L/6)\,, \quad N_5' = (-2\xi)/(L/6)\,,$$

Thus at $x = 0$ which corresponds to $\xi = -1$ on element 1 we get the transverse force

$$S(0) = P\left(N_1'(-1) \times 0 + N_2'(-1) \times 0.055556 + N_5'(-1) \times 0.038194\right) =$$

$$P/(L/6)\left((-1 - 1/2) \times 0 + (-1 + 1/2) \times 0.055556 + (-2(-1)) \times 0.038194\right) = 0.1944\text{N}$$

The error of the transverse force is shown in Figure 12.6 on the right. Apparently the error varies as a quadratic function of the distance along the element. Again, intuitively we would guess that since the L3 element can match quadratic deflection curves, which means linearly varying slopes, the first polynomial term that it fails to match in the exact slope (equivalently in the transverse force) is the quadratic.

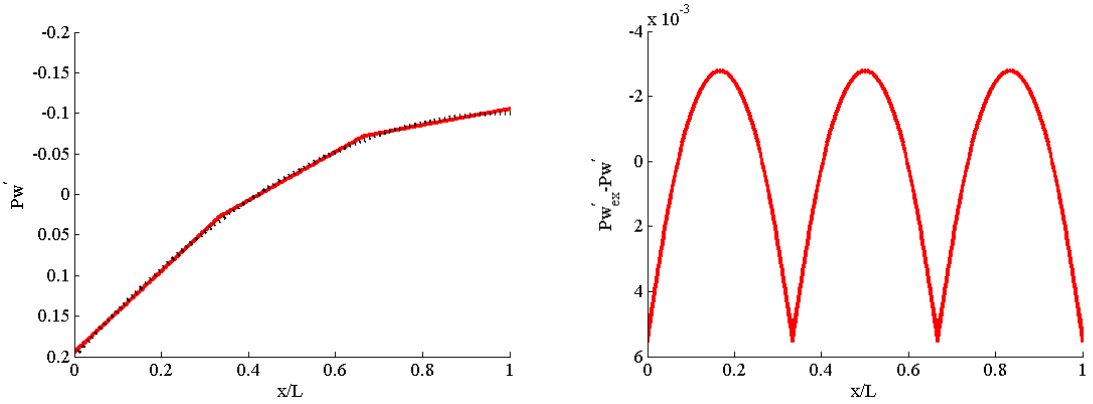


Fig. 12.6. Prestressed wire with linearly varying transverse load. The transverse force on the left, dotted curve exact, solid curve approximate. Error of the transverse force on the right.

12.2 Interpolation errors

We will estimate the difference between the "exact" distribution of temperature, $T(\boldsymbol{x})$, and an interpolation of this function on a finite element mesh, $\Pi_h T(\boldsymbol{x})$. Here h means the mesh "size", or characteristic dimension. Typically, mesh size is taken to mean edge length, or the diameter of the smallest ball that completely encloses an element.

12.2.1 Interpolation error for temperature

The interpolating function is defined as

$$\Pi_h T(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}_k) \ , \tag{12.1}$$

where $\boldsymbol{x}_k$ is the location of the node k, and $T(\boldsymbol{x}_k)$ is the value of the temperature at the location of the node k. For interpolation on a mesh consisting of three-node triangles, when $\boldsymbol{x}$ is in the interior of the element Δ_e, only three basis functions N_k are nonzero at $\boldsymbol{x}$. The basic tool is the Taylor series

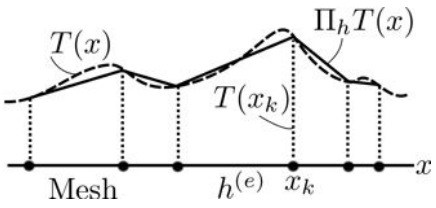


Fig. 12.7. Interpolating the temperature function on a mesh

which we use to expand the temperature at $\boldsymbol{x}$

$$T(\boldsymbol{y}) = T(\boldsymbol{x}) + \text{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{y} - \boldsymbol{x}) + R_1(\boldsymbol{y}, \boldsymbol{x}) \ , \tag{12.2}$$

where the remainder is written as

$$R_1(\boldsymbol{y}, \boldsymbol{x}) = \frac{1}{2} (\boldsymbol{y} - \boldsymbol{x}) \cdot \boldsymbol{H}(T) (\boldsymbol{y} - \boldsymbol{x}) \ . \tag{12.3}$$

The matrix of second derivatives (Hessian) is evaluated at $\boldsymbol{\xi}$ somewhere between the points $\boldsymbol{y}$ and $\boldsymbol{x}$

$$[\boldsymbol{H}(T)] = \begin{bmatrix} \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_1 \partial x_1} & \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_2 \partial x_1} & \dfrac{\partial^2 T(\boldsymbol{\xi})}{\partial x_2 \partial x_2} \end{bmatrix} \ .$$

The Taylor series (12.2) may be used to express the value of the temperature at the nodes– plug in $\boldsymbol{x}_k$ for $\boldsymbol{y}$– which then may be substituted into the interpolation (12.1) to yield

$$\begin{aligned} \Pi_h T(\boldsymbol{x}) = \sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}_k) = \\ \sum_k N_k(\boldsymbol{x}) \left[T(\boldsymbol{x}) + \text{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) + R_1(\boldsymbol{x}_k, \boldsymbol{x}) \right] \ . \end{aligned}$$

Due to the construction of the basis functions, we have these important equalities

$$\sum_k N_k(\boldsymbol{x}) = 1 \ , \quad \sum_k N_k(\boldsymbol{x}) \boldsymbol{x}_k = \boldsymbol{x} \ . \tag{12.4}$$

Therefore, the first term in (12.4) simplifies as

$$\sum_k N_k(\boldsymbol{x}) T(\boldsymbol{x}) = T(\boldsymbol{x}) \sum_k N_k(\boldsymbol{x}) = T(\boldsymbol{x}) \ ,$$

and the second will vanish

$$\sum_k N_k(\boldsymbol{x}) \text{grad}T(\boldsymbol{x}) \cdot (\boldsymbol{x}_k - \boldsymbol{x}) = \text{grad}T(\boldsymbol{x}) \cdot \sum_k N_k(\boldsymbol{x}) (\boldsymbol{x}_k - \boldsymbol{x}) = 0 \ .$$

Substituting into (12.4) gives

$$\Pi_h T(\boldsymbol{x}) = T(\boldsymbol{x}) + \sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x}) \ ,$$

or, reshuffling to get the error on one side,

$$T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x}) = -\sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x}) \ . \tag{12.5}$$

To estimate the magnitude of the difference, $|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})|$, we compute

$$|\sum_k N_k(\boldsymbol{x}) R_1(\boldsymbol{x}_k, \boldsymbol{x})| \leq \max|R_1(\boldsymbol{x}_k, \boldsymbol{x})| \, |\sum_k N_k(\boldsymbol{x})| = \max|R_1(\boldsymbol{x}_k, \boldsymbol{x})| \ ,$$

and make use of standard norm inequalities

$$|\boldsymbol{v} \cdot \boldsymbol{A}\boldsymbol{v}| \leq \|\boldsymbol{v}\| \|\boldsymbol{A}\boldsymbol{v}\| \leq \|\boldsymbol{A}\| \|\boldsymbol{v}\|^2 \ .$$

This may be applied to the definition of the remainder (12.3) together with (see Fig. 12.8)

$$\|\boldsymbol{x}_k - \boldsymbol{x}\| \leq h \ ,$$

and an estimate of the norm of the matrix of second derivatives of the temperature $\boldsymbol{H}(T)$ to give

$$|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})| \leq C h^2 \|\boldsymbol{H}(T)\| \ . \tag{12.6}$$

Here C is a generic constant with respect to h. If we wrap the norm of the matrix of the second derivatives into the constant, we may write

$$|T(\boldsymbol{x}) - \Pi_h T(\boldsymbol{x})| \leq C \left(\partial^2 T\right) h^2 \ , \tag{12.7}$$

where we agree to mean by $C\left(\partial^2 T\right)$ some constant whose magnitude depends on the curvatures of the function T. Importantly, $C\left(\partial^2 T\right)$ may also be understood as measuring the *rate of change of the heat flux* in the immediate neighborhood of $\boldsymbol{x}$.

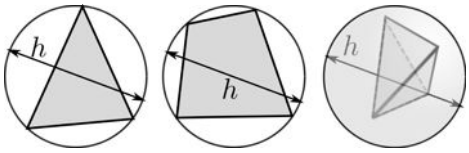


Fig. 12.8. Mesh size h as a "diameter" of an element: diameter of the tight-fitting circle that encloses a triangle or a quadrilateral, and of the smallest sphere that is circumscribed to a tetrahedron

The value of Eq. (12.7) is twofold:

- Firstly, it states that the errors of interpolation will get bigger the higher the curvature of the function of the exact temperature T (that is the faster the heat flux is changing) and the bigger the elements (i.e. the error will increase with h^2);
- Secondly, if we are interested in the interpolation error at a particular location, we may consider the curvatures at that location as given, and the Eq. (12.7) then says that the error will decrease as $O(h^2)$ as $h \to 0$ (order-of estimate: reduce h with a factor of two, and the error will decrease with a factor of four).

12.2.2 Interpolation error for temperature gradient

To estimate errors for the gradient of temperature, we start with the interpolation (12.4), of which we take the gradient

$$\begin{aligned}
\mathrm{grad}\Pi_h T(\boldsymbol{x}) = &\sum_k \mathrm{grad}N_k(\boldsymbol{x})T(\boldsymbol{x}_k) = \\
&\sum_k \mathrm{grad}N_k(\boldsymbol{x})\left[T(\boldsymbol{x}) + \mathrm{grad}T(\boldsymbol{x})\cdot(\boldsymbol{x}_k - \boldsymbol{x}) + R_1(\boldsymbol{x}_k, \boldsymbol{x})\right] = \\
&\sum_k \mathrm{grad}N_k(\boldsymbol{x})T(\boldsymbol{x}) + \sum_k \mathrm{grad}N_k(\boldsymbol{x})\mathrm{grad}T(\boldsymbol{x})\cdot(\boldsymbol{x}_k - \boldsymbol{x}) \\
&\qquad + \sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x}) = \\
&T(\boldsymbol{x})\sum_k \mathrm{grad}N_k(\boldsymbol{x}) + \mathrm{grad}T(\boldsymbol{x})\cdot\sum_k \mathrm{grad}N_k(\boldsymbol{x})(\boldsymbol{x}_k - \boldsymbol{x}) \\
&\qquad + \sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x})\ .
\end{aligned} \tag{12.8}$$

Differentiating (12.4) we obtain

$$\sum_k \mathrm{grad}N_k(\boldsymbol{x}) = 0\ , \quad \sum_k \boldsymbol{x}_k \mathrm{grad}N_k(\boldsymbol{x}) = \mathbf{1}\ , \tag{12.9}$$

which upon substitution into (12.8) yields

$$\mathrm{grad}\Pi_h T(\boldsymbol{x}) = \mathrm{grad}T(\boldsymbol{x}) + \sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x})\ . \tag{12.10}$$

Again, an estimate of the magnitude is desired,

$$\begin{aligned}
|\mathrm{grad}T(\boldsymbol{x}) - \mathrm{grad}\Pi_h T(\boldsymbol{x})| = |\sum_k \mathrm{grad}N_k(\boldsymbol{x})R_1(\boldsymbol{x}_k, \boldsymbol{x})| \leq \\
\max|R_1(\boldsymbol{x}_k, \boldsymbol{x})|\ \sum_k |\mathrm{grad}N_k(\boldsymbol{x})|\ .
\end{aligned}$$

To estimate the magnitude of the gradient of the basis function, we invoke the picture of the basis

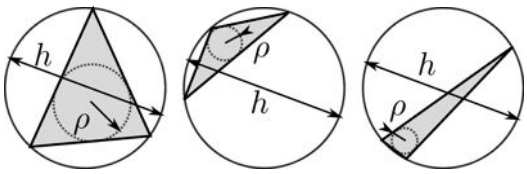


Fig. 12.9. Triangle quality measures using the radius of the inscribed circle and the diameter of the circumscribed circle. Good (almost equilateral) triangle on the left; bad triangles (obtuse, needle-like) on the right.

function as a plane that assumes value one at one node and drops off to zero along the opposite edge. Therefore, the largest magnitude of the basis function gradient will be produced by the smallest height in the triangle. The shortest height $d_{\min}$ may be estimated from the radius of the largest inscribed circle, ρ (see Fig. 12.9), as $d_{\min} \approx O(\rho)$. This can be linked to the so-called "shape quality" of a triangle using the ***quality measure***

$$\gamma = \frac{h}{\rho}\ ,$$

as $d_{\min} \approx O(\gamma^{-1})h$. The magnitude of the basis function gradient may be then estimated as

$$\max \text{grad} N_k(\boldsymbol{x}) = \frac{1}{d_{\min}} \approx \frac{\gamma}{h} .$$

Putting everything together, we obtain

$$|\text{grad}T(\boldsymbol{x}) - \text{grad}\Pi_h T(\boldsymbol{x})| \leq Ch^2 \frac{\gamma}{h} \|\boldsymbol{H}(T)\| = C\left(\partial^2 T\right)\gamma h . \tag{12.11}$$

The value of Eq. (12.11) is again twofold:

- Firstly, it states that the errors of interpolation for the gradient of temperature will get bigger the higher the curvature of the function of the exact temperature T, the larger the elements (i.e. the error will increase with h), and the larger the quality measure γ (i.e. the worse the shape of the triangle);
- Secondly, considering the curvatures at a fixed location as given, the equation (12.11) states that the error will decrease as $O(h)$ as $h \to 0$ (note that this is one order lower than for the temperatures themselves: reduce h with a factor of two, and the error will decrease with the same factor).

Importantly, Eq. (12.11) allows us to make a general observation: the quantity calculated in the finite element solution is the temperature, the gradient (or, alternatively, the heat flux) is obtained by *differentiation* of the computed temperature, which immediately results in a reduction of the order of dependence on the mesh size. Phrased differently: the temperature results will converge faster than the temperature-gradient results (or, equivalently, heat flux results), because h^2 approaches zero much quicker than h as $h \to 0$.

12.2.3 Controlling the error; Convergence rate

That the error depends on the mesh size is very important. The mesh size is in fact one of the levers we can use to control the error. If we set up the finite element procedure to solve the problem repeatedly, changing the mesh size to reflect the distribution of error (large error – small elements), we obtain the so-called ***adaptive refinement*** technique, or $h-$***adaptive refinement method***. The h stands for the mesh size as the control of the error. On the other hand, we could try to reduce the error by increasing the number of terms matched in the Taylor series (12.2). This could be achieved by using higher order polynomials as basis functions. The resulting procedure would be called the $p-$***adaptive refinement method***, where the p stands for the polynomial order as the control of the error.

In this book, we will select the polynomial order of the elements only by choosing the element type, linear or quadratic. We will not do this adaptively which would involve increasing the polynomial order in a targeted fashion, locally, and to much higher order than just quadratic.

We will focus on the control of the error by adjusting the mesh size. Hence, we will adopt the point of view of the $h-$adaptive refinement method. In order to be able to control the error, it must depend on the mesh size h, and furthermore it must be expressible as a *positive* (but not necessarily integral) *power* of h. Only then decreasing h will lead to a reduction of the error. For instance, for quantity q we require for the error

$$E_q(h) = q_{\text{ex}} - q_h \approx Ch^{\beta} ,$$

as the error then can be reduced by decreasing the mesh size

$$\lim_{h\to 0} E_q(h) = \lim_{h\to 0} Ch^{\beta} = 0 \qquad \text{for } \beta > 0 .$$

The exponent of the mesh size β is called the ***convergence rate*** (or rate of convergence).

Equation (12.11) tells us how to reduce the error. Consider that at a given location $\boldsymbol{x}$, $C\left(\partial^2 T\right)$ cannot be controlled as it is determined by the behavior of the exact solution at $\boldsymbol{x}$. What we *can*

influence is the shape of the elements (γ), and the mesh size (h). Now let the point $\boldsymbol{x}$ range across the computational domain. At some locations $C\left(\partial^2 T\right)$ is small, and at others it is large. As an illustration, let us contemplate Fig. 12.10 (and the close-up in Fig. 12.11). The constant $C\left(\partial^2 T\right)$ will be large where the heat flux changes a lot; therefore, where the red arrows which indicate the magnitude and direction of the heat flux strongly change direction or stretch or shrink, the error constant $C\left(\partial^2 T\right)$ should be expected to be large. Intuitively, those are the locations where reducing the error will make a difference. In these locations, the elements should be made smaller. How much? The answer to that question is somewhat elusive: in general a trial and error procedure or iteration will be required to construct a mesh that delivers the answer within acceptable error bounds. Automatic procedures to estimate the *relative* desired mesh size are becoming available in commercial softwares, and the trend is nowadays to provide some means for the user of finite element softwares the estimate and control error.

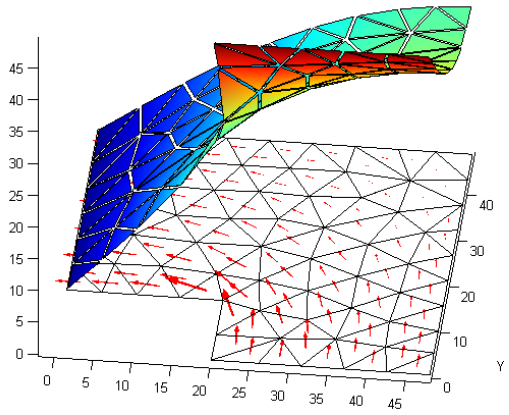


Fig. 12.10. The effect of a reentrant corner on the flux. Matlab script `lshape2`.

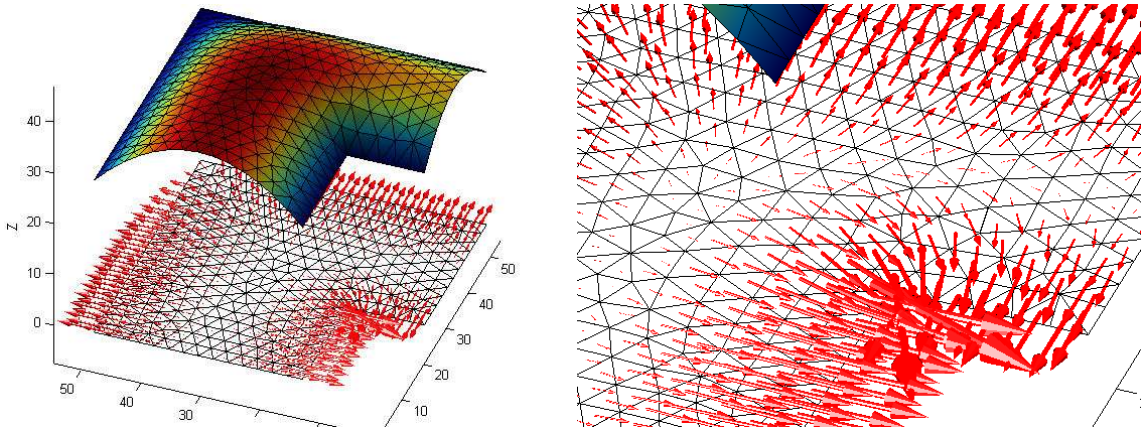


Fig. 12.11. L-shaped domain. The effect of a reentrant corner on the flux: overall view and close-up. Matlab script `lshape3`.

To summarize, the $h-$adaptive method will attempt to control the error by designing ***graded meshes***, with small elements located in regions of expected high error, and proportionally large elements elsewhere.

A very good indication of the presence large errors are the so-called ***reentrant corners*** (concave corners), where the solution typically displays singularities in the form of infinite curvature(s) of the

temperature directly in the corner. Figure 12.12 shows an appropriately refined mesh around the reentrant corner for the problem from Fig. 12.11.

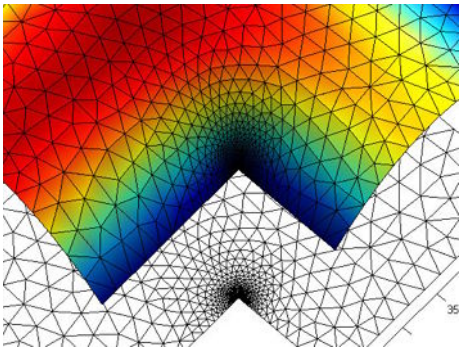


Fig. 12.12. The effect of a reentrant corner on the flux: adaptively refined mesh. Matlab script `lshape3ad`.

12.3 Richardson extrapolation

The second important use of the fact that the error decreases asymptotically as some power of the mesh size is a procedure to improve the estimate of the exact (true) answer based on a series of calculated solutions: in other words, mesh-size-based extrapolation.

Richardson extrapolation is a way of extracting an asymptotic estimate of some quantity of interest from a series of computed values for it. If we assume that the true error in the quantity q may be expanded in a Taylor series at mesh size $h = 0$, we may write

$$E_q(h) = q_{\text{ex}} - q_h \approx Ch^\beta \ , \tag{12.12}$$

where q_{ex} is the unknown true value of the quantity, q_h is the approximate value for nonzero h, $E_q(h)$ is the true error, C is an unknown constant of the leading term h^β, with β, again, unknown. Provided C, and β do not depend on h for small mesh sizes (this is presumed to hold in the so-called ***asymptotic range***), we might be able to compute all three q_{ex}, C, and β, if three numerical solutions are obtained for three different mesh sizes, q_{h_i}. (It does not matter whether the solutions are obtained with uniform or graded meshes.)

Remarkably, the estimate of the exact solution is available from a very easily solvable equation if the condition

$$\frac{h_1}{h_2} = \frac{h_2}{h_3} \ ,$$

holds, as we may then combine

$$\frac{q_{\text{ex}} - q_{h_1}}{q_{\text{ex}} - q_{h_2}} = \frac{h_1^\beta}{h_2^\beta} \quad \text{and} \quad \frac{q_{\text{ex}} - q_{h_2}}{q_{\text{ex}} - q_{h_3}} = \frac{h_2^\beta}{h_3^\beta} \ ,$$

to yield

$$q_{\text{ex}} = \frac{q_{h_2}^2 - q_{h_1} q_{h_3}}{2q_{h_2} - q_{h_1} - q_{h_3}} \ . \tag{12.13}$$

It is then straightforward to extricate the other two quantities. The constant C is of limited value, but the exponent β is the ***rate of convergence***. The computation is implemented in the toolbox `FAESOR` in the algorithm function `richextrapol`[1]. Thus we have the ***estimated true error***

[1] Folder: `FAESOR/algorithms`

$$E_q(h_j) = q_{\text{ex}} - q_{h_j} \ . \tag{12.14}$$

Extrapolation is in general a tricky procedure. Therefore, before we attempt the general Richardson extrapolation to estimate the true error the so-called ***approximate error*** should be considered.

$$E_{q,j} = q_{h_{j+1}} - q_{h_j} \ . \tag{12.15}$$

It relies only on computed quantities, $q_{h_{j+1}}$, and q_{h_j} for two different meshes with mesh sizes h_{j+1} and h_j. We can expect it to be more "honest" about the convergence than the formula for the estimation of the true error. The property of the approximate error that makes it so useful when we attempt to validate the predicted true error is that the approximate error depends on the mesh size in the same way as the true error. We can see that by adding and subtracting the true value to/from the approximate error as

$$E_{q,j} = q_{h_{j+1}} - q_{h_j} = q_{h_{j+1}} - q_{h_j} + q_{\text{ex}} - q_{\text{ex}} \ .$$

Regrouping the terms we obtain

$$E_{q,j} = \left(q_{h_{j+1}} - q_{\text{ex}}\right) + \left(q_{\text{ex}} - q_{h_j}\right) = -E_q(h_{j+1}) + E_q(h_j) \ . \tag{12.16}$$

i.e. as the difference of the true errors for the mesh sizes h_j and h_{j+1}. Consequently, we see that in the limit of the mesh size decreasing towards zero the approximate error will behave as a power of the mesh size, and the convergence rate will be the same as that of the true error of equation (12.12). We show this by introducing (12.12) into (12.16) to obtain

$$E_{q,j} = -E_q(h_{j+1}) + E_q(h_j) = -Ch_{j+1}^{\beta} + Ch_j^{\beta}$$

If all the meshes are related by a constant mesh refinement factor $\alpha < 1$ such that the mesh sizes for two successive meshes are $h_{j+1} = \alpha h_j$, we arrive at

$$E_{q,j} = -Ch_{j+1}^{\beta} + Ch_j^{\beta} = -C(\alpha h_j)^{\beta} + Ch_j^{\beta} = \left(1 - \alpha^{\beta}\right)\left(Ch_j^{\beta}\right) = \left(1 - \alpha^{\beta}\right) E_q(h_j) \tag{12.17}$$

Exercise 63. Solve for the maximum temperature in the square chimney of Figure 12.13. Estimate the true value using Richardson extrapolation. Consider Newton's boundary conditions: The interior ambient temperature is given as $T_a = 1000°\text{C}$, and the corresponding interior surface heat transfer coefficient is $h_{a,i} = 5\text{W}/(m^{2\circ}\text{K})$. The exterior ambient temperature is given as $T_a = -20°\text{C}$, and the corresponding exterior surface heat transfer coefficient is $h_{a,e} = 15\text{W}/(m^{2\circ}\text{K})$. Assume homogeneous isotropic material with thermal conductivity $\kappa = 0.5\text{W}/(m^{\circ}\text{K})$. The dimension is $a = 1.0\text{m}$.

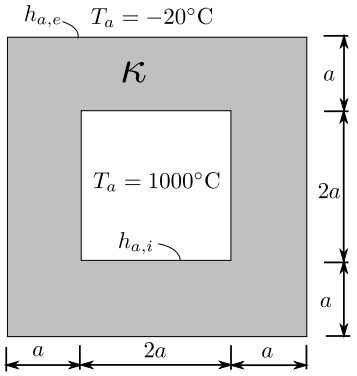


Fig. 12.13. Square chimney with convection boundary conditions

Solution: We shall use uniform meshes of T3 finite elements (for simplicity), and the solution will be obtained using several different mesh sizes. The meshes will be related by a factor of $\alpha = 1/2$, so for instance $h_2 = \alpha h_1$, $h_3 = \alpha h_2$. The computation is implemented in the script `ChimneyN`[2]. The

[2]Folder: `FAESOR/examples/diffusion`

temperature distribution is visualized in Figure 12.14. The computation repeated for the mesh sizes

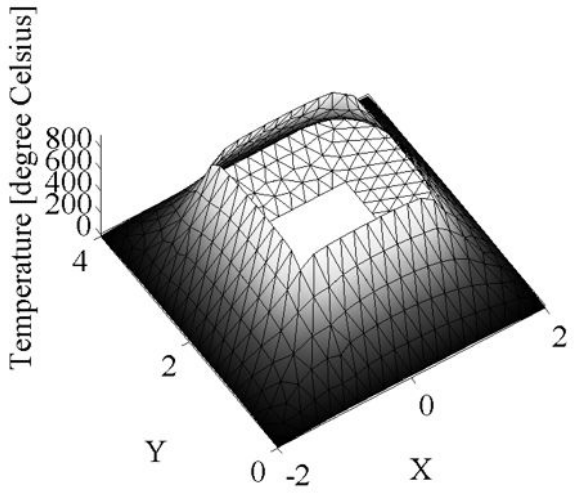


Fig. 12.14. Temperature distribution in the square chimney

(note the mesh refinement factor $\alpha = 1/2$)

```
>> mesh_sizes=0.5./2.^(0:1:4)
mesh_sizes =
  5.0000e-001  2.5000e-001  1.2500e-001  6.2500e-002  3.1250e-002
```

yields the computed maximum temperatures (coarsest to finest mesh)

```
maxT=[9.113434081642134e+002,9.059849518002127e+002,9.051223854809444e+002,...
    9.049018804469090e+002,9.048450848159254e+002];
```

The approximate errors are computed as

```
>> diff(maxT)
ans =
 -5.3585e+000 -8.6257e-001 -2.2051e-001 -5.6796e-002
```

If the approximate errors indeed depend on the mesh size through a power relationship then there is a way of presenting the data graphically to confirm this. Take the absolute value of (12.17) and write

$$|E_{q,j}| = \left|\left(1-\alpha^{\beta}\right) C \left(h_j^{\beta}\right)\right| = \left|\left(1-\alpha^{\beta}\right) C\right| \left|h_j^{\beta}\right|$$

Now take the log of both sides to obtain

$$\log|E_{q,j}| = \log\left|\left(1-\alpha^{\beta}\right) C\right| + \beta \log h_j \ ,$$

which is a linear relationship between $\log|E_{q,j}|$ and $\log h_j$. On a log-log plot the slope of the straight line is the convergence rate β. This is shown in Figure 12.15. The last three data points (for the three smallest mesh sizes) indeed lie on a straight line, whose slope is approximately 2.0. We can evaluate the slope using least-squares as

```
A =[log(mesh_sizes(2:end)'), ones(length(mesh_sizes)-1,1)];
b=log(abs(abs(diff(maxT)))');
A\b
ans =
  2.1648e+000
  4.5394e+000
```

which confirms our estimate as $\beta = 2.16$. Note that for large mesh size the computed approximate error deviates from the straight line. This is typical of results obtained for coarse meshes (the so-called pre-asymptotic range), because for coarse meshes more than one power of the mesh size contribute to the error and consequently the data points do not lie on a single straight line on the log-log scale.

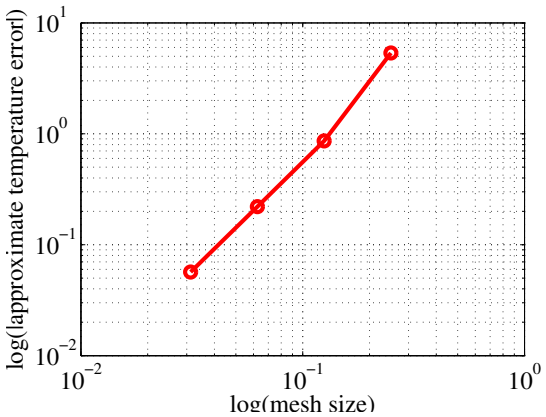


Fig. 12.15. Square chimney. Approximate error of the maximum temperature.

The true maximum temperature may be estimated from formula (12.13) from the results for the three finest meshes as

```
>> maxT1=maxT(3); maxT2=maxT(4); maxT3=maxT(5);
maxTestim=(maxT2^2-maxT1*maxT3)/(maxT2*2-maxT1-maxT3)
maxTestim =
  904.8254
```

The convergence rate may be solved for as

```
>> beta=log((maxTestim-maxT2)./(maxTestim-maxT3))/log(mesh_sizes(2)/mesh_sizes(3))
beta =
    1.9570
```

Note that the convergence rate is similar to that predicted by the slope of the approximate error curve, but not identical. The convergence rate should be theoretically identical in the limit of the mesh size going to zero, but before we get there there will be differences.

Both of the above calculations are implemented in a numerically robust fashion in the utility function `richextrapol`[3]:

```
>> [maxTestim, beta] = richextrapol(maxT(end-2:end),mesh_sizes(end-2:end))
maxTestim =
  904.8254
beta =
    1.9570
```

Using the estimated true solution `maxTestim` we can estimate the normalized true error as

```
>> (maxT-maxTestim)/maxTestim
ans =
  7.2036e-003  1.2815e-003  3.2825e-004  8.4546e-005  2.1777e-005
```

and this can be then presented graphically as shown in Figure 12.16

[3]Folder: `FAESOR/algorithms`

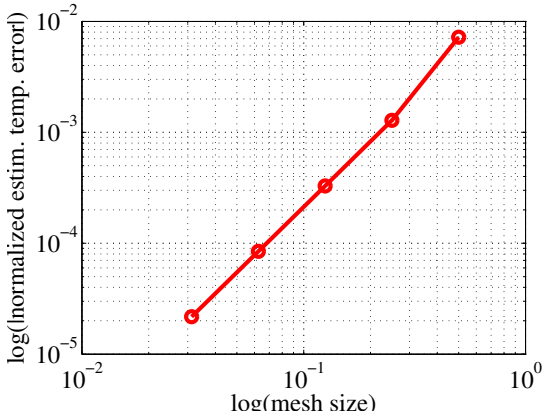


Fig. 12.16. Square chimney. Normalized estimated error of the maximum temperature.

An alternative way of estimating the true error may be deduced from equation (12.17). We express the true error $E_q(h_j)$ in terms of the approximate error and a factor which depends on the convergence rate. So, provided we can reliably estimate the convergence rate (when we plot the approximate errors they lie on a straight line on a log-log plot of slope β), we can express the true error as

$$E_q(h_j) = \frac{E_{q,j}}{1-\alpha^\beta} \ .$$

Substituting the definitions of the true error and of the approximate error we get

$$q_{\text{ex}} - q_{h_j} = \frac{q_{h_{j+1}} - q_{h_j}}{1-\alpha^\beta}$$

or

$$q_{\text{ex}} = q_{h_j} + \frac{q_{h_{j+1}} - q_{h_j}}{1-\alpha^\beta} \ , \tag{12.18}$$

which can be used as an alternative to the estimator (12.13).

Exercise 64. Repeat the estimation of the maximum temperature from Exercise 63 using the alternative estimation of equation (12.18).

Solution: The solutions for the five mesh sizes are

```
maxT =
  9.1134e+002  9.0598e+002  9.0512e+002  9.0490e+002  9.0485e+002
```

The mesh refinement factor is $\alpha = 1/2$ and the convergence rate was determined as $\beta = 2.16$ in Exercise 63. Therefore, using formula (12.18) with $j = 4$ we obtain the estimate of the maximum temperature

```
>> maxTj=maxT(4); maxTj1=maxT(5);
maxTj+(maxTj1-maxTj)/(1-(1/2)^ 2.16)
ans =
  9.0483e+002
```

which agrees well with the estimated value from Exercise 63. Good estimate is also obtained when we set $j = 2$

```
maxTj=maxT(2); maxTj1=maxT(3);
>> maxTj+(maxTj1-maxTj)/(1-(1/2)^ 2.16)
ans =
  9.0487e+002
```

as the estimated true value is off only in the fifth significant digit. Interestingly, using a ballpark estimate of the convergence rate ≈ 2, such as would be obtained by visual inspection of the graph of the approximate error, we get an impressively good estimate

```
maxTj=maxT(2); maxTj1=maxT(3);
>> maxTj+(maxTj1-maxTj)/(1-(1/2)^ 2.)
ans =
  9.0483e+002
```

That might be an exception to the rule, of course.

12.4 The T4 NAFEMS Benchmark revisited

This problem has been discussed in Section 9.4. The publication [CCS94] cites the reference value for the temperature at the point indicated in Fig. 9.5 of 18.3°C. However, more recent investigations of this benchmark indicate that value of 18.25°C should be expected [I05]. Let us check these numbers.

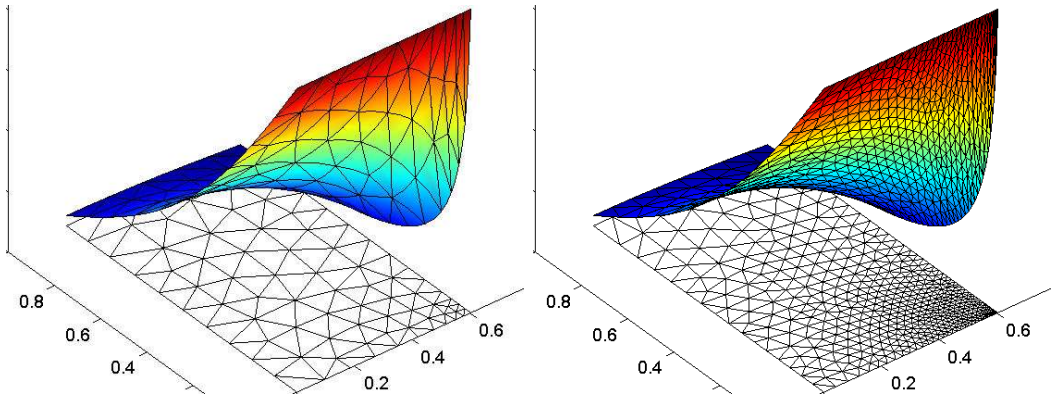


Fig. 12.17. T4 NAFEMS Benchmark: solution with quadratic elements, initial and final mesh.

Two models will be used, the first using elements T3, and the second using the more accurate quadratic elements T6. The Matlab script `t4nafems_conv`[4] runs the simulation (the initial and final adaptive meshes are shown in Fig. 12.17), with the results shown in Figure 12.18, the estimated true error (12.14) on the left, and the approximate error (as the difference between successive solutions (12.15)) on the right. For the quadratic elements, the Richardson extrapolation produces an estimate of the exact temperature 18.25396°C and the rate of convergence 2.2945. On the other hand, the element T3 performs erratically, and no asymptotic estimate is possible. That is clearly visible in Fig. 12.18: it should be possible to pass a straight line through the estimates of the error if the data is indeed in the asymptotic range, as taking a logarithm of (12.12) yields

$$\log|E_q(h)| = \log|q_{\text{ex}} - q_h| \approx \log C + \beta \log h \ ,$$

which is a straight line on a log-log scale (Fig. 12.18). That is out of the question for the element T3.

[4]Folder: `FAESOR/examples/diffusion`

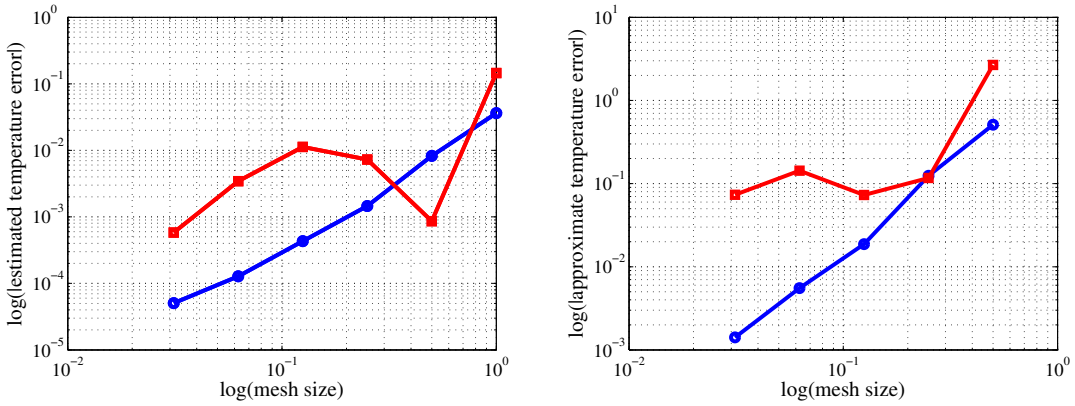


Fig. 12.18. T4 NAFEMS Benchmark: convergence in terms of an estimated error for linear and quadratic triangles

12.5 Graded meshes

A word on the meaning of the *mesh size in graded meshes* is in order. The mesh size in graded meshes varies from point to point (as opposed to uniform meshes, where the mesh size does not vary). However, if we take one particular graded mesh M_0, and produce a series of meshes from M_0 by scaling the mesh size as a function of $\boldsymbol{x}$ by the same number everywhere in the domain, we may take as the mesh size the scaling factor (the absolute values do not matter, only the relative changes $h_1/h_2 = h_2/h_3$). For instance, mesh M_1 would be produced with mesh size $h_1(\boldsymbol{x}) = \alpha h_0(\boldsymbol{x})$, mesh M_2 with mesh size $h_2(\boldsymbol{x}) = \alpha h_1(\boldsymbol{x}) = \alpha^2 h_0(\boldsymbol{x})$, and so on.

12.6 Shrink fitting revisited

Figure 12.19 shows the temperature distribution at three time instants. The extremely high gradient at the beginning is evident, but in fact high temperature gradients exist even at the end of the process.

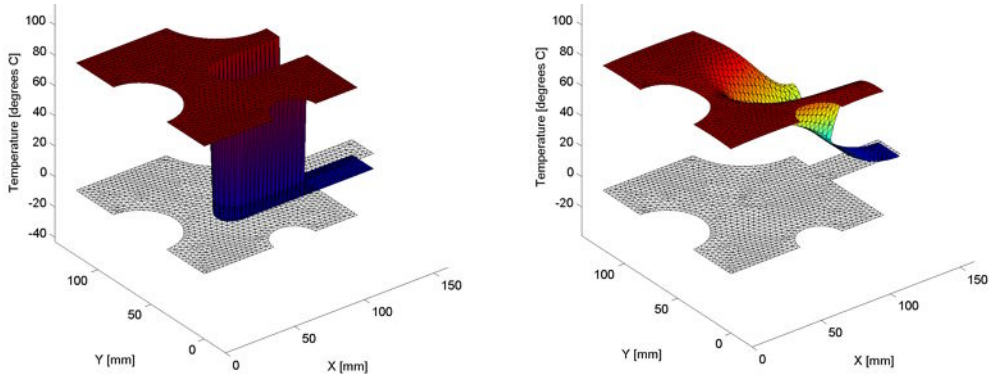


Fig. 12.19. Transient cooling of a shrink-fitted assembly; left to right: temperature distribution for time $t = 0$ and $t = 10$ seconds

As you will recall, the heat flux is derived from the temperature (Eq. (7.14)). The finite element approximation with the triangles (T3) and with the line elements (L2) will be able to reproduce linearly varying temperatures, hence constant temperature gradients (i.e. heat flux). Therefore, we

will conclude that where the heat flux changes, the finite element approximation will be in error. To control the error, we can reduce the element dimensions. Doing so in areas of steep changes in the heat flux, while keeping areas with approximately uniform heat flux tiled with coarse elements, is known as ***adaptive!mesh control***.

Figure 12.20 shows the heat flux on two meshes as arrows centered at the barycenters of the elements (barycenter here means average of the vertex locations). The first mesh is quite coarse (script `shrinkfitad1`[5]), but it is possible to identify regions in which the gradient changes strongly (next to the tungsten inset); the graded mesh is generated to reflect the demand for finer (smaller) elements (script `shrinkfitad2`[6]).

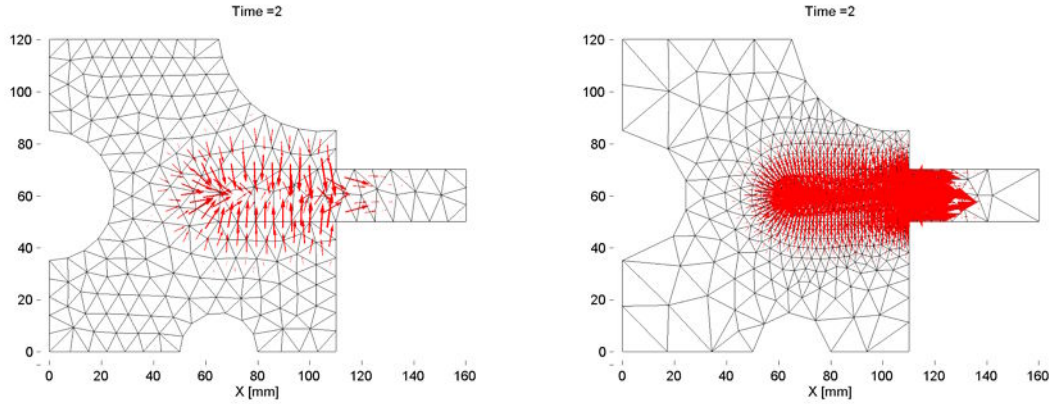


Fig. 12.20. Transient cooling of a shrink-fitted assembly; left: coarse mesh, right: adaptive mesh. Heat flux for time $t = 2$

The temperature evolution obtained with the two meshes, the coarse one, and the adaptively refined one, is illustrated in Fig. 12.21, and the higher-quality of the adaptive results should be noted: especially striking is the spurious oscillation of the highest temperature for the coarse uniform mesh.

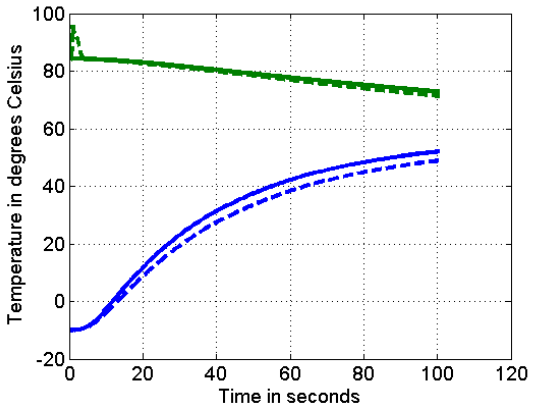


Fig. 12.21. Transient cooling of a shrink-fitted assembly: time evolution of the lowest and highest temperature in the assembly. Comparing temperatures obtained with a coarse model (dashed lines) and with an graded (adaptively refined) model (solid lines).

[5]Folder: `FAESOR/examples/diffusion`
[6]Folder: `FAESOR/examples/diffusion`

12.7 Representing functions by interpolation

Using the interpolation (12.1) we can approximate arbitrary functions on finite element meshes. It is of considerable interest to ask which of these functions may be represented exactly (***reproduced***). In other words, the question is for which functions $f(\boldsymbol{x})$ do we get the same function back,

$$\Pi_h f(\boldsymbol{x}) = f(\boldsymbol{x}) \ ,$$

when it is interpolated? To explore the issue, we will pick a quadratic element, the line element L3. The basis functions for this element are given on the standard interval by Eqs. (11.4). The element L3 is an iso-parametric element, meaning that the Cartesian coordinates of the nodes are interpolated using (8.25) to yield

$$x = N_1(\xi)x_1 + N_2(\xi)x_2 + N_3(\xi)x_3 \ , \tag{12.19}$$

which is a map of the form of (8.70). For simplicity, we consider the map to send ξ to an interval on the real line. Under suitable conditions, this map may be inverted to yield

$$\xi = \Gamma(x) \ .$$

If we substitute $\Gamma(x)$ for ξ in Eqs. (11.4), do we get basis functions $N_k(x)$ that are quadratic in x? Provided this is the case, we can answer our original question: if we interpolate three (distinct!) data points produced by a quadratic function f using quadratic basis functions, we will get back precisely f: The quadratic curve passing through these three data points is unique.

What are the conditions for the basis functions to be quadratic in x? Expanding the map (12.19), we obtain

$$x = N_1(\xi)x_1 + N_2(\xi)x_2 + N_3(\xi)x_3 = \frac{\xi^2}{2}(x_1 + x_2 - 2x_3) + \frac{\xi}{2}(x_2 - x_1) + x_3 \ .$$

Now notice that when $(x_1 + x_2 - 2x_3) = 0$ (that is, when $x_3 = (x_1 + x_2)/2$: the node x_3 is the midpoint of the interval), the map will be linear

$$x = \frac{\xi}{2}(x_2 - x_1) + x_3 \ .$$

For such a linear map, $\Gamma(x)$ is also linear in x, and an expression that is quadratic in ξ (namely the basis functions (11.4)), will be quadratic in x when we substitute for ξ. Therefore, if the node x_3 is the midpoint of the interval $x_1 \leq x \leq x_2$, the complete *quadratic function* $f(x) = ax^2 + bx + c$ will be *reproduced exactly* by the finite element interpolation on the element L3

$$\Pi_h f(x) = \sum_{k=1}^{M} N_k(x) f(x_k) = f(x) \ .$$

Another way of expressing the restriction on the form of the map is to say that the Jacobian must be constant (and positive).

No such restriction is required if we're interested in reproducing only linear functions $f(x) = bx + c$. The degrees of freedom are set as

$$f_k = bx_k + c \ ,$$

and we obtain

$$\Pi_h f(x) = \sum_{k=1}^{M} N_k(x) f_k = \sum_{k=1}^{M} N_k(x) \left[bx_k + c \right] \ .$$

The interpolation of the Cartesian coordinates (8.25) gives

$$\sum_{k=1}^{M} N_k(x) b x_k = b \sum_{k=1}^{M} N_k(x) x_k = bx \ ,$$

and the partition of unity property (8.24) yields

$$\sum_{k=1}^{M} N_k(x) c = c \sum_{k=1}^{M} N_k(x) = c \ .$$

Therefore, for $f(x)$ linear, we get $\Pi_h f(x) = f(x)$.

These observations may be generalized to all the elements discussed in this book: because all are isoparametric, linear functions may be reproduced exactly by interpolating on the mesh. Furthermore, provided the mapping from the standard shape to the element shape in the physical coordinates has a constant Jacobian, and provided the number of parametric and physical coordinates match, polynomials that are included in the basis functions on the standard shape will be reproduced also in the physical coordinates.

Exercise 65. Repeat Exercise 62. Perturb the location of the interior nodes of each L3 finite element by shifting it to the left by $L/15$ (which is $1/5$ of the length of the element): see Figure 12.22.

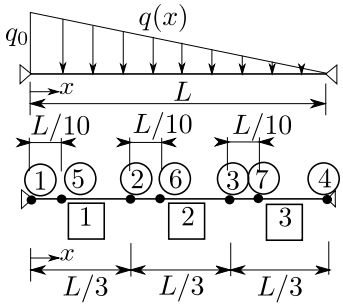


Fig. 12.22. Prestressed wire with linearly varying transverse load. The finite element mesh of three L3 Finite elements of equal length.

Solution: The solution proceeds exactly as in Exercise 62. The big difference is that the Jacobian is no longer constant. We show this by setting the location of the interior node to be offset from the midpoint. First we define the symbolic variables, `xK`, `xL`, `xM` as the locations of the end nodes and the interior node respectively; `xi` is the parametric coordinate.

```
syms xK xL xM xi  real
```

The array of the locations of the nodes is defined as

```
x=[xK;xL;xK+3/10*(xL-xK)];
```

where we note that the third node was displaced from the midpoint of the element. We compute the Jacobian.

```
>> J=x'*gradNxi;
J=collect(simplify(J),xi)
J =
((2*xL)/5 - (2*xK)/5)*xi + xL/2 - xK/2
```

Importantly, the Jacobian is no longer a constant. It is a linear function of the parametric coordinate. Consequently, all integrals will be affected since in all integral formulas we need the Jacobian.

The rest of the calculation needs to be carried out without the neat formulas for the load vector and the stiffness matrix: those were applicable in Exercise 62 for a constant Jacobian, but not here.

As before, we arrive at the finite element solution which is compared with the analytical solution in Figure 12.23. The difference is now clearly visible: The element that used to be so accurate now delivers a rather poor solution.

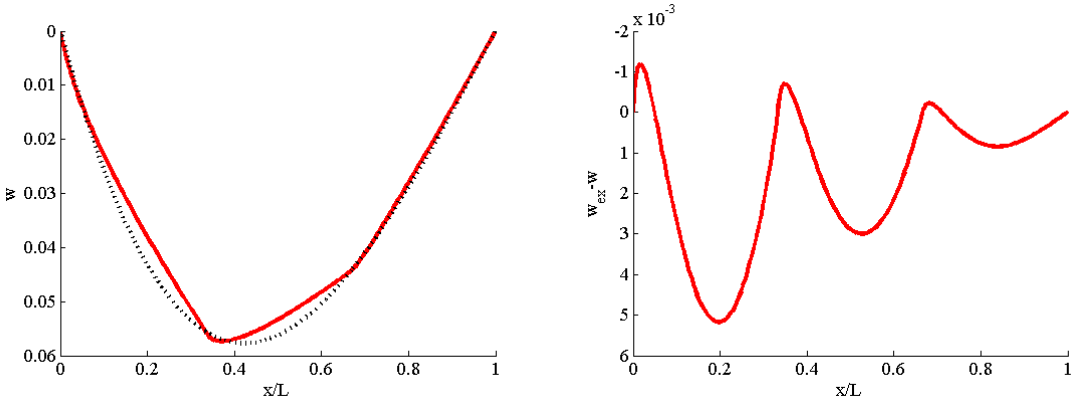


Fig. 12.23. Prestressed wire with linearly varying transverse load. Irregular location of the interior node. The deflection on the left, dotted curve exact, solid curve approximate. Error of the deflection on the right.

Plotting the true displacement error as the difference between the analytical deflection and finite element solution brings out the features of the error (Figure 12.23). It is now also hard to tell what kind of polynomial the deflection error looks like. It could be a quadratic with a bit of cubic thrown in.

The transverse force is computed as before as $S = Pw'$. The gradients of the basis functions need to be computed from (8.53). The Jacobian was shown above to be no longer constant, but linear in ξ, and hence the gradients with respect to x are not linear functions of ξ anymore. Rather they are rational functions:

```
gradN =
 -(xi - 1/2)/(xK/2 - xL/2 + xi*((2*xK)/5 - (2*xL)/5))
 -(xi + 1/2)/(xK/2 - xL/2 + xi*((2*xK)/5 - (2*xL)/5))
      (2*xi)/(xK/2 - xL/2 + xi*((2*xK)/5 - (2*xL)/5))
```

This has profound implications for the transverse force. The error of the transverse force is shown in Figure 12.24. The analytical solution is not matched well anymore.

In the following exercise we investigate the behavior of the L3 finite element applied to a problem which it can, under certain conditions, solve exactly.

Exercise 66.

Research the effect of the offset of the interior node from the midpoint of the element for uniform distributed transverse load with the mesh of three L3 finite elements.

Solution: It may have been sobering to see the effect of the nonuniform Jacobian on the performance of the L3 finite element for Exercise 65, but it gets worse here. While for the linearly varying transverse load the finite element model could not supply an exact solution, it can do so for uniform load. As shown in Figure 12.25 for the deflection and in Figure 12.26 for the transverse force the finite element model solves exactly (to within numerical precision) the boundary value problem on the condition the interior nodes are at the midpoint of the elements (which means that

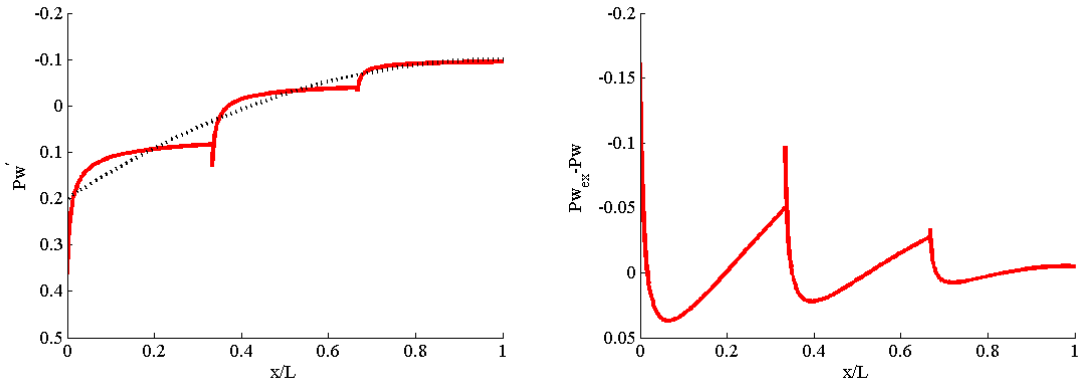


Fig. 12.24. Prestressed wire with linearly varying transverse load. Irregular location of the interior node. The transverse force on the left, dotted curve exact, solid curve approximate. Error of the transverse force on the right.

the Jacobians are constant). Why can the L3 element solve the problem exactly? Because for the constant Jacobian it can exactly reproduce quadratic functions (deflections) along its length. Since the exact solution is a quadratic function, the finite element will reproduce it without error. (The script `w129b_qu_l3_irreg`[7] solves this problem; set the variable `node_shift` to get either a regular or irregular mesh.)

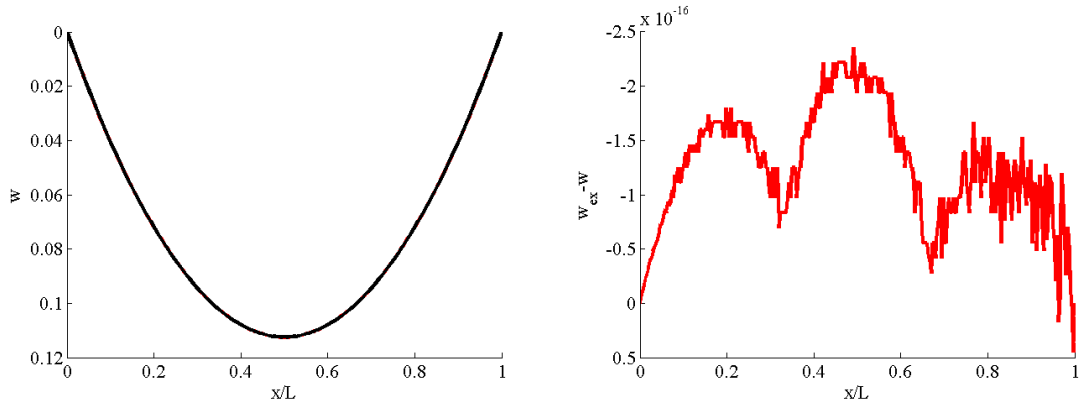


Fig. 12.25. Prestressed wire with uniform transverse load, three quadratic (L3) finite elements. The interior node at the midpoint. The deflection curve on the left, dotted curve exact, solid curve approximate. Error of the deflection on the right (note that the errors are on the order of machine epsilon).

If we use the same mesh except that we shift the interior nodes by 1/5 of the length of the element from the midpoint to the right, the situation drastically changes. As shown in Figure 12.27 for the deflection and in Figure 12.28 for the transverse force the finite element model definitely does not solve the problem exactly anymore. Compared to the machine precision attained previously with the regular mesh that errors are now huge. This is a clear indication that the L3 finite element can no longer represent exactly quadratic functions. Since the exact solution is a quadratic function, the finite elements cannot match it when they are distorted (i.e. when the Jacobian is nonconstant).

[7] Folder: `FAESOR/examples/taut_wire`

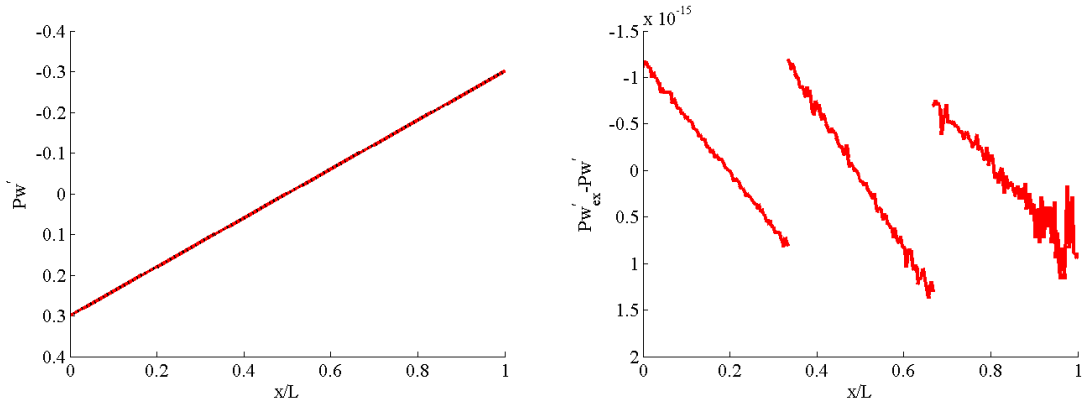


Fig. 12.26. Prestressed wire with uniform transverse load, three quadratic (L3) finite elements. The interior node at the midpoint. The transverse force on the left, dotted curve exact, solid curve approximate. Error of the transverse force on the right (note that the errors are close to machine precision).

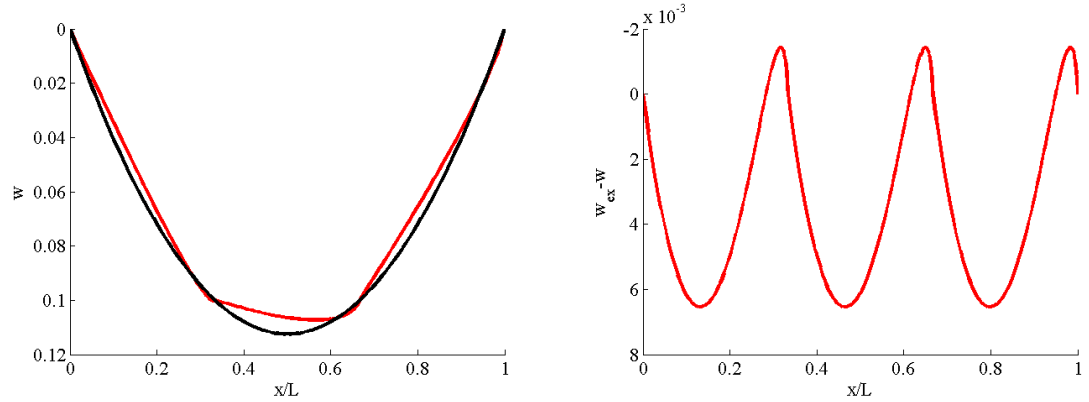


Fig. 12.27. Prestressed wire with uniform transverse load, three quadratic (L3) finite elements. The interior node offset by 1/5 of the length of the element from the midpoint to the right. The deflection curve on the left, dotted curve exact, solid curve approximate. Error of the deflection on the right (note that the errors are on the order of machine epsilon).

Exercises

1. Use the Matlab script `circle_area`[8] to compute the approximate area of the circle by integrating over a mesh of triangles.
   a) Derive the error as an order-of estimate in terms of the mesh size, $E_A \approx O(h^\beta)$, where h is the mesh size (length of a typical mesh edge).
   b) Relate the error estimate to the experimental data in the graph of the (log of) error versus the (log of) mesh size.
2. Modify the Matlab script `circle_area`[9] from assignment (1) to compute the location of the centroid of the circle by integrating over a mesh of triangles.
3. Modify the Matlab script `circle_area`[10] to compute all the moments of inertia (seconds moments) of the circle by integrating over a mesh of triangles.

[8] Folder: `FAESOR/examples/miscellaneous`
[9] Folder: `FAESOR/examples/miscellaneous`
[10] Folder: `FAESOR/examples/miscellaneous`

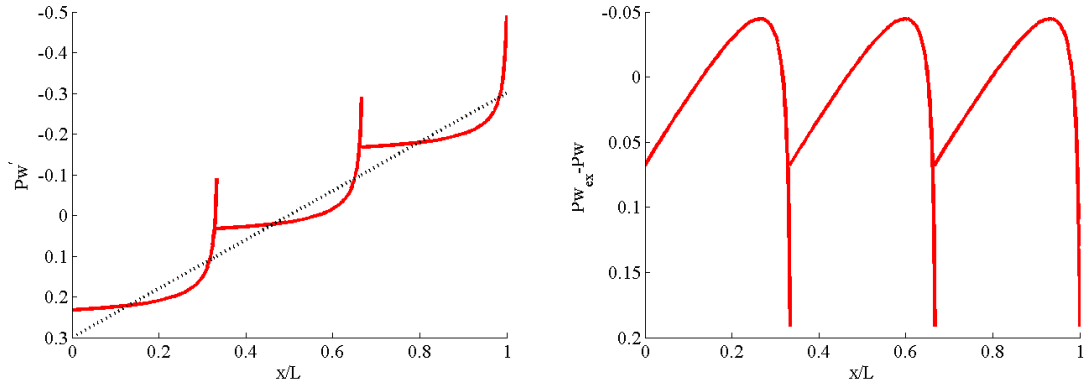


Fig. 12.28. Prestressed wire with uniform transverse load, three quadratic (L3) finite elements. The interior node offset by 1/5 of the length of the element from the midpoint to the right. The transverse force on the left, dotted curve exact, solid curve approximate. Error of the transverse force on the right (note that the errors are close to machine precision).

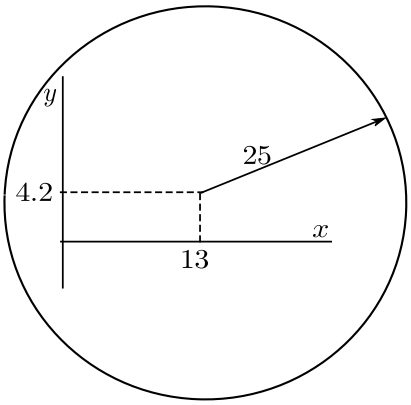


Fig. 12.29. Circle offset with respect to the coordinates xy

a) Derive the error as an order-of estimate in terms of the mesh size, $E_A \approx O(h^\beta)$, where h is the mesh size (length of a typical mesh edge).
b) Relate the error estimate to the experimental data in the graph of the (log of) error versus the (log of) mesh size.

4. Repeat the process of assignment (3), but using a three-point quadrature (`tri_rule(3)`).
   a) Derive the error as an order-of estimate in terms of the mesh size, $E_A \approx O(h^\beta)$, where h is the mesh size (length of a typical mesh edge).
   b) Relate the error estimate to the experimental data in the graph of the (log of) error versus the (log of) mesh size.
   c) Explain the difference between the results obtained with the two different quadrature rules.

13

Model of Elastodynamics

We can consider a deformable body to be a collection of particles, and apply the Newton's equation of motion of elementary dynamics to each particle, $m\dot{\boldsymbol{v}} = \boldsymbol{F}$, where $\dot{\boldsymbol{v}}$ is the particle acceleration, m is the particle mass, and $\boldsymbol{F}$ is the applied force. The complicating circumstance is that a deformable body can be thought of as a collection (of infinitely many) particles, all interacting through contact. Furthermore, our goal is to formulate a continuum model rather than deal with the discrete collection of particles.

13.1 Balance of linear momentum

Let us consider a body with some distributed force on parts of the boundary (the reactions must be included) and distributed force in the volume (for instance, gravity-induced load). For simplicity, we draw a sketch in two dimensions, but obviously we are thinking of a three-dimensional body; see Fig. 13.1. The distributed force on the boundary is therefore in units force/length^2, and units of the distributed force in the volume are force/length^3. The distributed force on the boundary is customarily called the ***traction***.

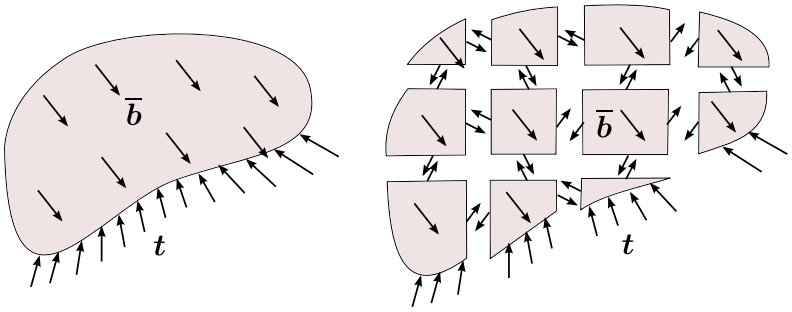


Fig. 13.1. A continuous body with applied distributed force on the boundary, and within the volume (on the left). The same body cut up into many small volumes (particles), with their interaction represented by distributed forces along the cuts (on the right).

The continuous body will be now divided into many very small (infinitesimally small) volumes, which we may consider "particles". The interaction between the particles is mediated by contact forces (tractions) along the cuts between the particles. Assuming we know these forces, the Newton's equation may be applied to each separately. However, we will apply this equation in the form of the change of ***linear momentum***

$$\frac{\mathrm{d}}{\mathrm{d}t}(m\boldsymbol{v}) = \boldsymbol{F} \ ,$$

from which the previous form of the equation of motion may be obtained provided m does not change. In our case, this will be true because each small volume holds a certain amount of material and does not exchange material with any other volume, so the mass of each volume is conserved.

As a consequence of the above, we may write for each small particle volume j the change of its linear momentum

$$\frac{\mathrm{d}}{\mathrm{d}t}(m_j \boldsymbol{v}_j) = \boldsymbol{F}_j \ , \tag{13.1}$$

where we use for the mass of the particle $m_j = \rho V_j$, with V_j the volume of the particle, and ρ the mass density, $\boldsymbol{v}_j$ the velocity, all at some point within the volume of the particle (we are using the mean-value theorem to express integrals over the volume of the particle!). The force $\boldsymbol{F}_j$ includes the body force $\overline{\boldsymbol{b}}$ and the tractions $\boldsymbol{t}$ on the surface of the particle volume

$$\boldsymbol{F}_j = \overline{\boldsymbol{b}} V_j + \int_{S_{\mathtt{int}}} \boldsymbol{t} \mathrm{d}S + \int_{S_{\mathtt{ext}}} \boldsymbol{t} \mathrm{d}S \ , \tag{13.2}$$

where the surface integral is split into two parts (see Fig. 13.2): the interior surfaces $S_{\mathtt{int}}$, where two particle volumes are separated, and the exterior surfaces $S_{\mathtt{ext}}$.

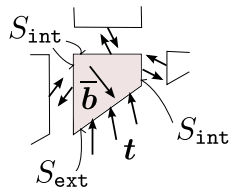


Fig. 13.2. Isolated particle volume.

Now we will collect the contributions of Eq. (13.1) by summing over all the particles

$$\sum_{j=1}^{N} \frac{\mathrm{d}}{\mathrm{d}t}(m_j \boldsymbol{v}_j) = \sum_{j=1}^{N} \boldsymbol{F}_j \ , \tag{13.3}$$

which may be rewritten in the limit of infinitely many particles as integrals

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_m \boldsymbol{v} \mathrm{d}m = \int_V \boldsymbol{b} \mathrm{d}V + \int_{S_{\mathtt{ext}}} \boldsymbol{t} \mathrm{d}S + \sum_{j=1}^{\infty} \int_{S_{\mathtt{int},j}} \boldsymbol{t} \mathrm{d}S \ , \tag{13.4}$$

where the last term (the sum) is over all the shared surfaces that separate the particle volumes. Using Newton's third law of action and reaction, we may conclude that whenever two particle volumes share a piece of their boundary, the traction at the material point A on the surface of particle 1 is equal in magnitude but opposite to the traction at the same material point (the one that has been split by the cut separating the two particles) at the corresponding point A on the surface of particle 2. Since the sum is over all the *pairs* of such surfaces, the last term in Eq. (13.4) cancels, and the final statement of the ***balance of linear momentum*** of the material in the volume V reads

$$\frac{\mathrm{d}}{\mathrm{d}t} \int_m \boldsymbol{v} \ \mathrm{d}m = \int_V \boldsymbol{b} \ \mathrm{d}V + \int_S \boldsymbol{t} \ \mathrm{d}S \ , \tag{13.5}$$

where m is the total mass of the material inside the volume V, and S is the bounding surface of the volume V. While the surface S and the volume V change with deformation, and hence are time-dependent, the total mass of the material m does not change (the same particles that were inside the volume before deformation are there during the deformation).

13.2 Stress

The traction vector $\boldsymbol{t}$ may be written in terms of components in a surface-aligned Cartesian basis as $\boldsymbol{t} = t_n\boldsymbol{n} + t_1\boldsymbol{e}_1 + t_2\boldsymbol{e}_2$, where t_n is the normal component, and t_k are the shear components. The Cartesian basis is defined at the given point on the surface by first taking the (outer, unit) surface normal as the third basis vector, and then picking arbitrary orthogonal directions in the tangent plane– see Fig. 13.3. The normal component is extracted as

$$t_n = \boldsymbol{n} \cdot \boldsymbol{t} \ . \tag{13.6}$$

The shear part of the traction $\boldsymbol{t}_s$ is obtained by subtracting the normal part of the traction from the traction vector $\boldsymbol{t}$

$$\boldsymbol{t}_s = \boldsymbol{t} - t_n\boldsymbol{n} \ . \tag{13.7}$$

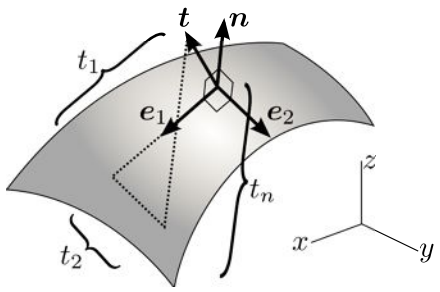


Fig. 13.3. Components of traction.

Next we need to relate the traction on the surface to the deformation of the material just below the surface. The deformation will be measured by strains, and the response of the material to the strains will be related to the tractions on the surface (and any body loads, if present) through the mathematical device of the ***stress***.

First, inspect Fig. 13.4: it is possible to define such a Cartesian coordinate system in the vicinity of a given point that the coordinate planes will cut out a (curvilinear) tetrahedron from the solid. Our plan is to make this tetrahedron very small indeed, but to still contain the given point on the surface. An enlarged image of such a tetrahedron is shown on the right, and we see how the curved edges may be approximated by straight lines in the limit of a very small tetrahedron. The goal is to relate the traction at the given point to the tractions on the internal cut planes, because these tractions are representations of the stress in the volume.

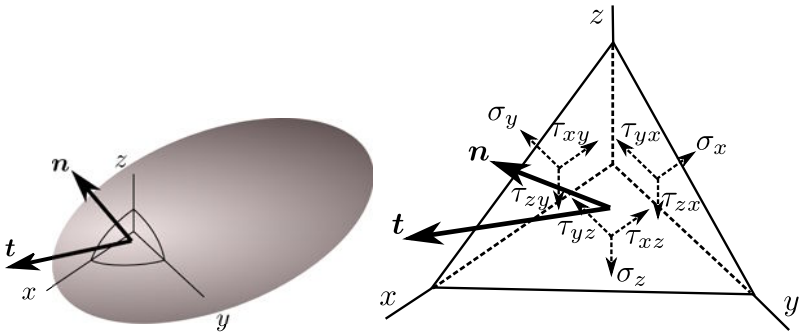


Fig. 13.4. Relating the components of traction to stress.

In anticipation of the definition of stress, the traction components on the three flat cut planes, with normals pointing against the three Cartesian basis vectors, are called σ_x, σ_y, σ_z (the normal

components), and τ_{xy}, τ_{yx}, τ_{xz}, τ_{zx}, τ_{yz}, τ_{zy}, for the shear components on all three planes. The areas of the triangular faces of the tetrahedron are related as $A_x = n_x A$, and so forth, where n_x, n_y, n_z are the components of the unit normal, and A_x is the area perpendicular to the x-axis and so on; this can be deduced from the volume of the tetrahedron in Fig. 13.5 written in terms of the heights d, i_x, i_y, i_z, and the corresponding areas.

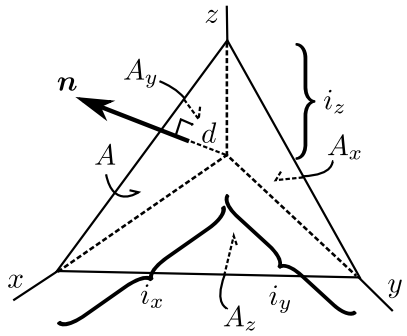


Fig. 13.5. Components of traction.

When we write the conditions of equilibrium in all three directions (the volume forces do not play a role; why?), the following three equations result

$$\begin{aligned} t_x &= \sigma_x n_x + \tau_{xy} n_y + \tau_{xz} n_z \ , \\ t_y &= \tau_{yx} n_x + \sigma_y n_y + \tau_{yz} n_z \ , \\ t_z &= \tau_{zx} n_x + \tau_{zy} n_y + \sigma_z n_z \ . \end{aligned} \tag{13.8}$$

This equation relates the components of the traction on the surface with the components of the traction on the special surfaces – coordinate planes – inside the volume. The components of the traction on the internal surfaces are called ***normal stresses*** (σ_x, σ_y, σ_z), and ***shear stresses*** (τ_{xy}, τ_{yx}, τ_{xz}, τ_{zx}, τ_{yz}, τ_{zy}). The form of Eq. (13.8) suggests the matrix expression

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} \ , \tag{13.9}$$

where all matrices hold components in the Cartesian basis. A component-free version would read

$$\boldsymbol{t} = \boldsymbol{\Sigma} \cdot \boldsymbol{n} \ ,$$

where $\boldsymbol{\Sigma}$ would be defined as a Cartesian tensor, the ***Cauchy stress tensor***. The traction vector and the normal would then also become tensors. However, in this book the tensor notation is avoided, and with a few exceptions tensors will not be needed. The two exceptions that may be mentioned here are coordinate transformations and the calculation of the ***principal stresses*** which are the eigenvalues of the matrix of the stress components.

First we will consider transformation of vectors from one Cartesian coordinate system into another. The Cartesian coordinate axes are defined by a triple of orthonormal basis vectors, one by the basis triple $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ and the other by the basis triple $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$. An arbitrary vector $\boldsymbol{u}$ may be written in terms of components and basis vectors in either coordinate system as

$$\boldsymbol{u} = u_x \boldsymbol{e}_x + u_y \boldsymbol{e}_y + u_z \boldsymbol{e}_z = u_{\overline{x}} \boldsymbol{e}_{\overline{x}} + u_{\overline{y}} \boldsymbol{e}_{\overline{y}} + u_{\overline{z}} \boldsymbol{e}_{\overline{z}} \ . \tag{13.10}$$

This may be written as a matrix expression, with the basis vectors as columns of matrices

$$\boldsymbol{u} = \begin{bmatrix} \boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z \end{bmatrix} \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}} \end{bmatrix} \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \ . \tag{13.11}$$

Now consider that the basis vectors themselves may be expressed in terms of their components on some basis: we will pick $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ as that basis. On this basis we can write the components of the vectors $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ themselves as

$$[\boldsymbol{e}_x] = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad [\boldsymbol{e}_y] = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad [\boldsymbol{e}_z] = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

We can now write, entirely in components,

$$\big[\,[\boldsymbol{e}_x], [\boldsymbol{e}_y], [\boldsymbol{e}_z]\,\big] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \left[\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} .$$

The components of the vectors $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ are

$$[\boldsymbol{e}_{\overline{x}}] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z \end{bmatrix}, \quad [\boldsymbol{e}_y] = \begin{bmatrix} \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z \end{bmatrix}, \quad [\boldsymbol{e}_z] = \begin{bmatrix} \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix},$$

where $\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x$ is the cosine of the angle between the two vectors $\boldsymbol{e}_{\overline{x}}$ and $\boldsymbol{e}_x$, and so on, so that we can write entirely in terms of components

$$\big[\,\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}\,\big] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = \left[\begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z \end{bmatrix}, \begin{bmatrix} \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z \end{bmatrix}, \begin{bmatrix} \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} \right] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} .$$

where we have defined the transformation matrix

$$[\boldsymbol{R}_m] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} , \tag{13.12}$$

composed of the direction cosines. In words, the columns of the transformation matrix consist of the components of the vectors $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ on the basis vectors $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$. The transformation matrix is recognized as an orthogonal matrix (rotation matrix), whose inverse is its transpose

$$[\boldsymbol{R}_m]^{-1} = [\boldsymbol{R}_m]^T$$

Thus we have the ***transformation of vector components***

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} . \tag{13.13}$$

Note that (13.12) is just a three-dimensional analog of the two-dimensional transformation (8.66).

Now we take up again the relationship between the traction vector, the stress at a point, and the normal to a surface through the point (13.9)

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} ,$$

and we consider what happens when both vectors are expressed using components in a different coordinate system. We apply (13.13) as

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix}$$

so that we can write

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} n_x \\ n_y \\ n_z \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} ,$$

or

$$[\boldsymbol{R}_m] \begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} ,$$

and finally or

$$\begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} .$$

Since one also has

$$\begin{bmatrix} t_{\overline{x}} \\ t_{\overline{y}} \\ t_{\overline{z}} \end{bmatrix} = \begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} \begin{bmatrix} n_{\overline{x}} \\ n_{\overline{y}} \\ n_{\overline{z}} \end{bmatrix} ,$$

we conclude that the ***stress tensor components transform*** when switching between coordinate systems as

$$\begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} = [\boldsymbol{R}_m]^T \begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} [\boldsymbol{R}_m] \tag{13.14}$$

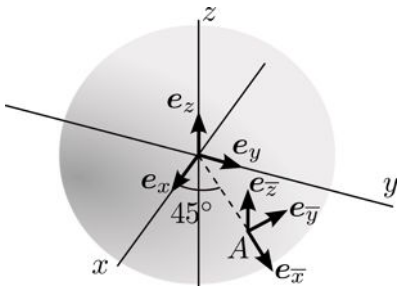
Of course, as for vectors this transformation also works back and forth so that

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} = [\boldsymbol{R}_m] \begin{bmatrix} \sigma_{\overline{x}} & \tau_{\overline{xy}} & \tau_{\overline{xz}} \\ \tau_{\overline{yx}} & \sigma_{\overline{y}} & \tau_{\overline{yz}} \\ \tau_{\overline{zx}} & \tau_{\overline{zy}} & \sigma_{\overline{z}} \end{bmatrix} [\boldsymbol{R}_m]^T . \tag{13.15}$$

Exercise 67. The stress components are given at the point A on the surface of the sphere in the Cartesian coordinate system with basis $\boldsymbol{e}_x, \boldsymbol{e}_y, \boldsymbol{e}_z$ as

$$[\sigma] = \begin{bmatrix} -350, & -150, & 0 \\ -150, & -350, & 0 \\ 0, & 0, & -200 \end{bmatrix}$$

Transform the stress components to the Cartesian coordinate system $\boldsymbol{e}_{\overline{x}}, \boldsymbol{e}_{\overline{y}}, \boldsymbol{e}_{\overline{z}}$ which is defined so that $\boldsymbol{e}_{\overline{x}}$ is in the xy plane, at an angle of 45° from $\boldsymbol{e}_x$, and $\boldsymbol{e}_{\overline{z}} = \boldsymbol{e}_z$.



Solution: The transformation (13.14) needs to be applied. Therefore, the transformation matrix $[\boldsymbol{R}_m]$ is needed. The basis vector $\boldsymbol{e}_{\overline{x}}$ and $\boldsymbol{e}_{\overline{z}}$ follow the definition

```
exb=[sqrt(2)/2; sqrt(2)/2; 0];   ezb=[0; 0; 1];
```

The third vector is obtained using the vector cross product

```
eyb=cross(ezb,exb)
eyb =
  -0.707106781186548
   0.707106781186548
                   0
```

Therefore we can write $[\boldsymbol{R}_m]$ with these three basis vectors as columns

```
Rm= [exb,eyb,ezb];
```

and the transformed stress components are obtained from the formula as

```
>> sig= [-350 -150         0
         -150 -350         0
         0       0      -200];
sigb= Rm'*sig*Rm
sigb =
  1.0e+002 *
  -5.000000000000001                   0                   0
                   0  -2.000000000000000                   0
                   0                   0  -2.000000000000000
```

This means that the shear stresses $\tau_{\overline{xy}}, \tau_{\overline{xz}}, \tau_{\overline{yz}}$ are all zero, and the normal stresses are $\sigma_{\overline{x}} = -500, \sigma_{\overline{y}} = -200, \sigma_{\overline{z}} = -200$.

The ***principal directions*** are normals to such special surfaces that the traction vector acting on the surface has only a normal component. We write this statement as

$$\begin{bmatrix} \widehat{t}_x \\ \widehat{t}_y \\ \widehat{t}_z \end{bmatrix} = \widehat{\sigma} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} \ . \tag{13.16}$$

In words, the traction vector $[\widehat{t}]$ has the same direction as the normal to the surface $[\widehat{n}]$ and (signed) magnitude $\widehat{\sigma}$. Substituting for the traction vector from (13.9) leads to

$$\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} = \widehat{\sigma} \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix} \ , \tag{13.17}$$

This is a standard eigenvalue problem, and for the solution to exist one further equation must be true

$$\det \left(\begin{bmatrix} \sigma_x & \tau_{xy} & \tau_{xz} \\ \tau_{yx} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{zy} & \sigma_z \end{bmatrix} - \widehat{\sigma} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \right) = 0 \ . \tag{13.18}$$

This equation is equivalent to the search for the roots of the cubic characteristic equation

$$-\widehat{\sigma}^3 + C_2\widehat{\sigma}^2 + C_1\widehat{\sigma} + C_0 = 0$$

where $C_2 = (\sigma_x + \sigma_y + \sigma_z)$, $C_1 = (\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2 - \sigma_x\sigma_y - \sigma_x\sigma_z - \sigma_y\sigma_z)$, $C_0 = -\sigma_z\tau_{xy}^2 + 2\tau_{xy}\tau_{xz}\tau_{yz} - \sigma_y\tau_{xz}^2 - \sigma_x\tau_{yz}^2 + \sigma_x\sigma_y\sigma_z$. Consequently, this being a cubic equation there must be three such roots – ***principal stresses***–which we denote

$\widehat{\sigma}_j \quad j = 1, 2, 3\,,$

and three mutually orthogonal ***principal directions*** (eigenvectors)

$$\begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_j \qquad j = 1, 2, 3\,.$$

The matrix of the stress components being real and symmetric guarantees that the roots of the characteristic equation are always real, and also the existence of the three eigenvectors that are orthonormal is guaranteed.

Exercise 68. Find the principal stresses of the stress matrix

$$[\sigma] = \begin{bmatrix} -350, & -150, & 0 \\ -150, & -350, & 0 \\ 0, & 0, & -200 \end{bmatrix}$$

Solution: While there are formulas for the roots of cubic equations, in finite element programs principal stresses are normally computed numerically and here we shall solve the eigenvalue problem using numerical methods built-in into the Matlab `eig` solver:

```
>> sig= [-350.0000 -150.0000         0
      -150.0000 -350.0000         0
         0         0      -200.0000];
     [V,D] =eig(sig)
V =
   0.707106781186547                   0  -0.707106781186547
   0.707106781186547                   0   0.707106781186547
                   0   1.000000000000000                   0
D =
  1.0e+002 *
  -5.000000000000000                   0                   0
                   0  -2.000000000000000                   0
                   0                   0  -2.000000000000000
```

The diagonal of `D` holds the principal stresses. By convention in continuum mechanics these should be ordered $\widehat{\sigma}_1 \geq \widehat{\sigma}_2 \geq \widehat{\sigma}_3$. Therefore, since `D(2,2)` is equal to `D(3,3)` we can set for instance

$$\widehat{\sigma}_1 = -200\,, \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\widehat{\sigma}_2 = -200\,, \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_2 = \begin{bmatrix} -0.707106781186547 \\ 0.707106781186547 \\ 0 \end{bmatrix}$$

and

$$\widehat{\sigma}_3 = -500\,, \quad \begin{bmatrix} \widehat{n}_x \\ \widehat{n}_y \\ \widehat{n}_z \end{bmatrix}_2 = \begin{bmatrix} 0.707106781186547 \\ 0.707106781186547 \\ 0 \end{bmatrix}$$

13.2.1 Balance of angular momentum and stress symmetry.

It would appear that there are nine components of stress that need to be related to the deformation, but it is straightforward to show that in the matrix (13.9) the elements reflected with respect to the diagonal must be equal: Consider a rectangular volume of material (again, for convenience the drawing in Fig. 13.6 is of a two-dimensional nature, but the argument applies to three dimensions). When the ***balance of angular momentum*** is written for the rotation about the axis perpendicular to the plane of the paper, the normal stresses and any body forces will turn out to be negligible compared to to the effect of the shear stresses, and from the resultant equation we obtain the symmetries

$$\tau_{xy} = \tau_{yx} \ , \quad \tau_{xz} = \tau_{zx} \ , \quad \tau_{yz} = \tau_{zy} \ . \tag{13.19}$$

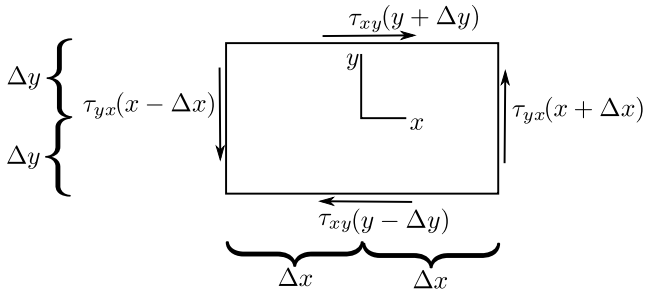


Fig. 13.6. Components of traction.

Consequently, there are only six components of the stress that are independent. It will be convenient to manipulate these six components as a vector (as opposed to a tensor)

$$[\boldsymbol{\sigma}] = [\sigma_x, \sigma_y, \sigma_z, \tau_{xy}, \tau_{xz}, \tau_{yz}]^T \ . \tag{13.20}$$

Equation (13.8) may be rewritten in terms of the stress vector $\boldsymbol{\sigma}$ as

$$\boldsymbol{t} = \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} \ , \tag{13.21}$$

where the matrix "***vector-stress vector dot product***" operator is defined as

$$\mathcal{P}_{\boldsymbol{n}} = \begin{bmatrix} n_x & 0 & 0 & n_y & n_z & 0 \\ 0 & n_y & 0 & n_x & 0 & n_z \\ 0 & 0 & n_z & 0 & n_x & n_y \end{bmatrix} \ . \tag{13.22}$$

Equation (13.21) may be used in a variety of ways: any of the three quantities may be given, which would then for another quantity being fixed produce the third as the result. Most useful are these two possibilities: $\boldsymbol{t}$ given, produce the stress vector in dependence on the normal; and $\boldsymbol{\sigma}$ given, produce the surface tractions for various normals.

Exercise 69.

Stress components at a point are given as

$$\sigma_x = 9 \ , \quad \sigma_y = -3 \ , \quad \sigma_z = 3 \ , \quad \tau_{xy} = -2 \ , \quad \tau_{xz} = 0 \ , \quad \tau_{yz} = 2$$

Find the magnitude of the normal and shear traction on the plane with normal $[n] = [2, 1, 2]^T/3$.

Solution: This can be done in two ways. Firstly we can organize the stress components in a square matrix and use equation (13.9). Secondly we can use (13.21) with the stress vector form.

The first approach: The traction vector is computed as

```
>> sig= [9,-2,0;-2,-3,2;0,2,3];
n= [2,1,2]'/3;
t=sig*n
t =
   5.333333333333333
  -1.000000000000000
   2.666666666666667
```

The normal component of the traction is

```
>> tn=dot(t, n)
tn =
     5
```

and the shear traction vector is

```
>> ts=t-tn*n
ts =
   2.000000000000000
  -2.666666666666666
  -0.666666666666667
```

The magnitude of the shear traction vector is

```
>> norm(ts)
ans =
   3.399346342395190
```

The second approach:

```
>> Pn= [n(1),0,0,n(2),n(3),0
    0,n(2),0,n(1),0,n(3)
    0,0,n(3),0,n(1),n(2)];
sigv=[9,-3,3,-2,0,2]';
t=Pn*sigv
t =
   5.333333333333333
  -1.000000000000000
   2.666666666666667
```

and from here on the same procedure as above is used with the same results.

Exercise 70. Stress components at a point are given as

$$\sigma_x = 9\,, \quad \sigma_y = -3\,, \quad \sigma_z = 3\,, \quad \tau_{xy} = -2\,, \quad \tau_{xz} = 0\,, \quad \tau_{yz} = 2$$

Find the largest possible shear stress at the given point. This means the maximum of the shear stress acting on all planes that can be passed through the point.

Solution: The maximum possible shear stress is computed from the principal stress values as

$$\tau_{\max} = \frac{\sigma_1 - \sigma_3}{2}$$

i.e. as half the difference between the largest and the smallest principal stress. The principal stress values are

```
>> sig= [9,-2,0;-2,-3,2;0,2,3];
>> [V,D] =eig(sig)
V =
  -0.147373562347511   0.096707445193820  -0.984341761363952
  -0.949880424611535   0.263565517348976   0.168108289517132
   0.275695868777197   0.959781687733545   0.053017919900119

D =
  -3.890784608594765                   0                   0
                   0   3.549219725105126                   0
                   0                   0   9.341564883489637
```

and the maximum shear stress is therefore found as $\tau_{\max} = (9.342 - -3.891)/2 = 6.616$. Notably the maximum shear stress appears in the Tresca failure criterion

$$\tau_{\max} \leq \frac{\sigma_{\mathrm{f}}}{2}$$

where σ_{f} is the failure stress of the material in uniaxial tension.

Ductile materials (such as metals) are often considered in failure analyses in relation to the stress quantity that describes the distortion of the volume of the material (no change in volume), the von Mises equivalent stress.

Exercise 71.

Stress components at a point are given as

$$\sigma_x = 9\ , \quad \sigma_y = -3\ , \quad \sigma_z = 3\ , \quad \tau_{xy} = -2\ , \quad \tau_{xz} = 0\ , \quad \tau_{yz} = 2$$

Find the Von Mises equivalent stress at the given point.

Solution: The von Mises equivalent stress may be computed from the principal stresses as

$$\sigma_{\mathrm{vm}} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_1 - \sigma_2)^2 + (\sigma_1 - \sigma_3)^2 + (\sigma_3 - \sigma_2)^2}$$

or directly from the components of the stress as

$$\sigma_{\mathrm{vm}} = \frac{1}{\sqrt{2}}\sqrt{(\sigma_x - \sigma_y)^2 + (\sigma_x - \sigma_z)^2 + (\sigma_z - \sigma_y)^2 + 6(\tau_{xy}^2 + \tau_{xz}^2 + \tau_{yz}^2)}$$

Substituting given stress components we obtain $\sigma_{\mathrm{vm}} = 11.489$. The von Mises equivalent stress is compared in the von Mises failure criterion

$$\sigma_{\mathrm{vm}} \leq \sigma_{\mathrm{f}}$$

with the failure stress of the material in uniaxial tension σ_{f}.

13.3 Local equilibrium

In complete analogy to the model of heat conduction, the global balance equation (13.4) (in this case, balance of linear momentum, for the heat conduction it was balance of heat energy (7.4)) needs to be converted to a local form. The local form expresses dynamic equilibrium of an infinitesimal particle as an equation that holds at a point.

13.3.1 Change of linear momentum

There are three terms in the global balance (13.4), and to produce the local form we'll have to convert all three integrals to volume integrals. The first one involves the time derivative of the integral

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\,\mathrm{d}m\ .$$

However, that causes no difficulties since the mass m inside the volume V does not change with time. Therefore,

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_m \boldsymbol{v}\,\mathrm{d}m = \int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\,\mathrm{d}m\ . \tag{13.23}$$

Introducing the ***mass density*** ρ (which as mass per unit volume depends on the deformation, and hence varies with time), we may write $\mathrm{d}m = \rho\mathrm{d}V$ and

$$\int_m \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\,\mathrm{d}m = \int_V \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\rho\,\mathrm{d}V\ . \tag{13.24}$$

13.3.2 Stress divergence

The divergence theorem may be now applied to the third term in (13.4), that is to the surface integral. However, if we introduce the abstract symbol for the divergence of stress by spelling out its definition, we generate more questions than answers. Therefore, it will be instructive to get to the needed form of the divergence theorem in a roundabout way.

Consider a small volume (parallelepiped) with faces parallel to coordinate planes of the global Cartesian basis (Fig. 13.7, and refer also to Fig. 13.1); for simplicity, the box is drawn as two-dimensional, and it is drawn twice so that we can display the normal and the shear stresses separately. The center of the box is at x, y, z, and the stress components may be expanded into a truncated Taylor series. For instance,

$$\sigma_x(x+\xi\Delta x, y+\eta\Delta y, z+\zeta\Delta z) \approx \sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\xi\Delta x + \frac{\partial\sigma_x(x,y,z)}{\partial y}\eta\Delta y + \frac{\partial\sigma_x(x,y,z)}{\partial z}\zeta\Delta z\ ,$$

where $-1 \le \xi \le +1$, $-1 \le \eta \le +1$, and $-1 \le \zeta \le +1$.

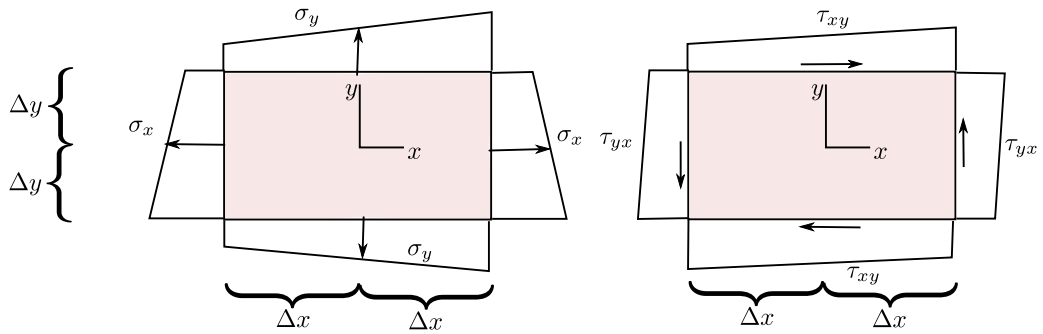


Fig. 13.7. Components of traction as functions of x, y along the sides of the box.

The box is loaded only by the tractions on its boundary, there are no body loads. Equilibrium in the x-direction requires integration of the stress σ_x over the vertical sides of the box, τ_{xy} over the

horizontal sides, and τ_{xz} over the faces parallel to the plane of the paper. For instance, integrating σ_x over the side at $\xi = 1$ leads to

$$\begin{aligned}\Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \sigma_x(x+\Delta x, y+\eta\Delta y, z+\zeta\Delta z)\,\mathrm{d}\eta\mathrm{d}\zeta \approx \\ \Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \left[\sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\Delta x \right. \\ \left. + \frac{\partial\sigma_x(x,y,z)}{\partial y}\eta\Delta y + \frac{\partial\sigma_x(x,y,z)}{\partial z}\zeta\Delta z\right]\,\mathrm{d}\eta\mathrm{d}\zeta\ .\end{aligned}$$

The terms with η and ζ integrate to zero, and the result is

$$4\Delta y\Delta z\left[\sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\Delta x\right]\ .$$

Next, integrating σ_x over the side at $\xi = -1$ leads to

$$\begin{aligned}\Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} -\sigma_x(x+\Delta x, y+\eta\Delta y, z+\zeta\Delta z)\,\mathrm{d}\eta\mathrm{d}\zeta \approx \\ \Delta y\Delta z \int_{-1}^{+1}\int_{-1}^{+1} \left[-\sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\Delta x \right. \\ \left. - \frac{\partial\sigma_x(x,y,z)}{\partial y}\eta\Delta y - \frac{\partial\sigma_x(x,y,z)}{\partial z}\zeta\Delta z\right]\,\mathrm{d}\eta\mathrm{d}\zeta\ .\end{aligned}$$

The terms with η and ζ integrate to zero, and the result is

$$4\Delta y\Delta z\left[-\sigma_x(x,y,z) + \frac{\partial\sigma_x(x,y,z)}{\partial x}\Delta x\right]\ .$$

Adding these two together gives the total contribution of the stress σ_x as

$$8\Delta x\Delta y\Delta z\frac{\partial\sigma_x(x,y,z)}{\partial x} = \Delta V\frac{\partial\sigma_x(x,y,z)}{\partial x}\ ,$$

with the elementary volume $\Delta V = 8\Delta x\Delta y\Delta z$. The same exercise is now repeated for the stress components τ_{xy} and τ_{xz}, giving the total force on the elementary volume in the x-direction

$$\Delta b_x^* = \Delta V\left[\frac{\partial\sigma_x(x,y,z)}{\partial x} + \frac{\partial\tau_{xy}(x,y,z)}{\partial y} + \frac{\partial\tau_{xz}(x,y,z)}{\partial z}\right]\ , \tag{13.25}$$

and analogously in the other two directions

$$\Delta b_y^* = \Delta V\left[\frac{\partial\tau_{yx}(x,y,z)}{\partial x} + \frac{\partial\sigma_y(x,y,z)}{\partial y} + \frac{\partial\tau_{yz}(x,y,z)}{\partial z}\right]\ , \tag{13.26}$$

and

$$\Delta b_z^* = \Delta V\left[\frac{\partial\tau_{zx}(x,y,z)}{\partial x} + \frac{\partial\tau_{zy}(x,y,z)}{\partial y} + \frac{\partial\sigma_z(x,y,z)}{\partial z}\right]\ . \tag{13.27}$$

Now the same argument that was established around Eq. (13.2) will be pursued: put together the total force on the body by collecting the contributions from all the elementary volumes. This can be done in two ways:

1. Add up all the tractions on the bounding faces of the elementary volumes. The tractions on the shared faces (internal surfaces) will cancel; only the tractions on the exterior surface will be left:

$$\int_S \boldsymbol{t}\,\mathrm{d}S$$

2. Add up all the resultant equivalent volume forces (13.25-13.27), which in the limit will become a volume integral

$$\int_V \boldsymbol{b}^* \, \mathrm{d}V$$

where the imaginary force $\boldsymbol{b}^*$ has components on the Cartesian basis

$$[\boldsymbol{b}^*] = \begin{bmatrix} \dfrac{\partial \sigma_x(x,y,z)}{\partial x} + \dfrac{\partial \tau_{xy}(x,y,z)}{\partial y} + \dfrac{\partial \tau_{xz}(x,y,z)}{\partial z} \\ \dfrac{\partial \tau_{yx}(x,y,z)}{\partial x} + \dfrac{\partial \sigma_y(x,y,z)}{\partial y} + \dfrac{\partial \tau_{yz}(x,y,z)}{\partial z} \\ \dfrac{\partial \tau_{zx}(x,y,z)}{\partial x} + \dfrac{\partial \tau_{zy}(x,y,z)}{\partial y} + \dfrac{\partial \sigma_z(x,y,z)}{\partial z} \end{bmatrix} \tag{13.28}$$

and may be recognized as the ***stress divergence***.

These two forces are equal, and we have the following form of the divergence theorem

$$\int_V \boldsymbol{b}^* \, \mathrm{d}V = \int_S \boldsymbol{t} \, \mathrm{d}S \; .$$

Using the template of the "vector-stress vector dot product" operator (13.22), we may write the stress divergence as

$$\boldsymbol{b}^* = \mathcal{B}^T \boldsymbol{\sigma} \; , \tag{13.29}$$

where the ***stress-divergence operator*** $\mathcal{B}^T$ is defined as

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & 0 & \partial/\partial y & \partial/\partial z & 0 \\ 0 & \partial/\partial y & 0 & \partial/\partial x & 0 & \partial/\partial z \\ 0 & 0 & \partial/\partial z & 0 & \partial/\partial x & \partial/\partial y \end{bmatrix} \; . \tag{13.30}$$

This operator (un-transposed) will make its appearance shortly yet again as the *symmetric gradient operator* to produce strains out of displacements. Using the definitions of both of these useful operators, the ***divergence theorem*** may be written in terms of stress as

$$\int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V = \int_S \mathcal{P}_{\boldsymbol{n}} \boldsymbol{\sigma} \, \mathrm{d}S \; . \tag{13.31}$$

13.3.3 All together now

Putting the three integrals from (13.4) into the volume-integral form leads to a pointwise expression of local equilibrium (following exactly the same argument as in Section 7.1):

$$\int_V \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \, \mathrm{d}V = \int_V \overline{\boldsymbol{b}} \, \mathrm{d}V + \int_V \mathcal{B}^T \boldsymbol{\sigma} \, \mathrm{d}V \quad \Rightarrow \quad \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \overline{\boldsymbol{b}} + \mathcal{B}^T \boldsymbol{\sigma} \; . \tag{13.32}$$

This is a statement of ***dynamic equilibrium*** of a point particle: On the left-hand side we have the inertial force (mass times acceleration), on the right-hand side is the body load and the force generated by a stress gradient across the particle. Analogously to the heat conduction problem, this local balance equation contains too many variables. The stress plays the role of the heat flux, and it also will be replaced by reference to measurable variables – the strains.

13.4 Strains and displacements

The strains measure the *relative* deformation, and based on the effect they represent when expressed in Cartesian coordinates, they may be divided into two groups: the normal strains (stretches), and the shear strains.

The strains are an expression of the local variations in the positions of material particles after deformation. The deformation (motion) is expressed as displacements. The ***displacement u*** is expressed in the Cartesian coordinates by components, and connects the locations of a given ***material point*** (particle) A before deformation and after deformation

$$[\boldsymbol{u}(A,t)] = \begin{bmatrix} x(A,t) \\ y(A,t) \\ z(A,t) \end{bmatrix} - \begin{bmatrix} x(A,0) \\ y(A,0) \\ z(A,0) \end{bmatrix} \ . \tag{13.33}$$

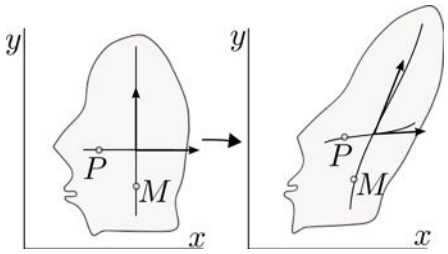


Fig. 13.8. Material curves, and tangents to material curves. Left: before deformation, right: after deformation.

It will be useful to approach the meaning of strains from the point of view of what happens to tangents to material curves during the deformation. A ***material curve*** consists of the same material points (particles) at any point in time. A visual may be useful: recall that some specimens have a square grid etched upon them before they are being mechanically tested (deformed). The etching curves that go in one direction may be thought of as sets of points whose one coordinate changes and the other is being held fixed. Figure 13.8 shows a blob of material with two material curves before and after deformation. Before deformation, the curve that is horizontal consists of points P such that the coordinates are

$$[P] = \begin{bmatrix} x \\ y = \text{constant} \end{bmatrix} \ ,$$

and the curve that is vertical consists of points M such that

$$[M] = \begin{bmatrix} x = \text{constant} \\ y \end{bmatrix} \ .$$

The parameter that varies along the curve through the point P is x. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \ . \tag{13.34}$$

The parameter that varies along the curve through the point M is y. Therefore, the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \ . \tag{13.35}$$

The tangent vectors (13.34) and (13.35) are of course just the basis vectors of the Cartesian coordinates.

After deformation, the curve that used to be horizontal consists of points P such that

$$[P] = \begin{bmatrix} x + u_x \\ (y = \text{constant}) + u_y \end{bmatrix} ,$$

and the curve that is vertical consists of points M such that

$$[M] = \begin{bmatrix} (x = \text{constant}) + u_x \\ y + u_y \end{bmatrix} .$$

Since these are material curves, they are still parameterized by the same parameters as before deformation. Consequently, for the originally horizontal curve we have the tangent vector after deformation

$$\frac{\partial}{\partial x}[P] = \begin{bmatrix} 1 + \dfrac{\partial u_x}{\partial x} \\ \dfrac{\partial u_y}{\partial x} \end{bmatrix} . \tag{13.36}$$

The parameter that varies along the curve through the point M is y. Therefore, after deformation the tangent vector to this curve is

$$\frac{\partial}{\partial y}[M] = \begin{bmatrix} \dfrac{\partial u_x}{\partial y} \\ 1 + \dfrac{\partial u_y}{\partial y} \end{bmatrix} . \tag{13.37}$$

The ***stretches*** measure the relative change in length of the tangent vectors at the same material point before and after deformation. For instance, the tangent vector (13.34) is of unit length before deformation, and the vector (13.36) is of length

$$\sqrt{(1 + \frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} = \sqrt{1 + 2\frac{\partial u_x}{\partial x} + (\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} .$$

If we now make the *assumption that the derivatives of the displacement components are very small* in magnitude,

$$|\frac{\partial u_k}{\partial j}| \ll 1, \quad k, j = x, y, z , \tag{13.38}$$

the length of the tangent vector may be expressed as

$$\sqrt{1 + 2\frac{\partial u_x}{\partial x} + (\frac{\partial u_x}{\partial x})^2 + (\frac{\partial u_y}{\partial x})^2} \approx 1 + \frac{\partial u_x}{\partial x} ,$$

and the relative change in length (the stretch in the x direction) is

$$\frac{1 + \dfrac{\partial u_x}{\partial x} - 1}{1} = \epsilon_x .$$

The ***shears*** measure the change in the angle between originally perpendicular directions of pairs of the Cartesian axes. Therefore, we could measure the change in the angle between the tangents of two intersecting material curves before and after deformation. For the two curves in Fig. 13.8, the initial angle is $\pi/2$; the cosine of the angle after the deformation is

$$\frac{\partial}{\partial x}[P]^T \frac{\partial}{\partial y}[M] = (1 + \frac{\partial u_x}{\partial x})\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}(1 + \frac{\partial u_y}{\partial y})$$

which, again using the assumption (13.38), gives for the change of the angle

$$\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x} = \gamma_{xy} \ .$$

In this way we define all six strain components: three stretches, and three shears. In fact, we could have defined nine strains (components of the ***strain tensor***), which would correspond to the nine components of the Cauchy stress tensor. However, we will stick to the vector representation in this book.

The six strain components are a mixture of the derivatives of the displacement components, and may be expressed in an operator equation, using the definition (13.29)

$$\boldsymbol{\epsilon} = \mathcal{B}\boldsymbol{u} \ , \tag{13.39}$$

where $\mathcal{B}$ is called the ***symmetric gradient*** (or strain-displacement) operator.

The components of the strain are

$$[\boldsymbol{\epsilon}]^T = [\epsilon_x, \epsilon_y, \epsilon_z, \gamma_{xy}, \gamma_{xz}, \gamma_{yz}] \tag{13.40}$$

Note the transpose, the strain vector has six rows and one column. The first three components are the stretches, the last three components are the shears.

13.5 Constitutive equation

The stress may now be replaced in the balance equation (13.32) by reference to the primary variable, the displacement. However, first we need to discuss the link between the measurable quantities, the strains, and the mathematical device in the balance equation, the stress. As for the thermal model, this link is the constitutive equation.

Since the angular momentum balance (13.19) reduces the number of stress components to six, correspondingly there are six components of strain. The energy of deformation may be defined as the work of each stress component on the corresponding strain component. Let us consider some pre-existing stressed state in a very small neighborhood of a given point. So that we don't have to specify the volume, we will refer to ***energy density*** (the energy in a certain volume may be obtained by integrating the energy density over this volume). The state of stress is described by the stress vector $\boldsymbol{\sigma}$. Let us superimpose an infinitesimal strain variation $\mathrm{d}\boldsymbol{\epsilon}$ upon the extant strains. The density of the work of the current stress on the strain change is expressed as

$$\mathrm{d}\boldsymbol{\epsilon}^T\boldsymbol{\sigma} \ . \tag{13.41}$$

The constitutive equation that will be of interest in this book is the model of ***linear elasticity***. It is expressed as a linear relationship between the strain and the stress, and since these are vectors, the linear relationship, the ***constitutive equation***, is expressed as a matrix product

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon} \ , \tag{13.42}$$

where $\boldsymbol{D}$ is a constant 6×6 matrix of the elastic coefficients (also known as the elasticities); $\boldsymbol{D}$ may be also referred to as the ***material stiffness*** matrix. Clearly, when there is no strain, the stress is zero. Let us now increase the strain from zero to its final value, $\boldsymbol{\epsilon}$, by scaling with a number $0 \leq \theta \leq 1$

$$\widehat{\boldsymbol{\epsilon}} = \theta\boldsymbol{\epsilon} \ ,$$

and furthermore use the linear elasticity (13.42). The expression for the change of the energy of deformation density (13.41) will become

$$\mathrm{d}\widehat{\boldsymbol{\epsilon}}^T\widehat{\boldsymbol{\sigma}} = \mathrm{d}\widehat{\boldsymbol{\epsilon}}^T\boldsymbol{D}\widehat{\boldsymbol{\epsilon}} = \mathrm{d}\theta\boldsymbol{\epsilon}^T\boldsymbol{D}\theta\boldsymbol{\epsilon} \ . \tag{13.43}$$

The deformation process starts at $\theta = 0$ and reaches its final stage at $\theta = 1$. In this process, the total energy density stored in the material is

$$\phi(\boldsymbol{\epsilon}) = \int_0^1 \mathrm{d}\theta \boldsymbol{\epsilon}^T \boldsymbol{D} \theta \boldsymbol{\epsilon} = \frac{1}{2} \boldsymbol{\epsilon}^T \boldsymbol{D} \boldsymbol{\epsilon} \ . \tag{13.44}$$

Mathematically, the expression $\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D} \boldsymbol{\epsilon}$ is known as a ***quadratic form***. One interesting property of the quadratic form is that the unsymmetrical part of the matrix $\boldsymbol{D}$ does not contribute to the energy:

$$\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D} \boldsymbol{\epsilon} = \left(\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D} \boldsymbol{\epsilon} \right)^T = \frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}^T \boldsymbol{\epsilon} \quad \Rightarrow \quad \frac{1}{2}\boldsymbol{\epsilon}^T (\boldsymbol{D} - \boldsymbol{D}^T) \boldsymbol{\epsilon} = 0 \ .$$

Because the energy of deformation is a fundamental quantity, from physical principles, and from the point of view of mathematical modeling, this is a very good reason for postulating a priori the ***symmetry of the material stiffness, $\boldsymbol{D} = \boldsymbol{D}^T$***.

At the moment, we will leave the material stiffness matrix unspecified, since a detailed discussion follows in Section 14.5.

13.6 Boundary conditions

Similarly to the heat conduction problem, at each point on the bounding surface a boundary condition is required. The boundary conditions may be in terms of the primary variable, the displacement, or in terms of the flux variable, the stress. For heat conduction, the boundary condition in terms of flux referred to the *normal* flux only, since the flux parallel to the surface is essentially impossible to control in physical experiments. Similarly, for elasticity the flux boundary condition will not attempt to prescribe all six components of stress, but rather the "projection" of stress, the traction.

A complicating circumstance is that the primary variable and the traction both have three components. Therefore, the surface of the solid needs to be considered three times as to the appropriate boundary condition, *once for each component*.

Selection of the appropriate boundary conditions is critical to successful modeling. Typically, the boundary conditions that are applied to our models are only approximations of the physical reality. Thus, the first guidelines for the application of boundary conditions will be based on *physical considerations*.

13.6.1 Example: concrete dam

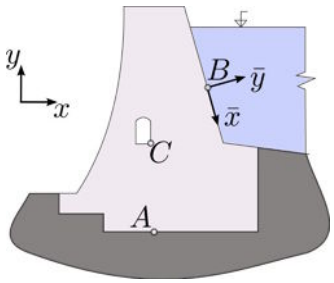


Fig. 13.9. Example of boundary conditions: concrete dam with a tunnel.

In the first example we will consider a concrete dam. Figure 13.9 shows the cross-section of a dam, and of interest is the stress near point C in the corner of the tunnel. Therefore, we could decide to neglect the deformation of the soil near the base of the dam, and prescribe zero magnitude for

all displacement components along the surface A. In reality, this is not strictly true, and a so-called *modeling error* is being introduced by making this choice. In a careful analysis, the influence of this error would be assessed, for instance by varying the boundary condition, or including the soil in the analysis.

On the surfaces exposed to the water behind the dam, including the one with point A, the structure is loaded by water *pressure*, which is a special kind of traction: using an ad hoc Cartesian coordinate system as indicated in the figure, the traction components are

$$t_{\bar{x}} = 0, \quad t_{\bar{y}} = p, \quad t_{\bar{z}} = 0$$

where p is the water pressure at the particular location.

All the other surfaces in the model that show up as curves, are assigned the so-called ***traction-free*** boundary condition: there are no known loads applied there. Furthermore, the model may be formulated using just two coordinates (reduced model of the so-called "plane strain" type): the remaining surfaces that are parallel to the plane of the paper will be assigned zero displacement normal to the paper, and zero shear components of traction in the plane of the paper. This type of model is discussed in detail later in the textbook.

Let us now examine the associated variable (so-called work-conjugate variable)– traction – along the surfaces where we prescribed displacements, for instance at point A. The physical meaning of such tractions, which are generated in the soil by the stress in the bulk of the dam near the surface, is clear: they are the ***reactions***. They are initially unknown, but as soon as the displacements are available from the solution, the reactions may be calculated.

The work-conjugate variable along the traction-free surfaces is displacement, which is initially unknown, but which will be produced during the solution process. Similarly, displacement is unknown on the surfaces exposed to the water behind the dam.

13.6.2 Example: rigid punch

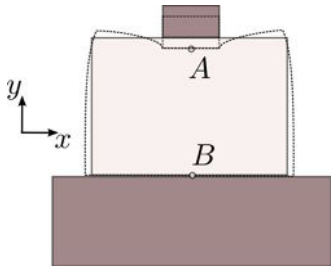


Fig. 13.10. Example of boundary conditions: rigid punch.

To model deformation under a stiff punch which is vertically pushed against a block of material, we may apply a set of approximate bilateral boundary conditions. (A further refinement would be a unilateral, contact, condition. But this is out of the range of this book.) Firstly, the punch may be assumed completely rigid, and perfect contact of the punch with the material underneath may be assumed. Also, perfect sticking or perfect slip under the punch may be assumed: the former when the surfaces in contact have a very high coefficient of friction, or perhaps they're bonded, the latter when the surfaces are lubricated. For perfect stick, we could prescribe the motion of the points on the conduct surface to be entirely driven by the punch: only vertical displacement, zero horizontal displacement. For perfect slip, the vertical motion is prescribed to be that of the punch, but a horizontal displacement under the punch is free. Therefore, for perfect slip we would apply the condition of zero shear traction under the punch.

13.6.3 Formal definition of the boundary conditions

At each point of the boundary we define a Cartesian coordinate system. It could be the surface-aligned system of Fig. 13.3, it could be the global system, or an arbitrarily oriented system. For instance, refer to Fig. 13.11 where each of the surfaces may have its own coordinate system.

Both the traction vector and the displacement vector at a given point may be written in such a coordinate system in terms of the components as

$$[\boldsymbol{t}] = \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad , \quad [\boldsymbol{u}] = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} \quad .$$

For each direction $i = x, y, z$ we separate the surface S into two disjoint parts:

1. $S_{t,i}$ where the traction component t_i is being prescribed;
2. $S_{u,i}$ where the displacement component u_i is being prescribed.

It holds that $S = S_{t,i} \bigcup S_{u,i}$, and it could be that either $S_{t,i} = \emptyset$ or $S_{u,i} = \emptyset$.

The boundary conditions may now be expressed as

$$t_i = (\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})_i = \overline{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \tag{13.45}$$

as the ***traction*** (natural) ***boundary condition*** for the ith component, and

$$u_i = \overline{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \tag{13.46}$$

as the ***displacement*** (essential) ***boundary condition*** for the ith component. When setting up a finite element model, note that the natural boundary condition need only be specified explicitly when $\overline{t}_i \neq 0$; the case of $\overline{t}_i = 0$ is implicitly active when we say nothing. In particular, no boundary condition needs to be explicitly defined for any component for a traction-free surface.

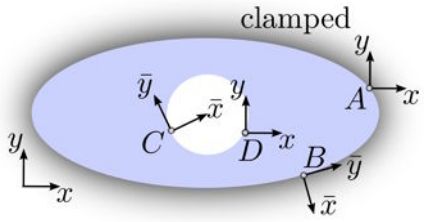


Fig. 13.11. Local coordinate systems used for boundary condition definitions.

13.6.4 Inadmissible "concentrated" boundary conditions

Consider that a resultant force of magnitude F is to be applied along the z-direction, that is perpendicularly to the surface shown in Fig. 13.12, as traction t_z applied to the area $\Delta x \Delta y$. As we need to have

$$F = t_z \Delta x \Delta y \ ,$$

if $\Delta x \to 0, \Delta y \to 0$, the traction component must approach infinity $t_z \to \infty$. In addition, since from the boundary conditions we have $\sigma_z = t_z$, we must conclude that in the immediate vicinity of the infinitesimal patch on the surface, at least some of the stresses must approach infinity as the traction component approaches infinity. The problem of a force applied to an infinite half space has been solved analytically by Boussinesq and others [S83], and perhaps the most significant conclusion is that the displacement under the force is infinite. Consequently, for any finite force, the *energy* in the system is *infinite*. As a consequence, we should remember the following caveats when using a *concentrated force as a boundary condition*:

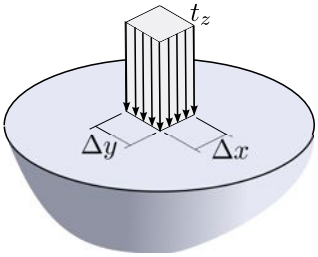


Fig. 13.12. Concentrated force as the limit of traction on infinitesimally small area.

1. Trying to obtain a converged solution for the displacement under the force or for the energy is pointless;
2. Displacements and stresses near the point of application of the force are most likely wrong for any purpose;
3. Displacements or stresses removed from the point of application of the force may be useful, but we have to always ask ourselves whether the concentrated force is truly needed or whether we use it only because we haven't thought the problem through.

Furthermore, as a corollary, we must conclude that if we apply a *displacement boundary condition at a point*, the associated reaction will be zero in the limit, which is wrong, unless we can guarantee for instance from global force equilibrium conditions that the reaction should be zero.

Very similar analysis may be performed for a *distributed load along a curve*: Figure 13.13. To maintain a finite value of the distributed load (force per unit length) as $\Delta y \to 0$, the traction t_z must approach infinity. The same list of caveats applies.

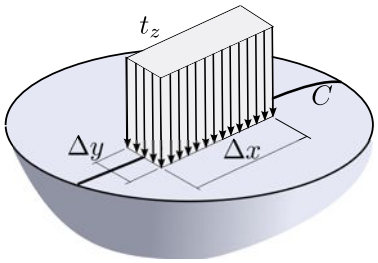


Fig. 13.13. Distributed load along a curve as the limit of traction on infinitesimally small area.

To summarize, the following should be remembered for concentrated force boundary conditions:

> Do not use the concentrated force or force along a curve boundary condition *unless* it is essential. Remember the caveats.

Furthermore, this should be remembered for concentrated displacement boundary conditions (support at a point, or support along a curve):

> Do not use the concentrated support boundary condition *unless* the associated reaction is guaranteed to be zero.

13.6.5 Symmetry and anti-symmetry

Considerable benefits may be often derived when the solution is expected to possess either symmetry, or anti-symmetry.

For the solution to display ***symmetry with respect to reflection*** in a symmetry plane, all the ingredients that go into the definition of the problem must display the same kind of symmetry: the geometry, the material, the boundary conditions, the initial conditions.

Let us first look at the conditions that must hold for the *displacements* on the plane of symmetry: Figure 13.14. By inspection, we see that the arrow representing displacement at point P is reflected into an arrow at point P' which is best described in components which are (i) in the plane of symmetry: these are the same for both arrows; and (ii) perpendicular to the plane of symmetry: these have opposite signs. Therefore, if P is made to approach the plane of symmetry, its mirror image merges with it when they both reach the plane of symmetry (point M), and since the two displacements then must be the same, we may conclude that the perpendicular component of displacement $u_\perp$ at a point on the plane of symmetry must be zero

$$u_\perp = 0 \,. \tag{13.47}$$

Now for the *tractions* on the plane of symmetry. By the symmetry conditions, the shear part of the

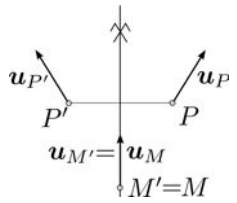


Fig. 13.14. Symmetric displacement pattern.

traction at point M on the surface with normal $\boldsymbol{n}$ must be equal to the shear part of the traction on the surface with the opposite normal $-\boldsymbol{n}$, i.e.

$$\boldsymbol{t}_{s(-n)} = \boldsymbol{t}_{s(n)} \,.$$

Furthermore, using equation (13.7) we have

$$\boldsymbol{t}_{s(n)} = \boldsymbol{t}_{(n)} - \big(\boldsymbol{n} \cdot \boldsymbol{t}_{(n)}\big)\, \boldsymbol{n} \,,$$

and substituting from (13.22) with $\boldsymbol{\sigma}$ being the stress at the point M (which is the same irrespectively of the normal)

$$\boldsymbol{t}_{s(n)} = \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma} - (\boldsymbol{n} \cdot \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})\, \boldsymbol{n} \,,$$

$$\boldsymbol{t}_{s(-n)} = \mathcal{P}_{-\boldsymbol{n}}\boldsymbol{\sigma} - (-\boldsymbol{n} \cdot \mathcal{P}_{-\boldsymbol{n}}\boldsymbol{\sigma})\,(-\boldsymbol{n}) = -\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma} + (\boldsymbol{n} \cdot \mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})\, \boldsymbol{n} = -\boldsymbol{t}_{s(n)} \,.$$

In order for both requirements to be satisfied,

$$\boldsymbol{t}_{s(n)} = \boldsymbol{0} \,, \tag{13.48}$$

must hold.

The state of ***anti-symmetry with respect to reflection in a plane*** is defined by conditions that specify it as the opposite of symmetry. The situation is illustrated in Fig. 13.15. The arrow representing *displacement* at point P is transformed into an arrow at point P' which in terms of components gives (i) opposite sign parallel with the plane of anti-symmetry; and (ii) same sign in the direction perpendicular to the plane of anti-symmetry. Therefore, when P is made to approach the plane of symmetry and its mirror image merges with it at point M, we conclude that the two components of displacement $u_{\parallel,i}$ at a point on the plane of symmetry must be zero

$$u_{\parallel,i} = 0 \quad \text{for the two in-plane directions } i \,. \tag{13.49}$$

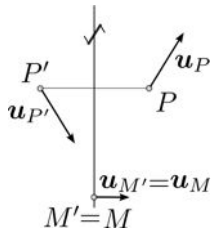


Fig. 13.15. Anti-symmetric displacement pattern.

For the tractions, an analysis quite similar to that leading to Eq. (13.48), but applied to the normal component of the traction, leads to the condition

$$t_\perp = 0 \; . \tag{13.50}$$

Therefore, we can summarize the boundary conditions on the plane of symmetry or on the plane of anti-symmetry with the delightfully simple Table 13.1. The boundary conditions that need to

Table 13.1. Boundary conditions on the plane of symmetry or anti-symmetry

Quantity	Symmetry	Anti-symmetry
Tractions $\parallel$	0	unknown
Displacements $\parallel$	unknown	$\boxed{0}$
Traction $\perp$	unknown	0
Displacement $\perp$	$\boxed{0}$	unknown

be explicitly prescribed in finite element analyses are boxed in the Table 13.1: zero tractions are incorporated automatically (natural boundary conditions!) and need not be explicitly specified. Note that the unknown tractions are the reactions.

13.6.6 Example: a pure-traction problem

Sometimes we encounter stress analysis problems where only a statically equilibrated set of traction and/or body loads is given. As an example, we consider the dog bone tensile specimen of Fig. 13.16 (slice through the axis of symmetry is shown). Uniform tractions are applied at the opposite cross-sections, equal in magnitude, but of opposite sign, so that the specimen is in static equilibrium: the so-called ***pure-traction problem***. As such, this set of boundary conditions does not allow for the finite element solution to be computed without additional devices: the entire specimen may be translated or rotated as a rigid body without any change in the stress state. Therefore, the stiffness matrix of the structure as a whole is singular.

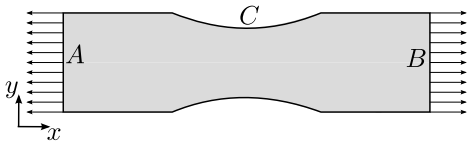


Fig. 13.16. Example of boundary conditions: dog bone specimen under tension.

The rigid body motion may be described by displacements of the form

$$\begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} 0 & -\Theta_z & \Theta_y \\ \Theta_z & 0 & -\Theta_x \\ -\Theta_y & \Theta_x & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \; , \tag{13.51}$$

where $\Theta_x, \Theta_y, \Theta_z$ and a_x, a_y, a_z are constants describing ***rotation*** (through the skew-symmetric matrix – another way of writing a cross product of two vectors), and ***translation*** of the points of a rigid body.

It doesn't take too much effort to verify that strains computed from displacements (13.51) are all identically zero. Therefore, also the deformation energy induced by the rigid body motion is zero, as is easily verified by referring to (13.44). As shown in Section 14.10, the global stiffness matrix of a structure that can move as a rigid body is by necessity singular. To restore the full rank of the stiffness matrix, all possible rigid body modes must be prevented by additional supports (displacement boundary conditions). Taking advantage of any symmetry conditions is a big help. Even though we may not be willing to actually solve the problem on a quarter of the geometry (producing a separate CAD model for each analysis is often not convenient), just inserting features such as split lines to which symmetry conditions may be applied will do the trick. For instance, Fig. 13.17 illustrates these two possibilities: on the left, one-quarter model is extracted from the full geometry, with zero normal displacement on the cut planes; on the right, split lines have been inserted, to which zero displacement perpendicular to the corresponding symmetry plane will be applied. Note that this complies with the modeling rule of Section 13.6.4: since the tractions applied at the end cross-sections are self-balancing, we can prove that all reactions along the symmetry planes must be zero. Therefore, it is okay to apply an "inadmissible" support along a curve.

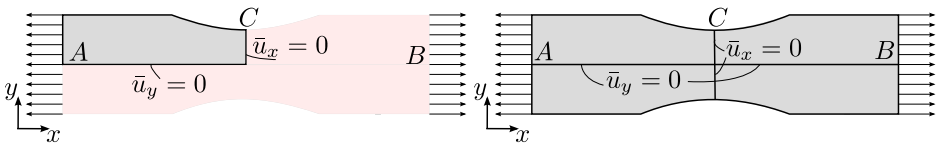


Fig. 13.17. Dog bone specimen under tension with two ways of using symmetry.

Another possibility is to support the specimen by applying six point supports. These would be selected to (i) prevent any rigid body motion, while (ii) ensuring all reactions at these point supports were identically zero. This would be achieved by formulating six equilibrium conditions for the specimen as a rigid body, and checking that the reactions at the supports would make equilibrium possible and unique (even though they *all must be zero*). To ensure that the reactions are statically determinate, the distances between the point supports must be able to change freely: while we want to support the specimen as a rigid body, it is not rigid, it needs to freely deform. An example of possible system of supports is shown in Fig. 13.18.

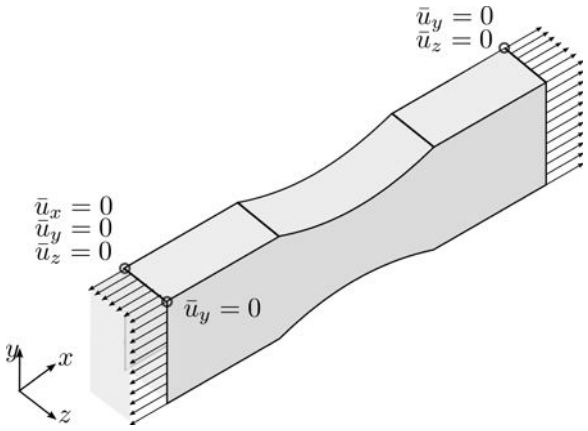


Fig. 13.18. Dog bone specimen supported with point constraints as a rigid body.

13.6.7 Example: shaft under torsion

Shear traction components are often generated by frictional contact between interacting bodies. However, contact problems are well outside the scope of this textbook, they are nonlinear and involve inequality constraints. The other situation in which we might wish to apply shear tractions is when we formulate simplified models in which the effect of the omitted part of a structure is introduced as a resultant (force or torque) into the model.

As an example, we will consider a shaft of circular cross-section, with two through-holes. The focus of our interest is the local stress concentration around the holes when the shaft is subjected to a known torque. As we do not wish to model the actual transmission of the torque into the shaft, the geometry of the shaft is reduced to just the small neighborhood of the holes, and the torque generated at the end points is applied as prescribed shear tractions in the end cross-sections of the short stump.

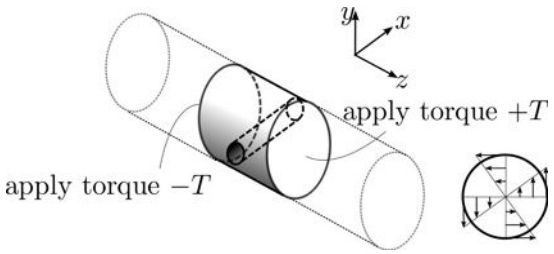


Fig. 13.19. Example of boundary conditions: shaft loaded by torque.

The way in which we distribute the shear tractions is only an approximation of the stress distribution that would exist in the complete part. Based on experimental observations accompanied by analyses of a few particular cases, the so-called ***Saint-Venant's principle*** [T83, B99], may be invoked: If a set of self-equilibrated tractions is applied on a limited subset of the boundary, its effect will be negligible beyond a certain range. By necessity, this principle is somewhat vague, and its applicability needs to be assessed case-by-case. Figure 13.20 illustrates the meaning: consider a beam of solid section, to which a traction $\overline{t}$ of a nonzero resultant is applied. If the traction is perturbed by a self-equilibrated load $\widehat{t}$, the stresses will change significantly only in a region extending in all directions approximately by the characteristic dimension of the beam d.

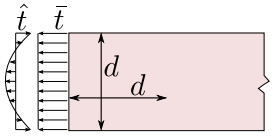


Fig. 13.20. Illustration of Saint-Venant's principle

Coming back to the shaft: relying on the Saint-Venant's principle, we apply the resultant torque by any convenient distribution of shear tractions. Again, this is a pure-traction problem, so essential boundary conditions to prevent rigid body motion should be added to make the solution unique. In this case, for instance a plane of anti-symmetry exists, or point supports could be added at convenient locations (for instance on the axis of the shaft).

13.6.8 Example: overspecified boundary conditions

At each point of the boundary, for each component either displacement or traction must be known. In some special circumstances, it might be of interest to prescribe *both* tractions and displacements on one and the same surface, for one or more components, and to compute the values of the boundary

conditions elsewhere. Consider for instance the thin plate of Fig. 13.21. The deflection could be measured experimentally at the top-facing surface, and the observation that it is traction free could be made. The question then is, could an elasticity problem be solved to determine the unknown tractions on the other five surfaces? The answer is yes, however the existence of a solution (any solution!) is not guaranteed at all, and the solution need not be unique. In particular, the assumptions on the top surface must be such that the displacements and the tractions along the surface are consistent. In this book we shall not attempt to solve problems of this nature, as special formulations are required and they are quite challenging.

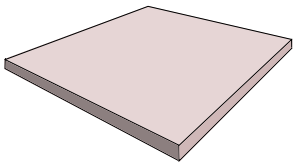


Fig. 13.21. Example of boundary conditions: plate with unknown boundary conditions everywhere except on the top face were both displacement and traction components are known.

13.7 Initial conditions

The ***initial conditions*** are essentially those discussed for the taut wire model: prescribe displacement and velocity at each point of the domain.

$$\boldsymbol{u}(\boldsymbol{x},0) = \bar{\boldsymbol{W}}(\boldsymbol{x}), \quad \frac{\partial \boldsymbol{u}}{\partial t}(\boldsymbol{x},0) = \bar{\boldsymbol{V}}(\boldsymbol{x}) \ , \tag{13.52}$$

where $\bar{\boldsymbol{W}}(\boldsymbol{x})$ (the initial deflection) and $\bar{\boldsymbol{V}}(\boldsymbol{x})$ (the initial velocity) are known functions. These functions need to be compatible with the boundary conditions as time $t = 0$: at all points $\boldsymbol{x}$ where we prescribe displacements on the boundary, the functions $\bar{\boldsymbol{W}}(\boldsymbol{x})$ and $\bar{\boldsymbol{V}}(\boldsymbol{x})$ must have components of the same value that is being prescribed.

Finally, it needs to be realized that the initial displacement determines also a distribution of the ***initial stress***. If the initial displacement is zero everywhere, the strains and hence also the stresses are zero everywhere. On the contrary, a nonzero initial displacement implies initial strains (likely to be also nonzero). From these strains, the initial stresses follow by the constitutive equation. Nonzero initial stresses need to be then applied as an initial condition.

14

Galerkin Formulation for Elastodynamics

Rearranging the balance equation (13.32) into the residual form leads to

$$\boldsymbol{r}_B = \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \overline{\boldsymbol{b}} - \mathcal{B}^T \boldsymbol{\sigma} \ , \tag{14.1}$$

which is a statement of imbalance when the force residual is nonzero. This is in complete analogy to the model of the taut wire, or the model of heat conduction. To arrive at a usable form of the method of weighted residuals for elastodynamics, the plan of action is the same as for the other models.

14.1 Manipulation of the residuals

1. Formulate the weighted residual equations for the balance equation, the force boundary condition, and the displacement boundary condition.
2. Satisfy the displacement condition by design of the trial functions: that will subject the trial functions to a condition along parts of the boundary.
3. Shift the derivatives from the stress to the test function. This will incorporate the natural boundary conditions in the balance residual equation (and eliminate the force boundary condition residual from further consideration); it will also place a condition on the form of the test functions.

14.1.1 The first two steps

We begin with step 1: The natural boundary condition (13.45) leads to the residual

$$r_{t,i} = \left(\mathcal{P}_n \boldsymbol{\sigma}\right)_i - \overline{t}_i \quad \text{on } S_{t,i} \quad \text{for } i = x, y, z \ , \tag{14.2}$$

which will be incorporated into the balance residual equation, and the displacement boundary condition (13.46) gives the residual

$$r_{u,i} = u_i - \overline{u}_i \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \ , \tag{14.3}$$

that will be made zero by the choice of the trial functions (which takes care of the step 2).

The weighted residual equations are integrals of the residuals over the corresponding surface. Since the displacement boundary condition residual is identically zero, it may be ignored. The traction boundary condition residual equation reads

$$\int_{S_{t,i}} r_{t,i} \eta_i \, \mathrm{d}S \ ,$$

or, expanded,

$$\int_{S_{t,i}} r_{t,i}\eta_i \,\mathrm{d}S = \int_{S_{t,i}} \left[(\mathcal{P}_n\boldsymbol{\sigma})_i - \overline{t}_i\right]\eta_i \,\mathrm{d}S = 0\,. \tag{14.4}$$

The balance weighted residual reads

$$\int_V \boldsymbol{r}_B\cdot\boldsymbol{\eta}\,\mathrm{d}V = \int_V \boldsymbol{\eta}\cdot\boldsymbol{r}_B\,\mathrm{d}V = \int_V \boldsymbol{\eta}\cdot\left(\rho\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} - \overline{\boldsymbol{b}} - \mathcal{B}^T\boldsymbol{\sigma}\right)\,\mathrm{d}V\,, \tag{14.5}$$

where $\boldsymbol{\eta}$ is a vector test function (with three components). At this point, we only require that the test function be sufficiently smooth for the integral to exist. The dot product of the residual and the vector test function is written in the "dot" form; when the weighted residual equation is written in terms of the components, transposes must be used as

$$\int_V \boldsymbol{\eta}\cdot\boldsymbol{r}_B\,\mathrm{d}V = \int_V [\boldsymbol{r}_B]^T[\boldsymbol{\eta}]\,\mathrm{d}V = \int_V [\boldsymbol{\eta}]^T[\boldsymbol{r}_B]\,\mathrm{d}V\,.$$

14.1.2 Step 3: Preliminaries

While the first two terms on the right-hand side of (14.1) present no difficulties, the stress term needs to be treated similarly to the previous two models to move one derivative from the stress to the test function. Therefore, in the next few paragraphs we focus on the integral

$$\int_V \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\,\mathrm{d}V\,.$$

It will be sought as one constituent of the chain-rule result (analogously to Eq. (8.4) for the heat conduction). The inner product of $\boldsymbol{\eta}$ and $\boldsymbol{\sigma}$ may be expressed using the vector-stress vector dot product operator (13.22) in the form $\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}$ (which is a vector). Therefore, we need an identity for the chain rule applied to the divergence $\operatorname{div}\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)$:

$$\operatorname{div}\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right) = (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma} + \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\,. \tag{14.6}$$

It may not be immediately clear *why* the right-hand side has this form, but to verify this formula is straightforward, albeit tedious. Expressing the rightmost term of (14.6) by the other two, we obtain

$$\int_V \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\,\mathrm{d}V = \int_V \operatorname{div}\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)\,\mathrm{d}V - \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\,\mathrm{d}V\,.$$

The divergence theorem (7.10) may be applied to the first term on the right to yield

$$\int_V \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\,\mathrm{d}V = \int_S \left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)\cdot\boldsymbol{n}\,\mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\,\mathrm{d}V\,.$$

The traction boundary condition (13.45) references $\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}$. To extricate this form from $\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)\cdot\boldsymbol{n}$ we note that the result of this dot product is a scalar (a number). This indicates that the stress is involved in a double dot product: the stress tensor is dotted with one vector, which is subsequently dotted with the second vector. Indeed, it is easily verified by multiplying through that

$$\left(\mathcal{P}_{\boldsymbol{\eta}}\boldsymbol{\sigma}\right)\cdot\boldsymbol{n} = (\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})\cdot\boldsymbol{\eta}\,.$$

As a result we obtain

$$\int_V \boldsymbol{\eta}\cdot\mathcal{B}^T\boldsymbol{\sigma}\,\mathrm{d}V = \int_S \boldsymbol{\eta}\cdot(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})\,\mathrm{d}S - \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\,\mathrm{d}V\,.$$

It will be useful to summarize the balance weighted residual (14.5) now as

$$\begin{aligned}\int_V \boldsymbol{\eta}\cdot\boldsymbol{r}_B\,\mathrm{d}V = &\int_V \boldsymbol{\eta}\cdot\rho\frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t}\,\mathrm{d}V - \int_V \boldsymbol{\eta}\cdot\overline{\boldsymbol{b}}\,\mathrm{d}V\\ &- \int_S \boldsymbol{\eta}\cdot(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma})\,\mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta})\cdot\boldsymbol{\sigma}\,\mathrm{d}V\,.\end{aligned} \tag{14.7}$$

14.1.3 Step 3: Conclusion

Following closely Section (8.6), the surface will now be split, for each component, into the part where traction is known, and the part where displacement is being prescribed

$$\int_S \boldsymbol{\eta} \cdot (\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}) \ \mathrm{d}S = \sum_{i=x,y,z} \int_{S_{t,i}} \eta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S + \int_{S_{u,i}} \eta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S \ ,$$

where $\eta_i = (\boldsymbol{\eta})_i$ is the i^{th} component of the test function. On the $S_{t,i}$ subset the traction component i is known from (13.45), but on the $S_{u,i}$ subset of the bounding surface, the component i of the traction is not known, it represents the reaction. To be able to ignore the reactions, we will resort to the same trick as for the other PDE models, namely we will put in place the requirement

$$\eta_i = 0 \quad \text{on} \quad S_{u,i} \ .$$

Therefore, with the constraint on the trial function to satisfy the essential boundary conditions on $S_{u,i}$, and a constraint on the test function to vanish on $S_{u,i}$, we have the weighted balance residual

$$\int_V \boldsymbol{\eta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \eta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V \tag{14.8}$$

To the weighted balance residual (14.8) we now add the weighted residuals of the natural boundary condition (14.4) and set the result equal to zero.

$$\begin{aligned} \int_V \boldsymbol{\eta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} \eta_i \left(\mathcal{P}_{\boldsymbol{n}}\boldsymbol{\sigma}\right)_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V \\ + \sum_{i=x,y,z} \int_{S_{t,i}} \eta_i \left[\left(\mathcal{P}_n\boldsymbol{\sigma}\right)_i - \overline{t}_i \right] \ \mathrm{d}S = 0 \end{aligned} \tag{14.9}$$

We can see that the terms with $\eta_i \left(\mathcal{P}_n\boldsymbol{\sigma}\right)_i$ cancel and we obtain the final form of the ***weighted residual equation***

$$\begin{aligned} &\int_V \boldsymbol{\eta} \cdot \rho \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} \ \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \overline{t}_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{\sigma} \ \mathrm{d}V = 0 \\ &\qquad u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \ . \end{aligned} \tag{14.10}$$

So far we have been using the velocity and the stress vector for convenience and brevity, but to produce a displacement-based computational model these will have to be replaced by references to the displacement field. Using

$$\boldsymbol{v} = \frac{\mathrm{d}\boldsymbol{u}}{\mathrm{d}t} \quad \Rightarrow \quad \frac{\mathrm{d}\boldsymbol{v}}{\mathrm{d}t} = \frac{\mathrm{d}^2\boldsymbol{u}}{\mathrm{d}t^2} = \ddot{\boldsymbol{u}} \ ,$$

and the constitutive equation (13.42) and the displacement-strain relation (13.39), the form that goes into the discretization process reads

$$\begin{aligned} &\int_V \boldsymbol{\eta} \cdot \rho \ddot{\boldsymbol{u}} \ \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \ \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \overline{t}_i \ \mathrm{d}S + \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\mathcal{B}\boldsymbol{u} \ \mathrm{d}V = 0 \\ &\qquad u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z \ . \end{aligned}$$

14.2 Method of weighted residuals as the principle of virtual work

An alternative route to Eq. (14.10) is via the ***principle of virtual work***. The test function is interpreted as a ***virtual displacement***. The constraint on the test function is postulated a priori:

virtual displacement is ***kinematically admissible***. The various terms in (14.10) are interpreted as the virtual work of the inertial forces, applied body load, applied tractions, and internal forces.

This approach tends to seem somewhat arbitrary, since a number of concepts are postulated to be taken on faith. In this book we therefore avoid this viewpoint. Nevertheless, it may be useful to be aware of these possible interpretations of the various terms as virtual quantities (especially work).

14.3 Discretizing

The primary variable, the displacement, is a vector quantity

$$\boldsymbol{u} = u_x \boldsymbol{e}_x + u_y \boldsymbol{e}_y + u_z \boldsymbol{e}_z \ ,$$

where $u_x, ...$ are the components, and $\boldsymbol{e}_x, ...$ are the basis vectors. All computer manipulations are performed in terms of components; the basis vectors are necessary only when a transition needs to be made from one basis to another. For simplicity, in this work and in the toolbox `FAESOR`, the basis in which the *displacement field* is expressed is the *global Cartesian basis*.

14.3.1 The trial function

The trial displacement vector function will be expressed in terms of the components in the global Cartesian basis (the basis is implied) as (compare with Sections 2.9 and 8.9)

$$[\boldsymbol{u}(\boldsymbol{x},t)] = \begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \sum_{i=1}^{N} N_i(\boldsymbol{x})[\boldsymbol{u}_i(t)] = \sum_{i=1}^{N} N_i(\boldsymbol{x}) \begin{bmatrix} u_{ix}(t) \\ u_{iy}(t) \\ u_{iz}(t) \end{bmatrix} \ . \tag{14.11}$$

Here $N_i(\boldsymbol{x})$ is a finite element basis function (at this point we assume it is defined on a three-dimensional mesh), and $u_{ix}(t), ...$ are nodal degrees of freedom (displacements at nodes) as functions of time. As in Section 8.9, it will be useful to separate the free degrees of freedom from the prescribed displacements. However, this will be somewhat more complicated because now we have three components, each of which has different sets of the free degrees of freedom and the prescribed ones. As an illustration consider Fig. 14.1: for the x component, the free degrees of freedom are u_{2x}, the prescribed (at zero value) are u_{1x}, u_{3x}; for the y component, the free degrees of freedom are u_{3y}, the prescribed (at zero value) are u_{1y}, u_{2y}. One possibility is to split the sum and write (note that the

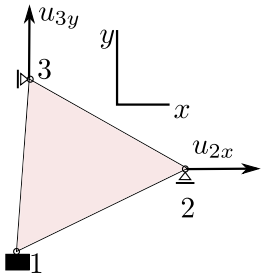


Fig. 14.1. A single element, with free degrees of freedom.

sets free i and prescribed i are in general different for each i)

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \begin{bmatrix} \sum\limits_{\text{free } i} N_i(\boldsymbol{x}) u_{ix}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x}) \overline{u}_{ix}(t) \\ \sum\limits_{\text{free } i} N_i(\boldsymbol{x}) u_{iy}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x}) \overline{u}_{iy}(t) \\ \sum\limits_{\text{free } i} N_i(\boldsymbol{x}) u_{iz}(t) + \sum\limits_{\text{prescribed } i} N_i(\boldsymbol{x}) \overline{u}_{iz}(t) \end{bmatrix} \ , \tag{14.12}$$

but a more convenient approach is the following trick: pretend the node has *both* free and prescribed degrees of freedom for each component at the same time, and zero out those that are inactive

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = \sum_{\text{all } i} N_i(\boldsymbol{x}) \left\{ \begin{bmatrix} u_{ix}(t) \\ u_{iy}(t) \\ u_{iz}(t) \end{bmatrix} + \begin{bmatrix} \overline{u}_{ix}(t) \\ \overline{u}_{iy}(t) \\ \overline{u}_{iz}(t) \end{bmatrix} \right\} , \tag{14.13}$$

where we define

$u_{ix}(t) = 0$ if the x degree of freedom at node i is prescribed;
$u_{iy}(t) = 0$ if the y degree of freedom at node i is prescribed;
$u_{iz}(t) = 0$ if the z degree of freedom at node i is prescribed,

and

$\overline{u}_{ix}(t) = 0$ if the x degree of freedom at node i is free;
$\overline{u}_{iy}(t) = 0$ if the y degree of freedom at node i is free;
$\overline{u}_{iz}(t) = 0$ if the z degree of freedom at node i is free.

Thus, for the example of Fig. 14.1 we have the following

$$\begin{bmatrix} u_{1x}(t) \\ u_{1y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{1x}(t) \\ \overline{u}_{1y}(t) \end{bmatrix} = \begin{bmatrix} \text{as given} \\ \text{as given} \end{bmatrix},$$

for node 1,

$$\begin{bmatrix} u_{2x}(t) \\ u_{2y}(t) \end{bmatrix} = \begin{bmatrix} u_{2x}(t) \\ 0 \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{2x}(t) \\ \overline{u}_{2y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ \text{as given} \end{bmatrix},$$

for node 2, and finally,

$$\begin{bmatrix} u_{3x}(t) \\ u_{3y}(t) \end{bmatrix} = \begin{bmatrix} 0 \\ u_{3y}(t) \end{bmatrix}, \quad \begin{bmatrix} \overline{u}_{3x}(t) \\ \overline{u}_{3y}(t) \end{bmatrix} = \begin{bmatrix} \text{as given} \\ 0 \end{bmatrix},$$

for node 3. In the toolbox code, the free degrees of freedom may be distinguished from the prescribed ones using the attributes of the `field` class. The free degrees of freedom get nonzero equation numbers (attribute `eqnums`), and the magnitudes are stored in the attribute `values`; the prescribed degrees of freedom are marked with the attribute `is_prescribed` and the value to which these degrees of freedom are being set is the `prescribed_value`. The `gather()` method of the `field` class may be used to retrieve all these attributes. A convenient way of looking at the attributes of the `field` object, or at any other `FAESOR` object for that matter, is the `OBgui` object browser.

14.3.2 The test function

Using the trick described below Eq. (14.13), we will write the test function as

$$\begin{bmatrix} \eta_x(\boldsymbol{x}) \\ \eta_y(\boldsymbol{x}) \\ \eta_z(\boldsymbol{x}) \end{bmatrix} = \sum_{\text{all } i} N_i(\boldsymbol{x}) \begin{bmatrix} \eta_{ix} \\ \eta_{iy} \\ \eta_{iz} \end{bmatrix} , \tag{14.14}$$

where we define

$\eta_{ix} = 0$ if the x degree of freedom at node i is prescribed;
$\eta_{iy} = 0$ if the y degree of freedom at node i is prescribed;
$\eta_{iz} = 0$ if the z degree of freedom at node i is prescribed;

otherwise $\eta_{ix}, \eta_{iy}, \eta_{iz}$ are arbitrary numbers.

We will also use a more succinct version

$$\begin{bmatrix} \eta_x(\boldsymbol{x}) \\ \eta_y(\boldsymbol{x}) \\ \eta_z(\boldsymbol{x}) \end{bmatrix} = [\eta(\boldsymbol{x})] = \sum_{\text{all } i} N_i(\boldsymbol{x}) \left[\eta_i \right] , \tag{14.15}$$

where $[\eta_i]$ stands for a column matrix holding the components $[\eta_i]_k$ of the vector of the degrees of freedom at node i.

14.3.3 Producing the requisite equations

It may be easily verified that the definition of the test function gives us just enough equations to solve for all the free degrees of freedom. Use (14.14) in Eq. (14.11) to obtain

$$\begin{aligned}\int_V [\boldsymbol{\eta}]^T \rho[\ddot{\boldsymbol{u}}]\,\mathrm{d}V - \int_V [\boldsymbol{\eta}]^T[\overline{\boldsymbol{b}}]\,\mathrm{d}V - \sum_{i=x,y,z}\int_{S_{t,i}} (\boldsymbol{\eta})_i\,\overline{t}_i\,\mathrm{d}S \\ + \int_V (\mathcal{B}[\boldsymbol{\eta}])^T\,\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\,\mathrm{d}V = 0 \qquad (14.16)\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\ .\end{aligned}$$

The key is to obtain $[\boldsymbol{\eta}]^T$ in all the terms. The trickiest tweaking will be required for $\mathcal{B}[\boldsymbol{\eta}]$. Substituting (14.14), we get

$$\mathcal{B}[\boldsymbol{\eta}] = \mathcal{B}\sum_{\text{all } j} N_j(\boldsymbol{x}) \begin{bmatrix}\eta_{jx}\\ \eta_{jy}\\ \eta_{jz}\end{bmatrix},$$

but since the $\eta_{jx}, \eta_{jy}, \eta_{jz}$'s are just numbers, the symmetric gradient operator works with the basis functions as if we simply multiplied the matrix of the operator with a scalar (the basis function N_j)

$$\mathcal{B}[\boldsymbol{\eta}] = \sum_{\text{all } j} \mathcal{B}\left(N_j(\boldsymbol{x})\right) \begin{bmatrix}\eta_{jx}\\ \eta_{jy}\\ \eta_{jz}\end{bmatrix}\ .$$

Spelled out in full:

$$\mathcal{B}\left(N_j(\boldsymbol{x})\right) = \begin{bmatrix} \dfrac{\partial N_j(\boldsymbol{x})}{\partial x} & 0 & 0 \\ 0 & \dfrac{\partial N_j(\boldsymbol{x})}{\partial y} & 0 \\ 0 & 0 & \dfrac{\partial N_j(\boldsymbol{x})}{\partial z} \\ \dfrac{\partial N_j(\boldsymbol{x})}{\partial y} & \dfrac{\partial N_j(\boldsymbol{x})}{\partial x} & 0 \\ \dfrac{\partial N_j(\boldsymbol{x})}{\partial z} & 0 & \dfrac{\partial N_j(\boldsymbol{x})}{\partial x} \\ 0 & \dfrac{\partial N_j(\boldsymbol{x})}{\partial z} & \dfrac{\partial N_j(\boldsymbol{x})}{\partial y} \end{bmatrix}.$$

Next, the test function is substituted as

$$\begin{aligned}\sum_{\text{all } j}[\eta_j]^T \int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]\,\mathrm{d}V - \sum_{\text{all } j}[\eta_j]^T \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]\,\mathrm{d}V \\ - \sum_{i=x,y,z}\sum_{\text{all } j}\left([\eta_j]\right)_i \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i\,\mathrm{d}S \\ - + \sum_{\text{all } j}[\eta_j]^T \int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\,\mathrm{d}V = 0 \qquad (14.17)\\ u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x,y,z\ .\end{aligned}$$

The components are independent, hence (14.17) should hold for each component separately

$$\sum_{\text{all } j}[\eta_j]_i \int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \,\mathrm{d}V - \sum_{\text{all } j}[\eta_j]_i \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \,\mathrm{d}V$$
$$- \sum_{\text{all } j}[\eta_j]_i \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i \,\mathrm{d}S$$
$$+ \sum_{\text{all } j}[\eta_j]_i \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V = 0 \tag{14.18}$$
$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

Regrouping the sums yields finally

$$\sum_{\text{all } j}[\eta_j]_i \left\{ \int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \,\mathrm{d}V - \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \,\mathrm{d}V \right.$$
$$\left. - \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i \,\mathrm{d}S + \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V \right\} = 0 \tag{14.19}$$
$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

Now we account for some of the $[\eta_j]_i$ components being zero, which happens whenever node j is on the boundary $S_{u,i}$. These components do not need any equations, and the contents of the braces may be ignored since the $[\eta_j]_i = 0$'s make them irrelevant.

On the other hand, the components $[\eta_j]_i$ that are not zero are completely arbitrary. Therefore, the contents of the braces have to vanish identically, yielding *one equation for each free degree of freedom.*

$$\int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \,\mathrm{d}V - \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \,\mathrm{d}V$$
$$- \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i \,\mathrm{d}S + \int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V = 0\,, \tag{14.20}$$
$$\text{where} \quad u_i = \overline{u}_i; \quad \text{for all} \quad j, \quad \text{and} \quad i = x, y, z, \quad \text{such that} \quad [\eta_j]_i \neq 0\,.$$

These equations express the ***dynamic force equilibrium*** at node j in the direction i.

14.4 The discrete equations: system of ODE's

Finally, we substitute the trial function from (14.13). We will use the more streamlined version

$$\begin{bmatrix} u_x(\boldsymbol{x},t) \\ u_y(\boldsymbol{x},t) \\ u_z(\boldsymbol{x},t) \end{bmatrix} = [u(\boldsymbol{x},t)] = \sum_{\text{all } k} N_k(\boldsymbol{x}) \left\{ \left[\, u_k(t) \,\right] + \left[\, \overline{u}_k(t) \,\right] \right\}\,, \tag{14.21}$$

where $[u_k]$ stands for a column matrix holding the components $[u_k]_m$ of the vector of the free degrees of freedom at node k; analogously for the prescribed degrees of freedom. To satisfy the essential boundary conditions by interpolation, we set for the prescribed mth component at node k

$$[\overline{u}_k(t)]_m = [\overline{u}(\boldsymbol{x}_k, t)]_m$$

where $\boldsymbol{x}_k$ is the location of node k.

The trial function is substituted into (14.20). To keep things orderly and clear, we will substitute term by term.

14.4.1 Inertial term: Mass matrix

We begin with the inertial effects:

$$\int_V N_j(\boldsymbol{x})\rho[\ddot{\boldsymbol{u}}]_i \, \mathrm{d}V =$$
$$\int_V N_j(\boldsymbol{x})\rho \sum_{\text{all } k} N_k(\boldsymbol{x}) \left\{ \left[\ddot{u}_k(t) \right]_i + \left[\overline{\ddot{u}}_k(t) \right]_i \right\} \, \mathrm{d}V =$$
$$\sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \left\{ \left[\ddot{u}_k(t) \right]_i + \left[\overline{\ddot{u}}_k(t) \right]_i \right\} =$$
$$\sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \, \delta_{im} \left[\ddot{u}_k(t) \right]_m$$
$$+ \sum_{\text{all } k} \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \, \delta_{im} \left[\overline{\ddot{u}}_k(t) \right]_m \ . \tag{14.22}$$

Two terms emerge: firstly, the ***inertial force***

$$F_{a,(j,i)} = \sum_{\text{all } k} M_{(j,i)(k,m)} \left[\ddot{u}_k(t) \right]_m \ , \tag{14.23}$$

produced by the free accelerations which are coupled together by the ***consistent mass matrix***

$$M_{(j,i)(k,m)} = \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, \mathrm{d}V \, \delta_{im} \ , \tag{14.24}$$

where (j,i) means *equation number* corresponding to component i at node j (which is a free degree of freedom). Note well that $M_{(j,i)(k,m)}$ is a *two-dimensional array*, addressed by the row index (j,i) and the column index (k,m).

Secondly, there is the ***inertial load*** produced by the acceleration of the supported nodes

$$F_{\overline{a},(j,i)} = \sum_{\text{all } k} \overline{M}_{(j,i)(k,m)} \left[\overline{\ddot{u}}_k(t) \right]_m \ , \tag{14.25}$$

where the matrix elements $\overline{M}_{(j,i)(k,m)}$ are calculated exactly as those of (14.24), but (k,m) corresponds to component m at node k, which is prescribed.

Exercise 72.

Compute the mass matrix of the tetrahedron T4 element. Assume uniform mass density.

Node	x	y	z
1	1.4727	0.9193	-0.7713
2	1.4727	1.2000	0
3	1.4727	0.4543	0.0530
4	3.0000	0.5063	0.0290

Solution: Numerical quadrature will be used, both one-point and four-point rules (c.f. Table 11.2). The volume integral

$$M_{(j,i)(k,m)} = \int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, dV \, \delta_{im} \ ,$$

will be approximated as

$$\int_V N_j(\boldsymbol{x})\rho N_k(\boldsymbol{x}) \, dV \approx \sum_q N_j(\boldsymbol{\xi}_q)\rho N_k(\boldsymbol{\xi}_q) W_q \det \left[J(\boldsymbol{\xi}_q) \right]$$

The basis functions for the tetrahedron are given in (11.7). Therefore, the (constant) gradients of the basis functions with respect to the parametric coordinates are obtained (analogously to the triangle T3; viz (8.56)) as

```
Nder=[-1,-1,-1;
    1,0,0;
    0,1,0;
    0,0,1];
```

and the Jacobian matrix follows from (8.54)

```
x= [1.4727    0.9193   -0.7713
 1.4727    1.2000         0
 1.4727    0.4543    0.0530
  3.0000    0.5063    0.0290];
J=x'*Nder
J =
         0         0    1.5273
    0.2807   -0.4650   -0.4130
    0.7713    0.8243    0.8003
```

The Jacobian is constant (i.e. independent of the integration point)

```
>> detJ=det(J)
detJ =
    0.9012
```

The one-point rule evaluates the basis functions of the centroid of the tetrahedron. Therefore all the basis functions assume the value of 1/4 at the quadrature point. For instance we get

$$\int_V N_1(\boldsymbol{x})\rho N_1(\boldsymbol{x})\, dV \approx N_1(\boldsymbol{\xi}_k)\rho N_1(\boldsymbol{\xi}_k) W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = (1/4)\rho(1/4)(1/6)0.9012 = 0.0094\rho$$

But a more descriptive expression will be obtained by keeping $W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = \det\left[J(\boldsymbol{\xi}_1)\right]/6 = V_{tet}$ instead of numbers, where V_{tet} is the volume of the tetrahedron element (call for Exercise 56), so that the above may be put as

$$\int_V N_1(\boldsymbol{x})\rho N_1(\boldsymbol{x})\, dV \approx \frac{1}{16} V_{tet}\rho = \frac{1}{16} m_{tet}$$

where m_{tet} is the mass of the tetrahedron. Since all basis functions have the same value of the quadrature point, the same expression is obtained for the one-point quadrature rule for all basis functions. The δ_{im} represents a 3×3 identity matrix, and the mass matrix therefore results as

$$M = \frac{1}{16} m_{tet} \begin{bmatrix} \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} \\ \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} \\ \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} \\ \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} & \begin{bmatrix} 1&0&0\\0&1&0\\0&0&1 \end{bmatrix} \end{bmatrix}$$

Note that there are four "1"'s in each row, which means that for the four nodes accelerating in either of the three directions with the same acceleration a (i.e. the element accelerates as a rigid body) the total inertial force is $a\, m_{tet}$, as expected.

Now we will apply the four-point quadrature rule from Table 11.2. For instance we have

$$\int_V N_2(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \sum_q N_2(\boldsymbol{\xi}_q)\rho N_3(\boldsymbol{\xi}_q) W_q \det\left[J(\boldsymbol{\xi}_q)\right] \approx \rho W_q \det\left[J(\boldsymbol{\xi}_q)\right] \sum_q N_2(\boldsymbol{\xi}_q) N_3(\boldsymbol{\xi}_q)$$

because $\rho W_q \det\left[J(\boldsymbol{\xi}_q)\right]$ is constant. Substituting the definitions of the basis functions, $N_2 = \xi$ and $N_3 = \eta$, yields

$$\sum_q N_2(\boldsymbol{\xi}_q) N_3(\boldsymbol{\xi}_q) = aa + aa + ba + ab = 2a(a+b) = 1/5$$

and hence

$$\int_V N_2(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \frac{1}{20} V_{tet}\rho = \frac{1}{20} m_{tet}$$

Therefore, the mass matrix will link the nodes 2 and 3 by the 3×3 matrix

$$\frac{1}{20} m_{tet} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Similarly

$$\int_V N_3(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \sum_q N_3(\boldsymbol{\xi}_q)\rho N_3(\boldsymbol{\xi}_q) W_q \det\left[J(\boldsymbol{\xi}_q)\right] \approx \rho W_q \det\left[J(\boldsymbol{\xi}_q)\right] \sum_q N_3(\boldsymbol{\xi}_q) N_3(\boldsymbol{\xi}_q)$$

and

$$\sum_q N_3(\boldsymbol{\xi}_q) N_3(\boldsymbol{\xi}_q) = aa + aa + aa + bb = 2/5$$

and hence

$$\int_V N_3(\boldsymbol{x})\rho N_3(\boldsymbol{x})\, dV \approx \frac{1}{10} V_{tet}\rho = \frac{1}{10} m_{tet}$$

$$M = \frac{1}{20} m_{tet} \begin{bmatrix} \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix} \end{bmatrix}$$

Again, to check the mass matrix we can consider the element accelerating as a rigid body in any one of the three directions in the Cartesian coordinates. Since for any selected direction, for each node the sum of the row of the mass matrix is $m_{tet}/4$, the total mass obtained by adding the contributions from all four nodes is m_{tet}, as expected.

Both of the above mass matrices are the so-called consistent mass matrices, but only the second is integrated exactly. The first matrix is integrated inaccurately. As we have seen for the prestressed wire model, various tricks can be undertaken with integration rules for mass matrices to improve the accuracy of the computed frequencies, so we can't necessarily conclude that integrating the mass matrix inaccurately will produce a less accurate estimate of the natural frequencies (and vice versa).

14.4.2 Body loads and traction loads

The next two terms represent external loads. The ***body load vector*** component (j,i) corresponding to free component i at node j is

$$F_{b,(j,i)} = \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \, \mathrm{d}V \ . \tag{14.26}$$

The ***surface traction load vector*** component (j,i) corresponding to free component i at node j is

$$F_{t,(j,i)} = \int_{S_{t,i}} N_j(\boldsymbol{x})\overline{t}_i \, \mathrm{d}S \ . \tag{14.27}$$

Note well that $F_{b,(j,i)}$ and $F_{t,(j,i)}$ are components of a *one-dimensional array* (column vector), addressed by the row index (j,i).

Exercise 73.

Compute the nodal loads of the tetrahedron T4 element due to a uniform body load $\overline{\boldsymbol{b}}$.

Solution: Numerical quadrature with a one-point rule will be adequate for this task since it can exactly integrate linear functions over the domain of the tetrahedron. The volume integral

$$F_{b,(j,i)} = \int_V N_j(\boldsymbol{x})[\overline{\boldsymbol{b}}]_i \, \mathrm{d}V$$

may be simplified for the uniform body load to read

$$F_{b,(j,i)} = [\overline{\boldsymbol{b}}]_i \int_V N_j(\boldsymbol{x}) \, \mathrm{d}V \ .$$

The integral over the volume of the element may be evaluated in the coordinates of the parametric domain of the standard shape

$$\int_V N_j(\boldsymbol{x}) \, \mathrm{d}V \ = \int_{V_{[\xi,\eta,\zeta]}} N_j(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right] \ \mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta$$

As shown in Exercise 56), the Jacobian of the T4 element is related to its volume as $\det\left[J(\boldsymbol{\xi})\right] = 6V_{tet}$. The one-point numerical quadrature therefore results in

$$\int_{V_{[\xi,\eta,\zeta]}} N_j(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right] \ \mathrm{d}\xi\mathrm{d}\eta\mathrm{d}\zeta = N_j(\boldsymbol{\xi}_1)W_1 \det\left[J(\boldsymbol{\xi}_1)\right] = (1/4)(1/6)6V_{tet} = \frac{V_{tet}}{4}$$

and therefore the uniform body load contributes the force

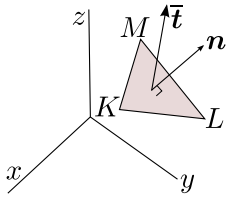
$$F_{b,(j,i)} = \frac{[\overline{\boldsymbol{b}}]_i V_{tet}}{4}$$

to each of the four nodes of the element. In other words, the total force $[\overline{\boldsymbol{b}}]_i V_{tet}$ acting on the entire element in the direction i is split equally among its four nodes.

Exercise 74.

Compute the nodal loads for the triangle T3 surface element generated by uniform traction load $\overline{\boldsymbol{t}}$.

Solution: The surface integral



$$F_{t,(j,i)} = \int_{S_{t,i}} N_j(\boldsymbol{x}) \overline{t}_i \, dS = \int_{S_{tri}} N_j(\boldsymbol{x}) \overline{t}_i \, dS \, ,$$

where S_{tri} is the triangle KLM, may be simplified for uniform traction load to read

$$F_{t,(j,i)} = \overline{t}_i \int_{S_{tri}} N_j(\boldsymbol{x}) \, dS$$

The task therefore reduces to the integration of basis functions over the surface of the triangle, which may be evaluated in the coordinates of the parametric domain of the standard triangle

$$\int_{S_{tri}} N_j(\boldsymbol{x}) \, \mathrm{d}S \; = \int_{S_{[\xi,\eta]}} N_j(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right] \; \mathrm{d}\xi \mathrm{d}\eta$$

The Jacobian for the triangle is obtained as the cross product of the two tangent vectors (refer to Figure 11.8)

$$\frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \xi}, \quad \text{and} \quad \frac{\partial \boldsymbol{x}(\xi,\eta)}{\partial \eta} \, ,$$

and because the basis functions are linear in ξ, η the tangent vectors are constant (the same at all points of the triangle) – refer to Exercise 33. It is also shown in Exercise 35 that the Jacobian of the surface is equal to twice the area of the triangle. Hence we get

$$\int_{S_{[\xi,\eta]}} N_j(\boldsymbol{\xi}) \det\left[J(\boldsymbol{\xi})\right] \; \mathrm{d}\xi \mathrm{d}\eta = \det\left[J(\boldsymbol{\xi})\right] \int_{S_{[\xi,\eta]}} N_j(\boldsymbol{\xi}) \; \mathrm{d}\xi \mathrm{d}\eta = 2 S_{tri} \int_{S_{[\xi,\eta]}} N_j(\boldsymbol{\xi}) \; \mathrm{d}\xi \mathrm{d}\eta$$

Numerical quadrature with a one-point rule (Table 8.1) will be adequate, as N_j is linear in its arguments

$$\int_{S_{[\xi,\eta]}} N_j(\boldsymbol{\xi}) \; \mathrm{d}\xi \mathrm{d}\eta \approx N_j(\boldsymbol{\xi}_1) W_1 = (1/3)(1/2) = 1/6$$

Thus we conclude that the uniform traction load contributes the force component i

$$F_{t,(j,i)} = \overline{t}_i \int_{S_{tri}} N_j(\boldsymbol{x}) \, dS = \overline{t}_i \times 2 S_{tri} \times (1/6) = \frac{\overline{t}_i S_{tri}}{3}$$

to each of the three nodes j of the element. In words, the total force $\overline{t}_i S_{tri}$ acting on the entire element in the direction i is split equally among its three nodes.

14.4.3 Resisting forces: Stiffness matrix

Finally, the trial function (14.21) is substituted into the last term of (14.20).

$$\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}[\boldsymbol{u}]\right]_i \,\mathrm{d}V =$$
$$\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\sum_{\text{all } k} N_k(\boldsymbol{x})\left\{\left[u_k(t)\right]+\left[\overline{u}_k(t)\right]\right\}\right]_i \,\mathrm{d}V =$$
$$\sum_{\text{all } k}\int_V \left[\mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\left\{\left[u_k(t)\right]+\left[\overline{u}_k(t)\right]\right\}\right]_i \,\mathrm{d}V =$$
$$\sum_{\text{all } k}\sum_{m}\left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,\mathrm{d}V\right]_{im}\left\{\left[u_k(t)\right]_m+\left[\overline{u}_k(t)\right]_m\right\} \tag{14.28}$$

Two contributions result: the first is the ***resisting force*** produced by the deformed material.

$$F_{r,(j,i)} = \sum_{\text{all } k} K_{(j,i)(k,m)}\left[u_k(t)\right]_m \ . \tag{14.29}$$

The matrix generating the resisting force is the ***stiffness matrix***

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,\mathrm{d}V\right]_{im} , \tag{14.30}$$

where (j,i) $[(k,m)]$ means equation number corresponding to component i at node j (component m at node k); both are free degrees of freedom.

The second is the ***nonzero-displacement load*** due to the deformation induced by prescribed essential boundary conditions. (Remark: In structural analysis, this kind of load is frequently associated with the loading condition called the *support settlement.*)

$$F_{\overline{r},(j,i)} = \sum_{\text{all } k} \overline{K}_{(j,i)(k,m)}\left[\overline{u}_k(t)\right]_m \ . \tag{14.31}$$

The elements $\overline{K}_{(j,i)(k,m)}$ are computed exactly as in (14.30), but (k,m) corresponds to component m at node k, which is prescribed.

14.4.4 Summary of the elastodynamics ODE's

Summing the forces (14.23), (14.25), (14.26), (14.27), (14.29), and (14.31) yields a system of second-order ordinary differential equations for the free displacements $\left[u_k(t)\right]_m$

$$\sum_{\text{all } k} M_{(j,i)(k,m)}\left[\ddot{u}_k(t)\right]_m + \sum_{\text{all } k} K_{(j,i)(k,m)}\left[u_k(t)\right]_m = -F_{\overline{a},(j,i)} - F_{\overline{r},(j,i)} + F_{b,(j,i)} + F_{t,(j,i)} \ . \tag{14.32}$$

In the convenient matrix notation, we could write

$$\boldsymbol{M}\ddot{\boldsymbol{U}} + \boldsymbol{K}\boldsymbol{U} = \boldsymbol{L} \ , \tag{14.33}$$

where $\boldsymbol{U}$ collects all the free degrees of freedom. These equations could be directly integrated using a Matlab integrator as indicated in Section 6.4, or even more suitably with a specialized mechanical integrator such as the Newmark average-acceleration integrator. Other approaches, such as integration of the harmonic modal equations are often used.

14.5 Constitutive equations of linearly elastic materials

The strain displacement operator (symmetric gradient operator) (13.39) links displacements in terms of their components in the global Cartesian basis to strains. The strains could be expressed in the

same Cartesian basis, but need not be. In fact, it will be most useful not to express the strains in the same global Cartesian coordinate system. The motivating factor is the constitutive equation: For some materials it will be important to keep track of the local orientation of the material volume. For instance, fiber reinforced materials will have very different stiffness properties along the fibers as opposed to perpendicularly to the fibers.

The components of the material stiffness matrix (or the material compliance matrix) are to be understood as being expressed in the local coordinate system $\boldsymbol{e}_{\overline{x}}$, $\boldsymbol{e}_{\overline{y}}$, $\boldsymbol{e}_{\overline{z}}$ attached to the material point in the form of (8.66) (except that the transformation matrix has three columns and rows).

14.5.1 General anisotropic material.

Because of the symmetry of the material stiffness, for the most general elastic material the number of elastic coefficients is only 21 out of the total of 36 elements of the material stiffness matrix: the ***general anisotropic material***. Still, to identify all of these constants represents a major experimental effort, and few engineering materials are characterized as fully anisotropic.

14.5.2 Orthotropic material.

If a material has three mutually orthogonal planes of symmetry, it is known as an ***orthotropic material***. For instance wood is often characterized as such type of material, and fiber-reinforced composites are in more sophisticated analyses also treated as orthotropic. The compliance matrix

$$\boldsymbol{C} = \boldsymbol{D}^{-1} ,$$

has a pleasingly simple appearance

$$\boldsymbol{C} = \begin{bmatrix} {E_1}^{-1}, & -\frac{\nu_{12}}{E_1}, & -\frac{\nu_{13}}{E_1}, & 0, & 0, & 0 \\ -\frac{\nu_{12}}{E_1}, & {E_2}^{-1}, & -\frac{\nu_{23}}{E_2}, & 0, & 0, & 0 \\ -\frac{\nu_{13}}{E_1}, & -\frac{\nu_{23}}{E_2}, & {E_3}^{-1}, & 0, & 0, & 0 \\ 0, & 0, & 0, & {G_{12}}^{-1}, & 0, & 0 \\ 0, & 0, & 0, & 0, & {G_{13}}^{-1}, & 0 \\ 0, & 0, & 0, & 0, & 0, & {G_{23}}^{-1} \end{bmatrix} .$$

All nine coefficients are independent, and need to be provided as input [H98]. The material stiffness matrix is a bit of a mess. The nonzero elements are

$$\begin{aligned} D_{11} &= \frac{C_{22}C_{33} - C_{23}C_{23}}{C}, & D_{12} &= \frac{C_{13}C_{23} - C_{12}C_{33}}{C}, \\ D_{13} &= \frac{C_{12}C_{23} - C_{13}C_{22}}{C}, & D_{22} &= \frac{C_{33}C_{11} - C_{13}C_{13}}{C}, \\ D_{23} &= \frac{C_{12}C_{13} - C_{23}C_{11}}{C}, & D_{33} &= \frac{C_{11}C_{22} - C_{12}C_{12}}{C}, \\ D_{44} &= G_{12}, & D_{55} &= G_{13}, & D_{66} &= G_{23}. \end{aligned}$$

Here $C = C_{11}C_{22}C_{33} - C_{11}C_{23}C_{23} - C_{22}C_{13}C_{13} - C_{33}C_{12}C_{12} + 2C_{12}C_{23}C_{13}$.

14.5.3 Transversely isotropic material.

If a material has an infinite number of planes of symmetry passing through an axis (in this case, the local x-axis), and one plane of symmetry perpendicular to this axis, it is known as a ***transversely isotropic material***. Unidirectionally reinforced composites are of this type, as are for instance muscles. The direction of the fibers is special (oriented along the local x-axis), but the material

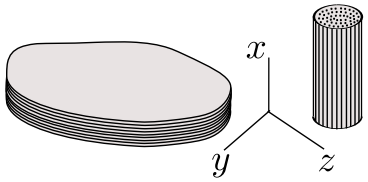


Fig. 14.2. Transversely isotropic model is appropriate for layered or fiber-reinforced materials

behaves isotropically in the planes perpendicular to the fibers. Layered materials are also modeled as transversely isotropic: here the direction perpendicular to the layers is special. Figure 14.2 offers an illustration of these two types.

The compliance matrix is obtained from the orthotropic compliance by setting $E_2 = E_3$, $\nu_{12} = \nu_{13}$, $G_{12} = G_{13}$, and importantly

$$G_{23} = \frac{E_2}{2(1+\nu_{23})} ,$$

requiring five independent constants, E_1, E_2, ν_{12}, G_{12}, and ν_{23}.

14.5.4 Isotropic material.

If a material has an infinite number of planes of symmetry of all possible orientations, it is known as an ***isotropic material***. The compliance matrix of isotropic material is based on two material properties, for instance the Young's modulus E and Poisson's ratio ν

$$\boldsymbol{C} = \begin{bmatrix} E^{-1}, & -\frac{\nu}{E}, & -\frac{\nu}{E}, & 0, & 0, & 0 \\ -\frac{\nu}{E}, & E^{-1}, & -\frac{\nu}{E}, & 0, & 0, & 0 \\ -\frac{\nu}{E}, & -\frac{\nu}{E}, & E^{-1}, & 0, & 0, & 0 \\ 0, & 0, & 0, & G^{-1}, & 0, & 0 \\ 0, & 0, & 0, & 0, & G^{-1}, & 0 \\ 0, & 0, & 0, & 0, & 0, & G^{-1} \end{bmatrix} .$$

The shear modulus G is not independent, but is expressed as

$$G = \frac{E}{2(1+\nu)} .$$

The material stiffness is then

$$\boldsymbol{D} = \begin{bmatrix} \lambda+2G & \lambda & \lambda & 0 & 0 & 0 \\ \lambda & \lambda+2G & \lambda & 0 & 0 & 0 \\ \lambda & \lambda & \lambda+2G & 0 & 0 & 0 \\ 0 & 0 & 0 & G & 0 & 0 \\ 0 & 0 & 0 & 0 & G & 0 \\ 0 & 0 & 0 & 0 & 0 & G \end{bmatrix} ,$$

where we introduce the Lamé constant

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}$$

for convenience.

14.6 Imposed (thermal) strains

Often the material from which a structure is built up reacts to the environment by deformation. The material experiences the environment in possibly different ways or different measures in different locations, and stresses are produced.

To get started, think about a very small piece of material that is exposed to the environment so that we can assume that the resultant relative deformation is homogeneous and no stress is produced. For the sake of this argument, let us consider one particular environmental effect: ***thermal expansion***. However, similar effects may be produced by shrinkage, swelling, piezoelectric effects, and so on.

When the small sample of material is at a *reference temperature* it is unstressed, and we define its displacements and the associated strains to be zero in this state: the *reference state*. Then the temperature is increased by ΔT, and the material responds by displacement, and because by assumption the deformation is homogeneous, the entire sample experiences uniform strains. Based on experimental evidence, the following model is adopted for orthotropic materials to describe the strains in coordinates aligned with the material directions

$$[\boldsymbol{\epsilon}_\Theta] = \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \Delta T \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \\ 0 \\ 0 \\ 0 \end{bmatrix} \ . \tag{14.34}$$

Evidently, the thermal expansion is assumed not to cause any shear strains. The factors $\alpha_x, \alpha_y, \alpha_z$ are the so-called ***coefficients of thermal expansion***; different in different directions, in general.

In addition to the imposed strain $\boldsymbol{\epsilon}_\Theta$, the sample is also exposed to stresses on its boundary, again such that they result in uniform strain. The total strains that the sample experiences consist of the imposed strains to which the mechanical strains are added. Since the mechanical strains are available from the stresses through the constitutive equation, we write

$$\boldsymbol{\epsilon} = \boldsymbol{\epsilon}_{\text{total}} = \boldsymbol{\epsilon}_\Theta + \boldsymbol{C}\boldsymbol{\sigma} \ . \tag{14.35}$$

Therefore, we have a modification of the constitutive equation

$$\boldsymbol{\sigma} = \boldsymbol{D}\left(\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\Theta\right) \ . \tag{14.36}$$

The total strain is related to the displacement via the strain-displacement relation (13.39), and thus we may write

$$\boldsymbol{\sigma} = \boldsymbol{D}\left(\mathcal{B}\boldsymbol{u} - \boldsymbol{\epsilon}_\Theta\right) \ .$$

This is sometimes also presented as

$$\boldsymbol{\sigma} = \boldsymbol{D}\mathcal{B}\boldsymbol{u} + \boldsymbol{\sigma}_\Theta \ ,$$

where we introduce the so-called ***thermal stress*** $\boldsymbol{\sigma}_\Theta$, but purely as a convenience; the primary quantity is the measurable thermal strain.

As expected, the residual equation form that enters the discretization process needs to be augmented with respect to (14.11) to become

$$\int_V \boldsymbol{\eta} \cdot \rho\ddot{\boldsymbol{u}} \, \mathrm{d}V - \int_V \boldsymbol{\eta} \cdot \overline{\boldsymbol{b}} \, \mathrm{d}V - \sum_{i=x,y,z} \int_{S_{t,i}} (\boldsymbol{\eta})_i \, \overline{t}_i \, \mathrm{d}S \tag{14.37}$$

$$+ \int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\left(\mathcal{B}\boldsymbol{u} - \boldsymbol{\epsilon}_\Theta\right) \, \mathrm{d}V = 0 \tag{14.38}$$

$$u_i = \overline{u}_i \quad \text{and} \quad (\boldsymbol{\eta})_i = 0 \quad \text{on } S_{u,i} \quad \text{for } i = x, y, z$$

Clearly, there will be one more term to discretize

$$-\int_V (\mathcal{B}\boldsymbol{\eta}) \cdot \boldsymbol{D}\boldsymbol{\epsilon}_\Theta \, \mathrm{d}V \, , \tag{14.39}$$

yielding the ***thermal strain load***

$$F_{\Theta,(j,i)} = \left[\int_V \mathcal{B}^T \left(N_j(\boldsymbol{x}) \right) \boldsymbol{D}\boldsymbol{\epsilon}_\Theta \, \mathrm{d}V \right]_i \, . \tag{14.40}$$

Therefore, with the inclusion of the thermal strains, the system of ordinary differential equations (14.32) that describe the discrete problem becomes

$$\begin{aligned} \sum_{\text{all } k} M_{(j,i)(k,m)} \left[\ddot{u}_k(t) \right]_m + \sum_{\text{all } k} K_{(j,i)(k,m)} \left[u_k(t) \right]_m = \\ - \sum_{\text{all } k} \overline{M}_{(j,i)(k,i)} \left[\overline{\ddot{u}}_k(t) \right]_i - \sum_{\text{all } k} \overline{K}_{(j,i)(k,m)} \left[\overline{u}_k(t) \right]_m \\ + F_{b,(j,i)} + F_{t,(j,i)} + F_{\Theta,(j,i)} \, . \end{aligned} \tag{14.41}$$

14.7 Strain-displacement matrix

The elements of the stiffness matrix (14.30) are evaluated using numerical quadrature element-by-element (explained in detail in Section 8.12 for the conductivity matrix). This operation produces element-level stiffness matrices that couple together the finite element nodes of each element. To simplify the notation, we will define the strain-displacement matrix as applied to a single basis function (the nodal strain-displacement matrix)

$$\boldsymbol{B}_k^e = \mathcal{B} \left(N_k(\boldsymbol{x}) \right) \quad \text{for node } \; k \text{ and } \; \boldsymbol{x} \in \text{element } e \, , \tag{14.42}$$

where k indicates the node number, and e identifies the element (it bears emphasis that the strain-displacement matrix of node k is different in each element that shares this node). Here $\mathcal{B}$ is the symmetric gradient operator

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 & 0 \\ 0 & \partial/\partial y & 0 \\ 0 & 0 & \partial/\partial z \\ \partial/\partial y & \partial/\partial x & 0 \\ \partial/\partial z & 0 & \partial/\partial x \\ 0 & \partial/\partial z & \partial/\partial y \end{bmatrix} \tag{14.43}$$

that has made an appearance many times already. The nodal strain-displacement matrix of (14.42) is written explicitly by applying the partial derivatives to the argument N_k

$$\boldsymbol{B}_k^e = \mathcal{B} \left(N_k(\boldsymbol{x}) \right) = \begin{bmatrix} \partial N_k(\boldsymbol{x})/\partial x & 0 & 0 \\ 0 & \partial N_k(\boldsymbol{x})/\partial y & 0 \\ 0 & 0 & \partial N_k(\boldsymbol{x})/\partial z \\ \partial N_k(\boldsymbol{x})/\partial y & \partial N_k(\boldsymbol{x})/\partial x & 0 \\ \partial N_k(\boldsymbol{x})/\partial z & 0 & \partial N_k(\boldsymbol{x})/\partial x \\ 0 & \partial N_k(\boldsymbol{x})/\partial z & \partial N_k(\boldsymbol{x})/\partial y \end{bmatrix} \, . \tag{14.44}$$

Using the nodal strain-displacement matrices we can compute the strain vector inside a finite element from the nodal displacements as (in components)

$$[\boldsymbol{\epsilon}^e] = \sum_k [\boldsymbol{B}_k^e][u_k] \tag{14.45}$$

Consider now a structure that consists of a single element, say a tetrahedron with four nodes, K, L, M and P (Fig. 14.3). Let us also assume that all the degrees of freedom are free (hypothetically:

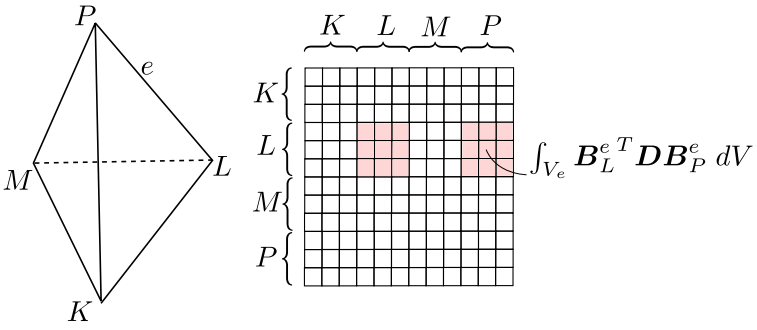


Fig. 14.3. Assembly of the element stiffness matrix

if they really were, the stiffness matrix would be singular). The stiffness matrix of such a structure (i. e. of this single element) has 12 rows and columns.

$$K_{(j,i)(k,m)} = \left[\int_{V_e} \boldsymbol{B}_j^{eT} \boldsymbol{D} \boldsymbol{B}_k^e \, \mathrm{d}V \right]_{im} \quad , \quad j,k = K,L,M,P \; . \tag{14.46}$$

For j,k fixed, say $j = L$ and $k = P$, the matrix

$$\int_{V_e} \boldsymbol{B}_L^{e\,T} \boldsymbol{D} \boldsymbol{B}_P^e \, \mathrm{d}V \; , \tag{14.47}$$

is a 3×3 submatrix of the element stiffness matrix: compare with the illustration in Fig. 14.3 (the off-diagonal block). Similarly, for $j = L$ and $k = L$ we obtain the diagonal block. Therefore, the entire element stiffness matrix may be computed in one shot as

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \, \mathrm{d}V \; , \tag{14.48}$$

using the blocked matrix (the element strain-displacement matrix)

$$\boldsymbol{B}_e = [\boldsymbol{B}_A^e, \boldsymbol{B}_B^e, \boldsymbol{B}_C^e, \boldsymbol{B}_D^e] \; . \tag{14.49}$$

Correspondingly, the displacements at the nodes will be ordered into a column vector of displacement components in the global Cartesian basis (with the Matlab syntax: semicolon means new line)

$$\boldsymbol{U}_e = [[u_A]; [u_B]; [u_C]; [u_D]] \; . \tag{14.50}$$

Exercise 75. For the T4 tetrahedron defined by the nodes

Node	x	y	z
K	2	0	0
L	0	3	0
M	0	0	0
P	0	0	4

compute the distribution of the mechanical strains due to the displacements

$$[u_K] = \begin{bmatrix} 0.1 \\ 0 \\ 0 \end{bmatrix} \qquad [u_P] = \begin{bmatrix} 0 \\ -0.4 \\ 0 \end{bmatrix}$$

(all displacements not given our identically zero).

Solution: The strains could be obtained by direct differentiation of the interpolation of the displacements using the finite element basis functions

$$u_x(x,y,z) = N_K(x,y,z) u_{Kx} + N_L(x,y,z) u_{Lx} + N_M(x,y,z) u_{Mx} + N_P(x,y,z) u_{Px}$$

and so on for the other components, so that for instance $\epsilon_x = \partial u_x/\partial x$

$$\epsilon_x = \partial u_x/\partial x = \frac{\partial N_K(x,y,z)}{\partial x}u_{Kx} + \frac{\partial N_L(x,y,z)}{\partial x}u_{Lx} + \frac{\partial N_M(x,y,z)}{\partial x}u_{Mx} + \frac{\partial N_P(x,y,z)}{\partial x}u_{Px}$$

As shown in Exercise 55, the linear functions N_K, N_L, N_M and N_P may be obtained for the tetrahedron (a simplex element) by direct solution for the coefficients of the linear functions.

```
>> X=[[2,0,0],1;
    [0,3,0],1;
    [0,0,0],1;
    [0,0,4],1];
A=inv(X)
A =
    0.5000         0   -0.5000         0
         0    0.3333   -0.3333         0
         0         0   -0.2500    0.2500
         0         0    1.0000         0
```

and therefore the matrix of (constant) basis function gradients is the transpose of the first three rows of `A`

```
>> A(1:3,:)'
ans =
    0.5000         0         0
         0    0.3333         0
   -0.5000   -0.3333   -0.2500
         0         0    0.2500
```

Recall the formula for ϵ_x above which upon substitution yields

$$\epsilon_x = (0.5)u_{Kx} + (0)u_{Lx} + (-0.5)u_{Mx} + (0)u_{Px} = (0.5)(0.1) = 0.05$$

Because the basis function gradients are uniform across the element, the strains are also uniform (constant) across the element.

Alternatively, we may use the strain-displacement matrix. Using the blocked strain-displacement matrix $\boldsymbol{B}_e$ of (14.49) we may write the strain vector from (14.45) as

$$[\boldsymbol{\epsilon}^e] = \boldsymbol{B}_e\boldsymbol{U}_e = [\boldsymbol{B}_K^e, \boldsymbol{B}_L^e, \boldsymbol{B}_M^e, \boldsymbol{B}_P^e]\begin{bmatrix}[u_K]\\ [u_L]\\ [u_M]\\ [u_P]\end{bmatrix} = \boldsymbol{B}_K^e[u_K] + \boldsymbol{B}_L^e[u_L] + \boldsymbol{B}_M^e[u_M] + \boldsymbol{B}_P^e[u_P]$$

The nodal strain displacement matrices of (14.44) are easily computed from the basis function gradients computed earlier. In our present case $[u_L]$ and $[u_M]$ are identically zero, and therefore the corresponding nodal strain displacement matrices need not be evaluated. The two remaining ones are

```
>> gradN =A(1:3,1)'% 1==K
B_K =[gradN(1),0,0;
      0,gradN(2),0;
      0,0,gradN(3);
      gradN(2),gradN(1),0;
      gradN(3),0,gradN(1);
      0,gradN(3),gradN(2)]
gradN =
    0.5000         0         0
B_K =
```

```
    0.5000         0         0
         0         0         0
         0         0         0
         0    0.5000         0
         0         0    0.5000
         0         0         0
```

and

```
>> gradN =A(1:3,4)'% 4==P
B_P =[gradN(1),0,0;
      0,gradN(2),0;
      0,0,gradN(3);
      gradN(2),gradN(1),0;
      gradN(3),0,gradN(1);
      0,gradN(3),gradN(2)]
gradN =
         0         0    0.2500
B_P =
         0         0         0
         0         0         0
         0         0    0.2500
         0         0         0
    0.2500         0         0
         0    0.2500         0
```

The contributions to the strain vector are: displacement of node K contributes

$$\boldsymbol{B}_K^e[u_K] = \begin{bmatrix} (0.5)\times(0.1) \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and displacement of node P contributes

$$\boldsymbol{B}_P^e[u_P] = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ (0.25)\times(-0.4) \end{bmatrix}$$

Thus we have the strain vector

$$\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \begin{bmatrix} 0.05 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.1 \end{bmatrix}$$

Exercise 76.

Consider displacements "radially" from the origin of the Cartesian coordinates

$\boldsymbol{u}(\boldsymbol{x}) = 0.001\boldsymbol{x}$

Verify that for the element given below the mechanical strains due to such radial displacements are purely volumetric (no shear). (Volumetric strain is defined as the relative change of volume, $\epsilon_v = \epsilon_x + \epsilon_y + \epsilon_z$.)

Node	x	y	z
K	-3	7	-1
L	0	9	1
M	-5	6	0
P	0	6	2

Solution: In order to compute the strains we need the gradients of the basis functions. These may be computed (for a change) from the formula (8.53). First the Jacobian matrix

```
>> Nder=[-1,-1,-1;
    1,0,0;
    0,1,0;
    0,0,1];
x= [-3,7,-1; 0,9,1; -5,6,0; 0,6,2];
J=x'*Nder
J =
     3    -2     3
     2    -1    -1
     2     1     3

>> Ndersp=Nder/J
Ndersp =
    0.2727   -0.2273   -0.6818
   -0.0909    0.4091    0.2273
   -0.3636    0.1364    0.4091
    0.1818   -0.3182    0.0455
```

The nodal strain-displacement matrices are computed as in Exercise 75. For instance

```
>> gradN =Ndersp(2,:) % 2==L
B_L =[gradN(1),0,0;
      0,gradN(2),0;
      0,0,gradN(3);
      gradN(2),gradN(1),0;
      gradN(3),0,gradN(1);
      0,gradN(3),gradN(2)]
gradN =
   -0.0909    0.4091    0.2273
B_L =
   -0.0909         0         0
         0    0.4091         0
         0         0    0.2273
    0.4091   -0.0909         0
    0.2273         0   -0.0909
         0    0.2273    0.4091
```

and so on for the other three. The nodal displacements are multiples of the location vectors

$$[u_K] = 0.001\begin{bmatrix}-3\\7\\-1\end{bmatrix}\ , \quad [u_L] = 0.001\begin{bmatrix}0\\9\\1\end{bmatrix}\ , \quad [u_M] = 0.001\begin{bmatrix}-5\\6\\0;\end{bmatrix}\ , \quad [u_P] = 0.001\begin{bmatrix}0\\6\\2\end{bmatrix}\ ,$$

and multiplying through results in the strain vector

```
>> B_K*(0.001*[-3,7,-1]')+B_L*(0.001*[0,9,1]') +B_M*(0.001*[-5,6,0]') +B_P*(0.001*[0,6,2]')
ans =
  1.0e-003 *
    1.0000
    1.0000
    1.0000
         0
   -0.0000
    0.0000
```

The first three components are equal to 0.001, the multiplier in the radial displacement expression! All normal strains are equal to this value, and yield the volumetric strain, also called dilatational strain as

$$\epsilon_v = \epsilon_x + \epsilon_y + \epsilon_z = 0.003$$

The last three components, the shear strains, are all zero (to numerical precision). Therefore our conjecture that radial displacements lead only to change of volume, no shear, is confirmed.

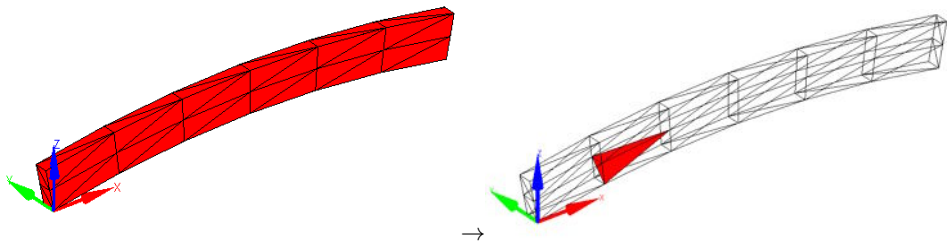
The T4 tetrahedron is hopelessly stiff for stress analyses. Bending problems are among those better resolved with other elements covered later in the book. This exercise shows the reason for the poor performance of the T4.

Exercise 77.

Consider "in-plane pure bending" displacements of a beam

$$u_x = -\theta(-1 + 2x/L)z, \quad u_y = 0, \quad u_z = \theta L(-x/L + x^2/L^2)$$

where $L = 6.13$, $\theta = -0.2$. The T4 element shown below is part of the mesh of the beam.



Element data:

Node	x	y	z
K	1.02167	0	0
L	1.02167	0.2	0.305
M	1.02167	0	0.305
P	2.04333	0	0.305

Solution: As in Exercise 76 the Jacobian matrix is evaluated as

```
>> x= [1.02167 0 0; 1.02167 0.2 0.305; 1.02167 0 0.305; 2.04333 0 0.305];
J=x'*Nder
J =
```

```
         0         0    1.0217
    0.2000         0         0
    0.3050    0.3050    0.3050
```

and the gradients of the basis functions follow as

```
>> Ndersp=Nder/J
Ndersp =
         0         0   -3.2787
         0    5.0000         0
   -0.9788   -5.0000    3.2787
    0.9788         0         0
```

The nodal strain-displacement matrices are computed as in Exercise 76, for instance

```
>> gradN =Ndersp(3,:) % 3==M
B_M =[gradN(1),0,0;
      0,gradN(2),0;
      0,0,gradN(3);
      gradN(2),gradN(1),0;
      gradN(3),0,gradN(1);
      0,gradN(3),gradN(2)]
gradN =
   -0.9788   -5.0000    3.2787
B_M =
   -0.9788         0         0
         0   -5.0000         0
         0         0    3.2787
   -5.0000   -0.9788         0
    3.2787         0   -0.9788
         0    3.2787   -5.0000
```

The displacements at the nodes are computed from the above formulas for the displacement components

Node	K	L	M	P
u_x	0	-0.0407	-0.0407	-0.0203
u_y	0	0	0	0
u_z	0.1703	0.1703	0.1703	0.2724

Finally, formula (14.45) is used to evaluate the strain

$$[\boldsymbol{\epsilon}^e] = \boldsymbol{B}^e_K[u_K] + \boldsymbol{B}^e_L[u_L] + \boldsymbol{B}^e_M[u_M] + \boldsymbol{B}^e_P[u_P]$$

which upon substitution of numerical values yields

$$\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \\ \gamma_{xz} \\ \gamma_{yz} \end{bmatrix} = \begin{bmatrix} \boxed{0.0199} \\ 0 \\ 0 \\ 0 \\ \boxed{\text{-0.0333}} \\ 0 \end{bmatrix}$$

The nonzero axial stretch in the first box is expected. The shear strain in the second box is undesirable (parasitic). It should not appear in pure bending situations. Here it does because the element does not have enough flexibility built in to deform as needed other than by shearing. This makes the

element too stiff since some energy needs to be invested into shearing while it should all go into bending.

14.8 Material directions and basis transformation

It remains to discuss the issue of the choice of coordinate systems when evaluating the stiffness matrix integrals. As discussed in Section 14.5, the components of the material stiffness matrix are expressed on the local Cartesian basis of material orientation directions. Referring to the definition of the stiffness matrix (14.30), we see that there is a need to transform between the local material directions and the global Cartesian basis. Drawing on the example of the element stiffness matrix (14.48), the element's restoring force may be expressed as

$$\boldsymbol{F}_e = \boldsymbol{K}_e \boldsymbol{U}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \, \mathrm{d}V \, \boldsymbol{U}_e \,, \tag{14.51}$$

While the displacement components in $\boldsymbol{U}_e$ and the force components in $\boldsymbol{F}_e$ are in the global Cartesian basis, the material stiffness matrix is expressed on the basis of the local material directions. Evidently, the linear algebra operations between the material matrix and displacements/forces must incorporate the transformation from one basis to another.

There are several ways in which this transformation could be carried out. In the `FAESOR` toolbox the following approach is used: the strain-displacement matrix is defined to produce strains in the local material basis, while taking displacement components in the global basis. This can be accomplished by writing

$$\begin{aligned} [\boldsymbol{\epsilon}]^{(\overline{x})} = \mathcal{B}^{\overline{x}} [\boldsymbol{u}]^{(\overline{x})} = \mathcal{B}^{(\overline{x})} \left([T] [\boldsymbol{u}]^{(x)} \right) = \\ \left(\mathcal{B}^{(\overline{x})} [T] \right) [\boldsymbol{u}]^{(x)} = \mathcal{B}^{(\overline{x},x)} [\boldsymbol{u}]^{(x)} \end{aligned} \tag{14.52}$$

where we use the superscript $(\overline{x})$ or (x) to indicate in which coordinate system the components are expressed. The strain-displacement operator $\mathcal{B}^{(\overline{x},x)} = \mathcal{B}^{(\overline{x})}[T]$ incorporates the geometric transformation $[T]$ of the displacement vector components from the global basis into the local material directions basis. This transformation may be derived from the equality

$$\left[\, \boldsymbol{e}_x, \, \boldsymbol{e}_y, \, \boldsymbol{e}_z \, \right] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \left[\, \boldsymbol{e}_{\overline{x}}, \, \boldsymbol{e}_{\overline{y}}, \, \boldsymbol{e}_{\overline{z}} \, \right] \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \,. \tag{14.53}$$

Pre-multiplying with

$$\begin{bmatrix} \boldsymbol{e}_{\overline{x}} \\ \boldsymbol{e}_{\overline{y}} \\ \boldsymbol{e}_{\overline{z}} \end{bmatrix}$$

leads to the transformation of vector components

$$[\boldsymbol{R}_m]^T \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = [T] \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix} = \begin{bmatrix} u_{\overline{x}} \\ u_{\overline{y}} \\ u_{\overline{z}} \end{bmatrix} \,. \tag{14.54}$$

The transformation $[T]$ is thus identified with an orthogonal (rotation) matrix

$$[\boldsymbol{R}_m] = \begin{bmatrix} \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_x, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_x \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_y, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_y \\ \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{y}} \cdot \boldsymbol{e}_z, & \boldsymbol{e}_{\overline{z}} \cdot \boldsymbol{e}_z \end{bmatrix} \,, \tag{14.55}$$

which is just a three-dimensional analog of the two-dimensional transformation (8.66).

The global-to-local ***strain-displacement matrix*** is therefore defined as

$$\boldsymbol{B}_k^e = \mathcal{B}^{(\overline{x},x)}\left(N_k(\boldsymbol{x})\right) = \mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)[\boldsymbol{R}_m]^T \text{ for } \boldsymbol{x} \in \text{element } e. \tag{14.56}$$

The $\boldsymbol{B}_k^e$ nodal matrices are used to compose the element strain-displacement matrix (14.49). In the `FAESOR` toolbox, the strain-displacement matrix is computed for a three-dimensional solid geometric cell by the private method `feblock_defor_ss_Blmat3` defined for the class `feblock_defor_ss`:

```
0024 function B = feblock_defor_ss_Blmat3[1](self,N,Ndersp,c,Rm)
0025     nfn= size(Ndersp,1);
0026     B = zeros(6,nfn*3); %initialize
0027     if (isempty(Rm)) % there is no global-to-local transformation
0028         for i= 1:nfn
0029             k=3*(i-1);
0030             B(1,k+1)= Ndersp(i,1);
0031             B(2,k+2)= Ndersp(i,2);
0032             B(3,k+3)= Ndersp(i,3) ;
0033             B(4,k+1)= Ndersp(i,2); B(4,k+2)= Ndersp(i,1);
0034             B(5,k+1)= Ndersp(i,3); B(5,k+3)= Ndersp(i,1);
0035             B(6,k+2)= Ndersp(i,3); B(6,k+3)= Ndersp(i,2);
0036         end
0037     else % global-to-local transformation is requested
0038         for i= 1:nfn
0039             B(:,3*(i-1)+1:3*i)...
0040                 = [ Ndersp(i,1) 0           0  ; ...
0041                 0           Ndersp(i,2) 0  ; ...
0042                 0           0           Ndersp(i,3) ; ...
0043                 Ndersp(i,2) Ndersp(i,1) 0  ; ...
0044                 Ndersp(i,3) 0           Ndersp(i,1) ; ...
0045                 0           Ndersp(i,3) Ndersp(i,2) ]*Rm';
0046         end
0047     end
0048     return;
0049 end
```

The input arguments are the values of the basis functions `N` and the values of the space gradients of the basis functions `Ndersp`, where both arrays are computed at the quadrature point, the coordinates of the quadrature point `c` (the 3-D version of (8.55)), and the global-to-local transformation matrix $\boldsymbol{R}_m$. Comparing with the definitions (14.56) and (14.49), the Matlab code is an almost literal translation of these formulas. Note that the global location vectors for each node `x` are transformed into the local Cartesian basis of material directions, `x*Rm`, as the matrix $\mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)$ in Eq. (14.56) requires derivatives with respect to the material basis.

The method `feblock_defor_ss_Blmat3` is for 3-D solid elements defined for the finite element block class `feblock_defor_ss`, because this matrix is specific to the equations of elastodynamics, and class `feblock_defor_ss` is specialized for application. Therefore it is also a private method, meaning that it cannot be called from outside of the scope of the public methods for this class. Analogously, one-dimensional and two-dimensional elements have their versions of this method defined for their particular needs.

14.9 Stiffness matrix

In Chapter 6 and further in Section 8.12, it was pointed out that all the problem dependent code was kept in a descendent of the `feblock` class. The elastodynamics model of this chapter is implemented in the `feblock_defor_ss` finite element block class.

[1] Folder: `FAESOR/classes/feblock/@feblock_defor_ss/private`

The stiffness matrix (14.30) is assembled from the element-wise stiffness matrices (14.48). These are calculated by the `stiffness` method, and returned in the `ems` array. The `stiffness` method takes as arguments the geometry of the mesh (field `geom`), in order to have access to the locations of the nodes, and the displacement field `u` to provide the global equation numbers of the unknowns. First the integration point data is established, including the values of the basis functions and the values of the gradients of the basis functions with respect to the parametric coordinates.

```
0008 function ems = stiffness[2] (self, geom, u)
0009     gcells = get(self.feblock,'gcells');
0010     nfens = get(gcells,'nfens');
0011     % Integration rule
0012     integration_rule = get(self.feblock, 'integration_rule');
0013     pc = get(integration_rule, 'param_coords');
0014     w  = get(integration_rule, 'weights');
0015     npts_per_gcell = get(integration_rule, 'npts');
0016     for j=1:npts_per_gcell
0017         Ns{j} = bfun(gcells,pc(j,:));
0018         Nders{j} = bfundpar(gcells,pc(j,:));
0019     end
```

To deal with the global-to-local transformations for materials which have general preferred directions (not coaxial with the global Cartesian axes) is expensive. Therefore, we try to find out if these transformations are indeed necessary. Considerable savings of processing time may be realized if the matrix is a constant for the entire block; otherwise the transformation matrix will be computed at each quadrature point for each finite element.

```
0020     % Material orientation /directions
0021     Rm = get(self,'Rm');
0022     eval_Rm = 0;
0023     if strcmp(class(Rm),'function_handle')
0024         eval_Rm = true; Rmh =Rm;
0025     end
0026     % Material
0027     mat = get(self.feblock, 'mater');
0028     % Now loop over all gcells in the block
0029     conns = get(gcells, 'conn'); % connectivity
0030     Ke = cell(size(conns,1),1);
0031     eqnums = cell(size(conns,1),1);
0032     for i=1:size(conns,1)
0033         eqnums{i} =gather(u,conns(i,:),'eqnums');
0034         Ke{i} =zeros(get(geom,'dim')*nfens);
0035     end
0036     xs =gather(geom,(1:get(geom,'nfens')),'values','noreshape');
```

After the connectivity, element matrix data, and the array of the coordinates of the nodes was prepared, we are ready to enter the two loops, one over all the elements, and one over all the quadrature points within each element. The connectivity of the geometric cell is retrieved, and, based on the connectivity, the locations of the nodes `x` are gathered from the geometry field `geom`.

```
0037     for i=1:size(conns,1)
0038         conn =conns(i,:);
0039         x=xs(conn,:);
```

The physical location of the quadrature point within the global Cartesian coordinates is computed (this would be needed for inhomogeneous materials). Next the Jacobian matrix is evaluated.

[2] Folder: `FAESOR/classes/feblock/@feblock_defor_ss`

```
0040         for j=1:npts_per_gcell
0041             c =Ns{j}'*x;% physical location of the quadrature point
0042             J = x' * Nders{j};% We compute the Jacobian matrix
```

If necessary, the global-to-local transformation (the local material directions) matrix is computed. It may depend on the location of the quadrature point, and/or on the tangents to the parametric curves passing through the quadrature point (which are the columns of the Jacobian matrix), or indeed on any other user-defined parameter. If required, these quantities would be calculated by the function `Rmh` from the supplied arguments.

```
0043             if (eval_Rm)% need to evaluate the local material orientation?
0044                 Rm =Rmh(c,J);
0045             end
```

Compute the Jacobian associated with the integration point, and since the integration is to be performed over the 3-D volume, the method `Jacobian_volume` needs to be used; for the solid elements it is identical to the `Jacobian` method.

```
0046             Jac = Jacobian_volume(gcells,conn, Ns{j}, J, x);
```

Evaluate the gradients of the basis functions with respect to the space coordinates.

```
0047             Ndersp = Nders{j}/(Rm'*J);% derivatives wrt local coor
```

The strain-displacement matrix for the element is calculated from the spatial gradients of the basis functions, and the matrix of material directions. Note that for 3-D solid elements the function `self.hBlmat` is `feblock_defor_ss_Blmat3` discussed above. However, this code will actually work for all elements from L2 to Q4 to H20 (which we haven't discussed yet, but we will), in 1-, 2-, and 3-coordinate models because the finite element block will select the private method to compute the strain-displacement matrix appropriately for the number of coordinates. More on this in Chapter 17 that deals with model-dimension reduction for plane strain, plane stress, and axially symmetric analysis.

```
0048             B = self.hBlmat(self,Ns{j},Ndersp,c,Rm);% strain-displacement
```

The material stiffness matrix is calculated by the material object `mat`, and since for inhomogeneous materials it may change from point to point, the location of the current integration point, `c`, is passed to the method.

```
0049             D = tangent_moduli(mat,struct('xyz',c));
```

The product of the strain-displacement matrix and the tangent moduli (the material stiffness matrix) is accumulated in the element stiffness matrix `Ke{i}`.

```
0050             Ke{i} = Ke{i} + (B'*(D*(Jac*w(j)))*B);
0051         end
0052     end
```

Finally, the computed element stiffness matrices are stored in the `ems` object of the `elematset` class. Note that the equation numbers are gathered from the displacement field.

```
0053     ems  = elematset(struct('mat',{Ke}, 'eqnums',{eqnums}));
0054 end
```

14.10 Pure-traction problems and singular stiffness

For pure-traction problems, first introduced in Section 13.6.6, the stiffness matrix will become singular (not of full rank). It doesn't take too much effort to verify that strains computed from the rigid-body displacements (13.51) are all identically zero. Therefore, also the deformation energy induced by the rigid body motion is zero, as is easily verified by referring to (13.44). Another way

of calculating the energy of deformation is by invoking the formula for the strains in terms of the element displacements, $\boldsymbol{\epsilon} = \boldsymbol{B}^e \boldsymbol{U}_e$, and the assembly of the element stiffness matrices:

$$\begin{aligned}\Phi(\boldsymbol{\epsilon}) = \int_V \phi(\boldsymbol{\epsilon})\, \mathrm{d}V = \sum_e \int_{V_e} \phi(\boldsymbol{\epsilon})\, \mathrm{d}V = \\ \int_V \frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{D}\boldsymbol{\epsilon}\, \mathrm{d}V = \sum_e \frac{1}{2}\boldsymbol{U}_e^T \int_{V_e} \boldsymbol{B}^{eT}\boldsymbol{D}\boldsymbol{B}^e\, \mathrm{d}V\, \boldsymbol{U}_e = \\ \sum_e \frac{1}{2}\boldsymbol{U}_e^T \boldsymbol{K}_e \boldsymbol{U}_e = \frac{1}{2}\boldsymbol{U}^T \boldsymbol{K}\boldsymbol{U}\ . \end{aligned} \tag{14.57}$$

Since this energy vanishes for a nonzero displacement $\boldsymbol{U} \neq \boldsymbol{0}$, the global matrix $\boldsymbol{K}$ must be singular to produce a positive semi-definite quadratic form $\frac{1}{2}\boldsymbol{U}^T\boldsymbol{K}\boldsymbol{U} \geq 0$. The stiffness matrix falls short of the full rank by the number of possible rigid body modes: six, when all three rotations and translations are possible, or less. The rigid-body displacements must be prevented; additional supports, or springs grounding the structure are common solutions.

Exercises

1. For an isotropic material, consider whether the material parameter values produce a reasonable (that is positive definite) material stiffness matrix. Provide sufficient detail to argue your point.
   a) Young's modulus $E = 0$, Poisson's ratio $\nu > 0$.
   b) Young's modulus $E > 0$, Poisson's ratio $\nu = 0$.
   c) Young's modulus $E > 0$, Poisson's ratio $\nu = 1/2$.
   d) Young's modulus $E > 0$, Poisson's ratio $\nu = -1/4$.

15

Finite Elements for true 3-D Problems

With the formulation sketched out, it only remains to pick a finite element to be able to perform an analysis. We begin with the tetrahedron T4. All the necessary formulas have been put into place in Section 11.5, and we may immediately proceed to the first example.

The equation of motion (14.33) stands for the so-called free vibration when there is no forcing, $\boldsymbol{L} = \boldsymbol{0}$; compare also with (3.20). The built-in Matlab eigenvalue solver will be used to obtain the numerical solution, and the task for the `FAESOR` script is relatively simple: compute the stiffness matrix and the mass matrix.

15.1 Modal analysis with the tetrahedron T4: the drum

As our first example, we consider the vibration of a moderately thick circular plate as shown in Fig. 15.1. The cylindrical surface is fully clamped (all displacements zero). The material is isotropic. The analytical solution has been worked out in dependence on the number of "nodes" (locations of approximately zero displacement) radially and circumferentially [B01]. Therefore, this is a very good example with which to test the finite element formulation.

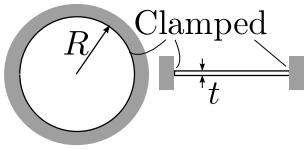


Fig. 15.1. Clamped circular plate (drum).

The Matlab script `drum_t4`[1] solves the vibration problem for the four lowest eigenvalues (shown in Fig. 15.2). First, a few variables are defined and a mesh is produced using the mesh function `t4cylinderdel`[2]. The mesh is relatively coarse, with only two element edges through the thickness and seven element edges radially.

```
0001 E=0.1e6;% Pa
0002 nu=0.3;
0003 rho=1000;% kg
0004 R= 25.0e-3;% m, radius
0005 t= 2.0e-3;% m, thickness
...
0010 [fens,gcells] = t4cylinderdel(t,R, 2,7);
```

[1] Folder: `FAESOR/examples/stress`
[2] Folder: `FAESOR/meshing`

The material is small-strain (`ss`), linearly elastic (`linel`), isotropic (`iso`), and triaxial (`triax`). The property class `property_linel_iso` supplies methods to compute the material stiffness matrix for this type of material. Each type of material has its own property class, and the material class `mater_defor_ss_linel_triax` undertakes to insulate the methods of the finite element block from the details of the properties. Note that the mass density also needs to be supplied (dynamics!).

```
0012 prop=property_linel_iso(struct('E',E,'nu',nu,'rho',rho));
0013 mater=mater_defor_ss_linel_triax(struct('property',prop));
```

The finite element block is of class `feblock_defor_ss`. The integration rule is a one-point quadrature, provided by the class `tet_rule`.

```
0013 % Finite element block
0014 feb = feblock_defor_ss (struct ('mater',mater, 'gcells',gcells,...
0015     'integration_rule',tet_rule (1)));
```

The geometry field and the displacement field are set up as usual. The essential boundary condition is applied to all finite element nodes on the cylindrical surface (line `0024`).

```
0018 geom=field(struct ('name',['geom'],'dim',3,'fens',fens));
0019 % Define the displacement field
0020 u   = 0*geom; % zero out
0021 % Apply EBC's
0022 for i=1:length(fens)
0023     xyz = get(fens(i),'xyz');
0024     if abs(R-norm(xyz(2:3))) <0.001*R
0025         u   = set_ebc(u, i, [1], [], 0.0);
0026     end
0027 end
0028 u   = apply_ebc (u);
0029 % Number equations
0030 u   = numbereqns (u);
```

The two methods, `stiffness` and `mass`, discussed previously, are invoked. Note that the method `mass` computes the consistent element mass matrices, and the free vibration problem solution will be sought with a consistent mass matrix.

```
0032 K = start (sparse_sysmat, get(u, 'neqns'));
0033 K = assemble (K, stiffness(feb, geom, u));
0034 M = start (sparse_sysmat, get(u, 'neqns'));
0035 M = assemble (M, mass(feb, geom, u));
```

Finally, the built-in Matlab solver `eigs` is called. Four lowest eigenvalues are requested (the flag `'SM'`). The returned eigenvalues need to be sorted from the smallest to the largest (line `0039-0040`).

```
0037 neigvs = 4;
0038 [W,Omega]=eigs(get(K,'mat'),get(M,'mat'),neigvs,'SM');
0039 [Omegas,ix]=sort(diag(Omega));
0040 Omega= diag(Omegas);
```

An interesting feature of the algorithms implemented in `eigs` is that they depend on random inputs. If we would like to get the same results every time we call this script, the Matlab built-in generator of random numbers `rand` needs to be reset to the same state; otherwise the results will show a random variation.

```
0006 rand('state',0);% try to comment out this line and compare
0007 %                   results for several subsequent runs
```

The plotting section of the script is omitted, but the resulting shapes and the calculated frequencies are summarized in Fig. 15.2: taking the analytical solution as reference, it is clear that the errors are huge. Evidently, a much more refined mesh would be required to obtain reasonable answers (recall: smaller elements, smaller error). A valuable observation: all the calculated values are above the reference frequencies. This is called "convergence from above", and it is an indication that the discrete model (in other words, the T4 finite element) is *too stiff*.

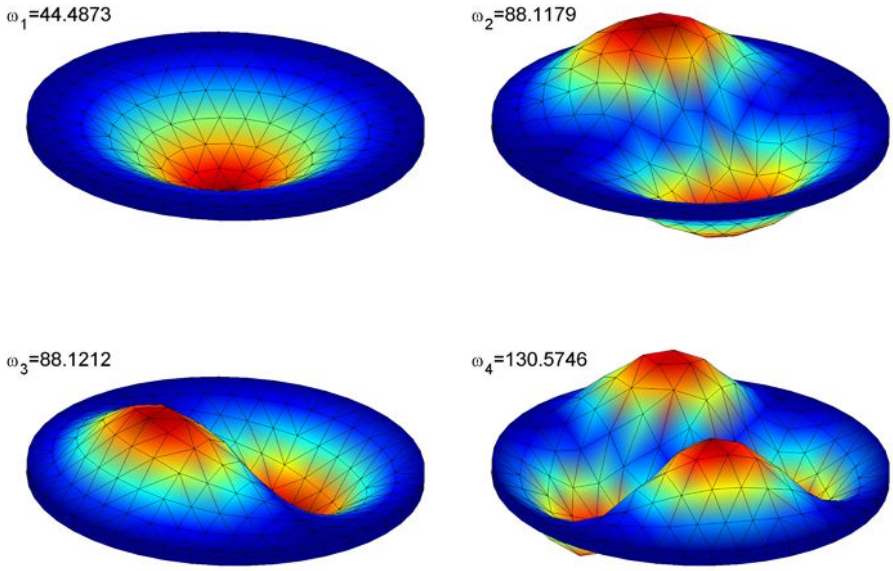


Fig. 15.2. Mode shapes of thick clamped plate. The analytical solution for the natural frequencies yields for these modes $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz

15.2 Modal analysis with the tetrahedron T4: the composite rod

As the second example we will again consider a free vibration problem, but this time the material of the structure will be modeled as transversely isotropic. The structure is a straight rod of circular cross-section, manufactured from carbon fiber infused with polymer resin (Fig. 15.3). The reinforcing fibers are twisted, and for lack of other information, we assume that the twist angle decreases from the outside surface (15°) to zero along its axis. The rod is clamped at both ends. Of interest is the lowest natural frequency of the structure.

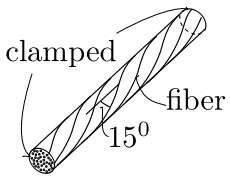


Fig. 15.3. The composite rod.

The Matlab solution of the problem is in the form of a function, `twist_t4`[3], in order to allow for a convergence analysis to be performed on a series of meshes. Therefore, an internal function will be run repeatedly to solve for the first natural frequency for different resolutions radially (number of element edges `nR`) and longitudinally (number of element edges `nt`).

```
0001 function twist_t4
0002     nR = [2, 3, 4];
0003     nt = [30, 40, 50, 60, 70];
0004     fs =zeros(length(nR),length(nt));
0005     rand('state',0);% try to comment out this line and compare
0006 %                   results for several subsequent runs
0007     for i=1:length(nR)
0008         for j=1:length(nt)
0009             fs(i,j) =do_twist_t4(nR(i),nt(j))
0010         end
0011     end
...
0014 function frequency1 =do_twist_t4(nR,nt)
0015  ...
```

Note that on line 0012 and below we test whether the norm of the difference between the computer frequencies and constant matrix is below a certain tolerance (`1e-6`). The result is assigned to variable `faesor_test_passed` as a logical value (true if the test passed). The constant matrix with which the computed frequencies are compared consists of the reference values of the frequencies. This allows us to automatically test the `FAESOR` every time a change is made to its internal code. If the toolbox code still works, the computed values of the frequencies will be identical or close to the reference values, and the code is then said to be verified; otherwise the verification test fails and the offending code needs to be corrected. ***Verification*** is a big deal in professional-quality simulation software, and most commercial packages of repute include dozens or even hundreds of verification problems that the software recomputes regularly as part of a quality assurance process. In the `FAESOR` toolbox the function `run_verification` runs all verification examples and writes out a log file summarizing the results.

```
0012      assignin('caller','faesor_test_passed',(norm(...
0013         1.0e+003 *[2.306698566966278   ...   1.923588876963714]-fs)<1e-6));
0016 end
```

The transversely isotropic material from Section 14.5.3 requires the definition of the local material orientation matrix (13.12) at each integration point. The axis $\overline{x}$ needs to be oriented along the fibers; the orientation of the remaining two basis vectors in the isotropy plane is arbitrary. In the function `twist`, the orientation matrix is derived by first turning a basis triad around the first vector (line 0035-0037), and then twisting the intermediate triad by an angle ramped up from 0° to 15° proportionally to the distance from the axis of the rod (line 0038).

```
0033     function Rm = default (XYZ, ts)
0034         Rm= eye(3);
0035     end
0036     function Rm = twist (XYZ, ts)
0037         r= norm(XYZ( 2:3));
0038         if r>0
0039             y=XYZ(2);z=XYZ(3);
0040             e2 = [0, y/r, z/r]';
0041             e3 = skewmat([1, 0, 0])*e2;
0042             Rm= [[1, 0, 0]',e2,e3];
0043             Rm=rotmat(r/R*twist_angle*skewmat(e2))*Rm;
```

[3]Folder: `FAESOR/examples/stress`

```
0044          else
0045              Rm= eye(3);
0046          end
```

The property class `property_linel_transv_iso` defines the material properties for the transversely isotropic material model. Note that five independent material constants need to be supplied.

```
0051      prop = property_linel_transv_iso (...
0052          struct('E1',E1,'E2',E2,'G12',G12,'nu12',nu12,'nu23',nu23,'rho',rho));
0053      mater = mater_defor_ss_linel_triax (struct('property',prop));
```

It is noteworthy that the function `twist` that defines the directions of the local material basis is supplied to the finite element block as a function handle. Call for Section 14.9 for details on the use of the material orientation matrix.

```
0055      feb = feblock_defor_ss (struct ('mater',mater, 'gcells',gcells,...
0056          'integration_rule',tet_rule (integration_order),'Rm',@twist));
```

The rest of the function `twist_t4` is omitted. The first mode shape of the composite rod for a relatively fine mesh is shown in Fig. 15.4: notice that the rod twists as well as bends due to the orientation of the reinforcing fibers in the form of a helix.

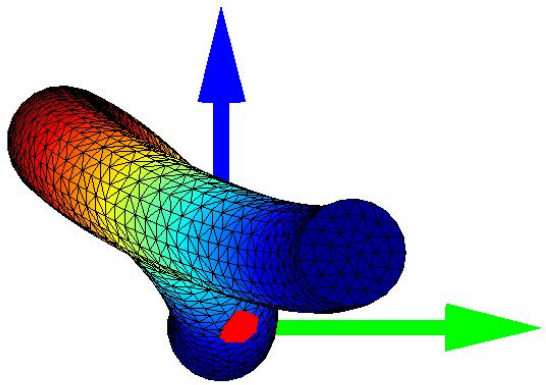


Fig. 15.4. The shape of the first eigenmode.

Let us now address the issue of convergence: The lowest frequency has been computed for a number of meshes with varying number of elements radially, and along the length of the rod. The lowest natural frequency is displayed for these different meshes in Fig. 15.5. The frequency varies from approximately 2300Hz to slightly less than 1900Hz. It should be noted that the first natural frequency decreases in magnitude with refinement (convergence from above), but the resulting surface is not smooth. (The bad shapes of some elements in the mesh have a profound effect on the accuracy.) The numerical answers are still changing significantly with the refinement, and it is therefore not clear how far from the "exact" solution we might be.

15.3 Tetrahedron T10

The quadratic tetrahedron is a simple extension of the quadratic triangle to three dimensions. The same idea of constructing the basis functions as (normalized) products of planes will work, yielding for the basis functions the quadratic expressions in the parametric coordinates ξ, η, ζ

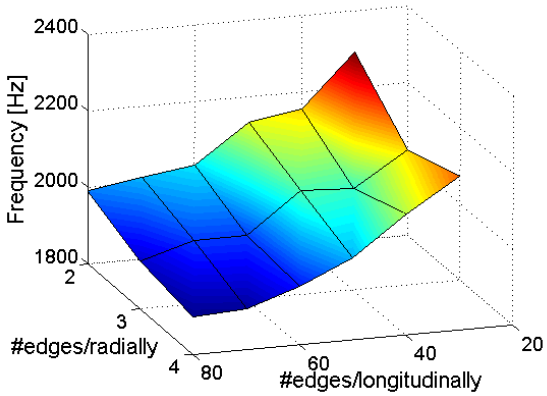


Fig. 15.5. Convergence of the lowest natural frequency with the T4 element.

$$
\begin{aligned}
N_1 = (1-\xi-\eta-\zeta)(2(1-\xi-\eta-\zeta)-1)\ , \quad N_2 = \xi(2\xi-1)\ ,\\
N_3 = \eta(2\eta-1)\ , \quad N_4 = \zeta(2\zeta-1)\ , \quad N_5 = 4(1-\xi-\eta-\zeta)\xi\ ,\\
N_6 = 4\xi\eta\ , \quad N_7 = 4\eta(1-\xi-\eta-\zeta)\ , \quad N_8 = 4(1-\xi-\eta-\zeta)\zeta\ ,\\
N_9 = 4\xi\zeta\ , \quad N_{10} = 4\eta\zeta\ .
\end{aligned}
\tag{15.1}
$$

Since the basis functions are quadratic in the parametric coordinates, their gradients with respect to the parametric coordinates will be linear functions of ξ, η, ζ. Provided the Jacobian matrix in equation (8.50) is constant (independent of ξ, η, ζ), the gradients of the basis functions with respect to x, y, z as computed from (8.45) are going to be linear in those coordinates (see Section 12.7 for details). Hence, computing the elements of the stiffness matrix can be done exactly with the four-point rule from Table 11.2. To the contrary, for curved elements (when a mid-edge node is moved from the mean of the locations of the corners) the four-point rule is not going to be able to integrate the stiffness matrix exactly. However, when the distortion of the element is not excessive, the integration error is not significant.

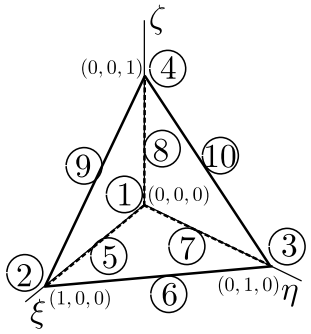


Fig. 15.6. The standard quadratic tetrahedron for the T10 element.

15.3.1 Example: the drum revisited

The Matlab script `drum_t10`[4] is a variation on `drum_t4` which applies the quadratic element instead of T4. The natural frequencies obtained for different meshes with the two elements, T4, and T10, are illustrated in Fig. 15.7. While increasing the number of unknowns by making the elements (uniformly)

[4]Folder: `FAESOR/examples/stress/3-D`

smaller leads generally to more accurate answers, the element T4 is clearly not performing very well. Even for a fairly fine mesh, the results are probably not of much use. On the contrary, the element T10 produces estimates of the frequencies which are of good engineering accuracy (within a fraction of a percent). The first frequency is estimated quite well even with the coarsest mesh (only a single element through the thickness, and two elements radially).

The graph of Fig. 15.7 may serve as a crude guide to the relative accuracy of the two tetrahedral elements. Sometimes the linear tetrahedron will fare better than in this example, oftentimes much worse.

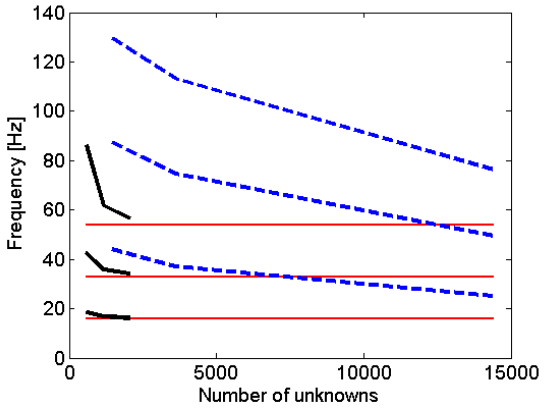


Fig. 15.7. Comparison of the first four natural frequencies of the drum computed with the two tetrahedral elements, the T4 (dashed line) and the T10 (solid line). The analytical solution for the natural frequencies, $\omega_1 = 15.7511$Hz, $\omega_2 = \omega_3 = 32.7659$Hz, $\omega_4 = 53.757$Hz, is indicated with horizontal lines.

15.4 The composite rod with the tetrahedron T10

The problem from Section 15.2 is addressed here with the quadratic tetrahedron T10. Remarkably, the Matlab function `twist_t10`[5] differs from `twist_t4` in only a couple of lines. Firstly, the mesh generated for the rod is converted from the type T4 to the type T10 by inserting additional nodes at the midpoints of the edges with the meshing function `T4_to_T10`[6].

```
0042      [fens,gcells] = t4cylinderdel(t,R, nt,nR);
0043      [fens,gcells] = T4_to_T10(fens,gcells);
```

Secondly, the integration rule is boosted to a four-point formula.

```
0049      feb = feblock_defor_ss (struct ('mater',mater,
                 'gcells',gcells,...
0050           'integration_rule',tet_rule (4),
                 'Rm',@twist));
```

Otherwise, the two functions are identical. The results are very different though. The higher order tetrahedron converges very quickly and produces monotonically converging answer for the lowest frequency (approximately 1766.6Hz). The convergence behavior with respect to the number of elements radially and longitudinally is compared with the earlier analysis with T4 in Fig. 15.8.

[5] Folder: `FAESOR/examples/stress`
[6] Folder: `FAESOR/meshing`

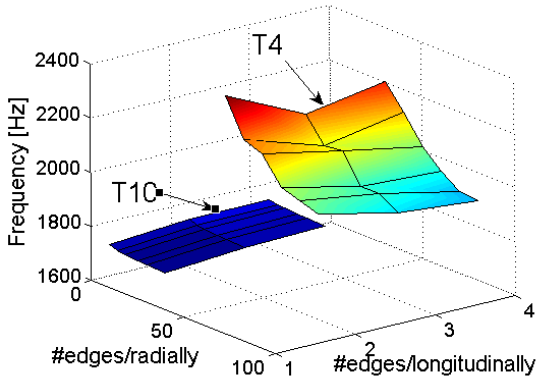


Fig. 15.8. Convergence of the lowest natural frequency. Comparison of the two tetrahedral elements.

15.5 Static analysis with hexahedra H8 and H20

15.5.1 Hexahedron H8

This element is a straightforward extension of the Q4 quadrilateral to three dimensions. In fact, the Q4 quadrilateral is compatible with the discretization on the faces of the hexahedron, which is essential for the implementation of the integrals of the surface terms (heat flux or traction loads).

The numbering of the nodes is given in Fig. 11.12. The basis function N_1 may be written as the product of one one-dimensional Lagrange interpolation function on the interval $-1 \leq \xi \leq +1$, one on the interval $-1 \leq \eta \leq +1$, and one on the interval $-1 \leq \zeta \leq +1$; all functions correspond to the left-hand side end of the interval

$$N_1(\xi,\eta,\zeta) = \frac{\xi-1}{-1-1} \times \frac{\eta-1}{-1-1} \times \frac{\zeta-1}{-1-1} = \frac{(\xi-1)(\eta-1)(\zeta-1)}{8} \,. \tag{15.2}$$

Analogously for the remaining functions: see the class method `bfun`[7] of the class `gcell_H8`.

Gauss integration at $2\times2\times2$ points is considered a "full integration" of the stiffness (conductivity) matrix. There are also other, more economical integration rules (for instance, six-point mid-face rule). One-point integration at the centroid is not sufficient, and leads to rank-deficient element matrices.

15.5.2 Dilatational locking

Consider the split of the strains into two groups: relative change of volume, and distortional strain. The relative change in volume ϵ_v (***volumetric strain***, also called dilatational strain) may be written as

$$\epsilon_v = \boldsymbol{m}^T \boldsymbol{\epsilon} \,, \tag{15.3}$$

where we use the notation of Reference [ZT89], namely the operator

$$[\boldsymbol{m}]^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix} \,.$$

The part of the strain that corresponds to volume change is therefore

$$\boldsymbol{\epsilon}_v = \frac{1}{3} \boldsymbol{m}\boldsymbol{m}^T \boldsymbol{\epsilon} \,.$$

Correspondingly, the ***deviatoric*** (distortional) ***strain*** is

[7] Folder: `FAESOR/classes/gcell/@gcell_H8`

$$\boldsymbol{\epsilon}_d = \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_v = \boldsymbol{\epsilon} - \frac{1}{3}\boldsymbol{m}\boldsymbol{m}^T\boldsymbol{\epsilon} = \boldsymbol{I}_d\boldsymbol{\epsilon} \ , \tag{15.4}$$

where we use the deviatoric projector

$$\boldsymbol{I}_d = \mathbf{1} - \frac{1}{3}\boldsymbol{m}\boldsymbol{m}^T \ .$$

For *isotropic* elastic materials, applying the split of the strains in the constitutive equation, $\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon}$, leads after some manipulations to the form that separates the two kinds of effects also in the stress

$$\boldsymbol{\sigma} = \boldsymbol{D}\boldsymbol{\epsilon} = K\boldsymbol{m}\epsilon_v + 2G\boldsymbol{I}_0\boldsymbol{\epsilon}_d = K\boldsymbol{m}\boldsymbol{m}^T\boldsymbol{\epsilon} + 2G\boldsymbol{I}_0\boldsymbol{I}_d\boldsymbol{\epsilon} \ , \tag{15.5}$$

where $K = E/3/(1-2\nu)$ is the ***bulk modulus***, $G = E/2/(1+\nu)$ is the shear modulus, and

$$\boldsymbol{I}_0 = \text{diag}\left[1, 1, 1, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}\right] \ .$$

As the Poisson's ratio approaches $\nu \to 1/2$, the elasticity model describes deformation which approaches the state of zero volume change. In other words, most (in the limit $\nu = 1/2$, all) of the deformation will be limited to distortion. Material models reproducing this behavior are called ***incompressible***. They are suitable for rubbery materials and fluid-saturated tissues, among others.

The bulk modulus approaches infinity for $\nu \to 1/2$. Therefore, the first part of the material stiffness matrix (15.5) grows without bounds as $\nu \to 1/2$. The consequence for the finite element formulation is that the large (and in the limit, infinite) stiffness $K\boldsymbol{m}\boldsymbol{m}^T$ will penalize any and all deformations that result in a non-zero relative volume change. Hence, unless the kinematics of deformation that a finite element provides allows for zero volumetric deformation to occur with $\boldsymbol{\epsilon} \neq \mathbf{0}$, the magnitude of possible strains will be severely limited, and in the worst case no deformation will be possible at all. Such a state of affairs is called ***dilatational locking***.

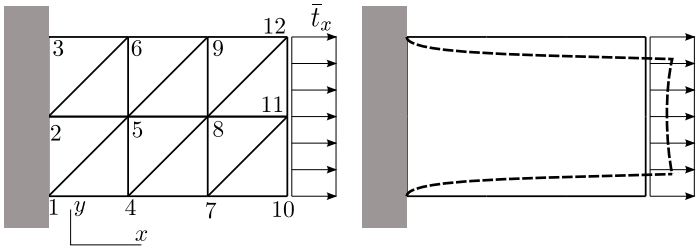


Fig. 15.9. Dilatational locking in a mesh composed of triangles

Figure 15.9 supplies a visual explanation. Consider the case $K \to \infty$ with the mesh shown as a collection of triangles, but similar reasoning would apply to tetrahedra. The base in the triangle 152 is fixed. To conserve its volume, node 5 is allowed to move only vertically. Analogously for triangle 263, and to conserve the volume of triangle 256, the two nodes 5 and 6 must move by the same amount. Similarly we prove that node 9 must move only vertically. Compare these severe limitations on the deformation allowed by the incompressibility constrains with the expected shape produced by a tensile load in the x direction: clearly the results are not going to be very useful. In fact, it is easy to produce situations where the incompressibility would not allow any displacement at all at the nodes 2, 3, 5, 6, 9: For instance, consider the effect of applying a roller condition in the vertical direction on the upper edge, such as would be used to model an axis of symmetry.

The triangles T3 and tetrahedra T4 are in this way severely exposed to locking for almost incompressible materials. The deformations that they can experience are very limited as soon as we require zero (or very small) volumetric change.

Let us look at an example to study the performance of the hexahedron H8. Figure 15.10 should be referred to as to the geometry and the load (uniform shear traction on the free end), but the

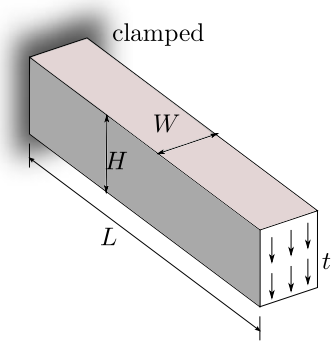


Fig. 15.10. Beam of almost incompressible material with transverse load

important detail is that the material has a Poisson's ratio of $\nu = 0.4999$ (a number representative of some filled rubbers). (A comprehensive study is described later in Section 15.5.9; the analysis is run by the Matlab script `rltb`[8].) Figure 15.11 shows the coarsest and the finest mesh with a deflected shape.

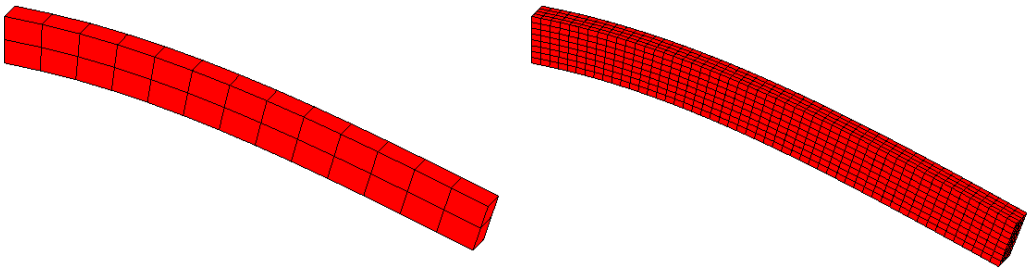


Fig. 15.11. Meshes for the clamped beam

The lower curve in Fig. 15.12 belongs to the regular H8, and it clearly illustrates that it also suffers severe dilatation locking (very slow convergence). However, in this case there is a remedy. The numerical integration of the stiffness matrix essentially imposes the constraint of (almost) zero volumetric strain at each integration point. Since this is impossible to meet for H8, the element locks. However, the technique known under the name of ***selective reduced integration*** [H00] may be applied to alleviate the symptoms. Essentially, the two terms in (15.5) split the stiffness matrix, and the two parts of the stiffness matrix are integrated using different Gauss rules: the dilatational part is integrated with one-point rule, while a $2 \times 2 \times 2$ rule is used on the deviatoric part. Reducing the number of constraints due to lack of compressibility in this way leads to much improved performance. The upper curve in Fig. 15.12 is obtained with this formulation (H8 with selective reduced integration, SRI).

If we apply different numbers of elements along the axis of the beam, which results in elements elongated along the axis of the beam to varying degrees, we realize that while the problem of dilatation locking seems to have been solved, the bending response of the H8 element deteriorates significantly with the aspect ratio (the more elongated the element gets, the worse the result). There are evidently other effects than just dilatational locking at play here. Indeed, the H8 element suffers also from the so-called shear locking.

[8]Folder: `FAESOR/examples/stress/3-D`

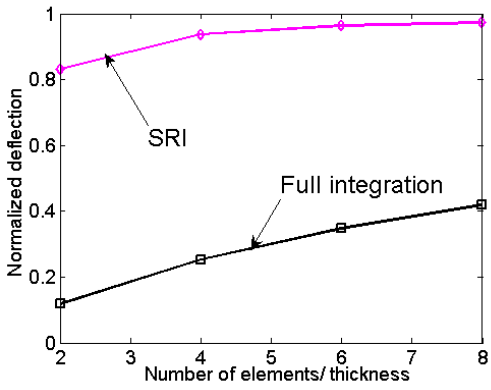


Fig. 15.12. Convergence of the tip deflection

15.5.3 Shear locking

The representation of bending by the brick element H8 (or the quadrilateral Q4 in two-dimensional models) is notoriously poor due to excessive shear stiffness being generated as a side effect of the element assuming a "bent" configuration: refer to Fig. 15.13 for an illustration. The result is known as ***shear locking***. It has to do with the partitioning of the deformation energy: consider a beam modeled with the brick element H8. When both the beam and the element are very stocky, the energies of shear and bending are comparable. On the other hand, when the beam and the element are very thin, the energy stored in bending should be much higher than that stored in shear. If the thin element needs to experience shear deformation to bend, the overall deformation will be severely limited. The reason is the big difference between the energy that should be stored in bending to attain a certain deflection (relatively small), and the spurious shear energy that is produced when the element is deformed to achieve that deflection (much larger). For a given amount of energy it then takes much smaller deflection to store it in shear than in bending, and locking results. We illustrate this behavior with an example next.

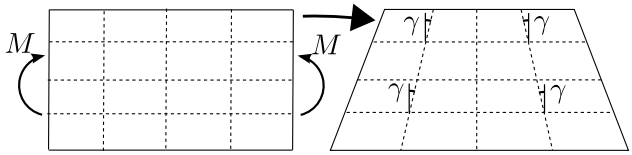


Fig. 15.13. Illustration of deformation that lead to shear locking

15.5.4 Thin clamped square plate with concentrated load

The setup is illustrated in Fig. 15.14, and the input data is fully specified in the Matlab script `clsqconc`[9]. This is one of the classic benchmarks for thin structures, see the Reference [TW-K59]. To match the thin-plate analytical solution is typically quite challenging for *plate* finite elements (that is structural elements specialized for bending deformations). Here we will attempt to compute this solution with *solid* elements. The essential boundary condition is trivial, but the concentrated force should give us a pause, as it is not an admissible load for 3-D elasticity. However, since the plate is thin, and since we are interested in the deflection of the structure as a "plate", committing this sin is not going to be fatal to the solution. Put differently, we are essentially pretending there is no singularity. Don't look at it: when we can't see it, it is not there.

[9]Folder: `FAESOR/examples/stress/3-D`

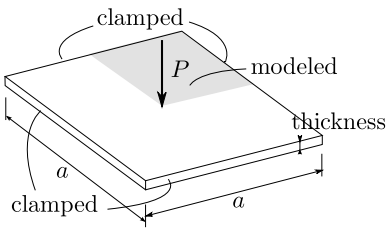


Fig. 15.14. Clamped square plate with center load

Due to the symmetry of the problem, we will discretize just one quarter of the geometry, and apply symmetry boundary conditions. For a very thin plate (thickness/span = 1/2000), the deflected shape is shown in Fig. 15.15. However, this shape cannot be computed with the H8 element. Even with many thousands of elements, the normalized computed deflection is just a fraction of a percent of the analytical solution. The H8 element locks very badly: As the plate is so thin, most of the energy should be in bending, but the element deformation mode puts most of it into shear and as a result the model is way too stiff. Even for thicker plates (up to the thickness/span ratio of 0.1, when the plate should be considered thick), the element H8 delivers only middling accuracy: see Fig. 15.16. The error is unacceptably large not only for extremely thin plates, but also for plates which are quite common in structures (thickness/span = 1/100). In comparison, the tetrahedron T10 is a solid performer, even for plates which are rather thin. The mesh shown in Fig. 15.15 gives decent results for the shown thickness to span ratio.

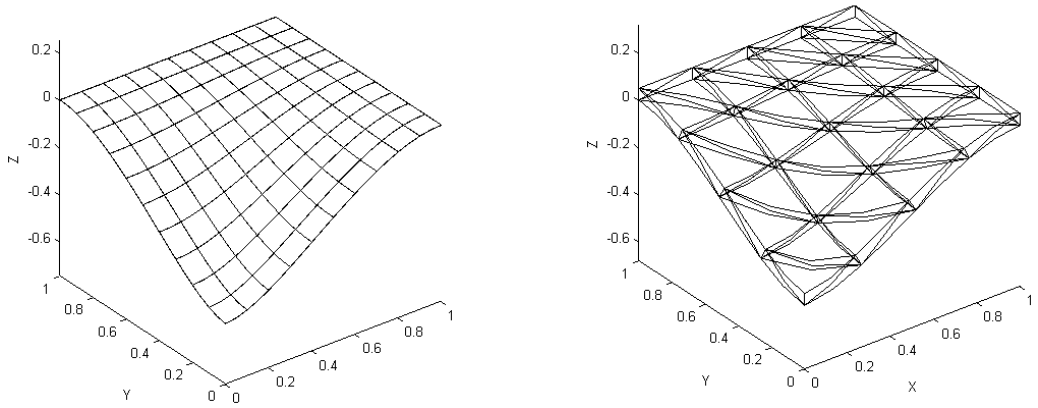


Fig. 15.15. Clamped square plate with center load. Very thin plate (thickness/span = 1/2000) and moderately thin (thick?) plate (thickness/span = 1/40); H20 and T10 meshes

15.5.5 Quadratic element H20

Similarly to the pair T4 and T10, the quadratic hexahedron H20 cures most of the stiffness in the joints of the H8. Contrary to (likely) expectations, the quadratic element is not going to be derived by taking the Cartesian product of the basis functions of the quadratic one-dimensional element L3 (Section 11.2). The result would have been an element with 27 nodes: $3 \times 3 \times 3 = 27$, the so-called quadratic Lagrangean hexahedron.

The quadratic hexahedron H20 is a member of the ***serendipity*** family. The nodes associated with basis functions are located at the corners and the midpoints of the edges: see Fig. 15.17; eight corners plus the midpoints of 12 edges gives 20 basis functions. The derivation of the basis functions may proceed as follows: start with the basis functions of the hexahedron H8, and add in the quadratic

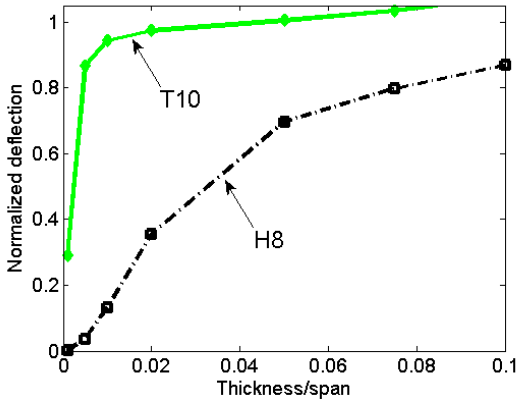


Fig. 15.16. Clamped square plate with center load. Normalized deflection under the load versus the thickness/span ratio, with a single element through the thickness and 20x20 elements in the plane. H8 and T10 meshes

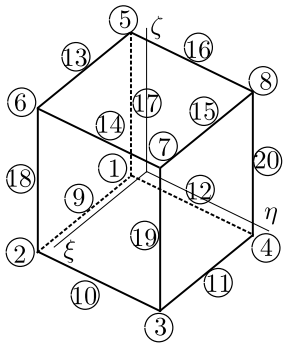


Fig. 15.17. Numbering of the nodes of the hexahedron H20

functions associated with the midpoints. Then modify the original linear functions so that all the basis functions add up to one at any point within the element.

The functions that are being added for the midpoints are produced as products of one quadratic function (in one variable) and two linear functions (in the remaining two variables). For instance, consider the basis function N_{11}: the variation of this function along the edge $3, 4$ should be the quadratic $(1 - \xi^2)$ (Fig. 15.18). Furthermore, N_{11} should vanish at all the nodes except 11. The function $(1 - \xi^2)$ is zero along the faces $2, 3, 7, 6$ and $4, 1, 5, 8$. To make it vanish also along the two faces $7, 8, 5, 6$ and $1, 2, 6, 5$, we multiply it with the two functions $(\eta + 1)$ and $(\zeta - 1)$. Finally, this product is normalized to assume value $+1$ at the node 11. The result is

$$N_{11} = \frac{(1 - \xi^2)(\eta + 1)(\zeta - 1)}{8} \, .$$

The basis functions for all the other mid-side nodes are obtained in the same fashion. The final step is to subtract half of each mid-side basis function from the two corner basis functions that are borrowed from H8 along that edge so that we recover the partition of unity property (8.24)

$$\sum_{k=1}^{20} N_k(\xi, \eta, \zeta) = 1 \, .$$

Thus, for instance for N_1 we have

$$N_1 = \frac{(\xi - 1)(\eta - 1)(\zeta - 1)}{8} - \frac{N_{11}}{2} - \frac{N_{10}}{2} - \frac{N_{19}}{2} \, .$$

Detailing all the other basis functions would take up too much space: see the class method `bfun`[10] of the class `gcellset_H20`.

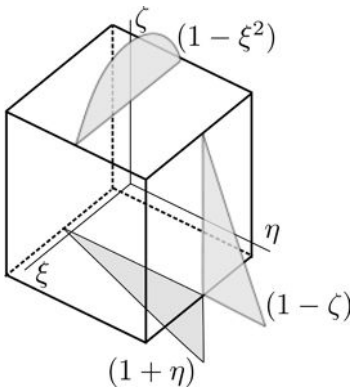


Fig. 15.18. Visualization of the three functions whose product gives the basis function N_{11} for the hexahedron H20

Gauss integration at $3\times3\times3$ points is considered a "full integration" of the stiffness (conductivity) matrix. However, the theoretically insufficient Gauss scheme $2 \times 2 \times 2$ points often leads to much improved results, giving a less constrained model with considerably improved flexibility.

The clamped plate problem is revisited with the 20 node hexahedron: Figure 15.19 shows the results for the normalized deflection under the load for the very thin plate. The hexahedron H20 is used with the two Gauss quadratures. While the convergence is similar, the reduced $2\times2\times2$ scheme has an edge. The mesh shown on the left in Fig. 15.15 is good for approximately 98% accuracy with the reduced integration, and a few percent worse for the full integration. The quadratic tetrahedron T10 does not manage to produce acceptable results for plate this thin.

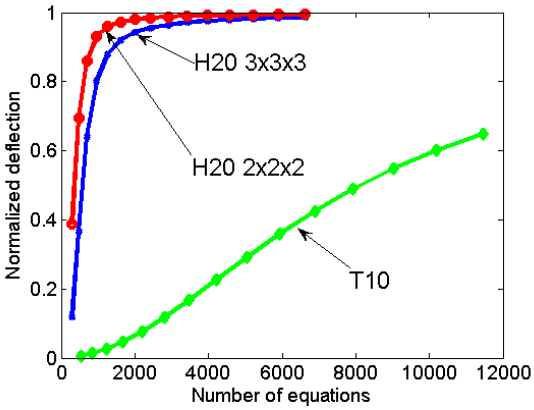


Fig. 15.19. Clamped square plate with center load. Normalized deflection under the load versus the number of unknowns for the thickness/span ratio= 1/2000, with a single element through the thickness and $4, 5, ..., 18$ elements along the in-plane directions

The aspect ratios of the elements obviously improve for thicker plates. Figure 15.20 demonstrates that accuracy improves, and Fig. 15.21 shows that, given the change of the vertical scale, the results for the moderately thick plate are quite satisfactory for all elements. However, the tetrahedron will always be more expensive than the hexahedron. For instance, for a given level of accuracy, more

[10]Folder: `FAESOR/classes/gcellset/@gcellset_H20`

degrees of freedom need to be included in the model; conversely, given a fixed budget of elements will lead to worse accuracy when using tetrahedra.

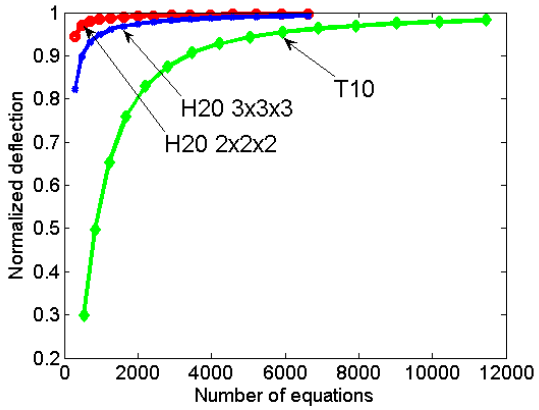


Fig. 15.20. Clamped square plate with center load. Normalized deflection under the load versus the number of unknowns for the thickness/span ratio= 1/200, with a single element through the thickness and $4, 5, ..., 18$ elements along the in-plane directions

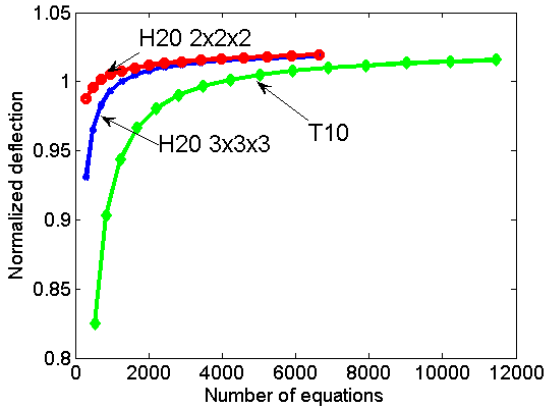


Fig. 15.21. Clamped square plate with center load. Normalized deflection under the load versus the number of unknowns for the thickness/span ratio= 1/40, with a single element through the thickness and $4, 5, ..., 18$ elements along the in-plane directions. The deflections are normalized by the analytical solution obtained for a thin plate, i. e. without consideration of shear deformations. That is why the numerical results apparently converge to a deflection higher than that predicted analytically.

15.5.6 Quadratic element Q8

The quadratic quadrilateral Q8 is compatible with the faces of the brick element H20 (compare Fig. 15.17 with Fig. 15.22). Therefore, to write down the basis functions for the quadrilateral, we may for instance substitute $\zeta = -1$ into the basis functions $N_j, j = 1, 2, 3, 4$ (corner functions), and $N_j, j = 9, 10, 11, 12$ (mid-side functions) of the hexahedron.

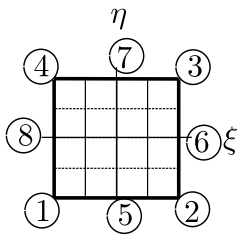


Fig. 15.22. Numbering of the nodes of the quadrilateral Q8

15.5.7 Pinched cylinder

Now we'll have a look at some shell problems. The first structure is defined in Fig. 15.23, and the input data is available in the Matlab script `pinchcyl`[11]. This is a widely used benchmark for shell structures. Due to symmetry, only 1/8 of the full structure is discretized, with only a single element through the thickness (the resolution in the circumferential and longitudinal direction is much more important).

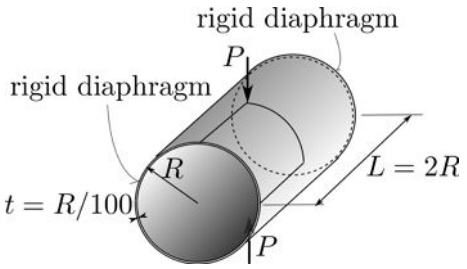


Fig. 15.23. Pinched cylinder: description of the problem

Of the two quadratic elements, H20 and T10, only the hexahedron performs well, and only for the reduced integration scheme $2 \times 2 \times 2$; full integration produces a model which is too stiff. The results are summarized in Fig. 15.24.

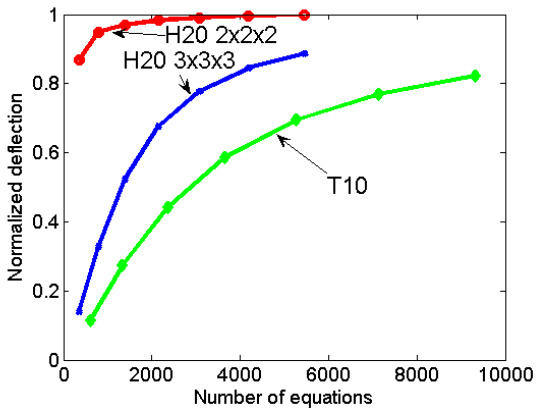


Fig. 15.24. Pinched cylinder. Normalized deflection under the load with elements H20 and T10

[11] Folder: `FAESOR/examples/stress/3-D`

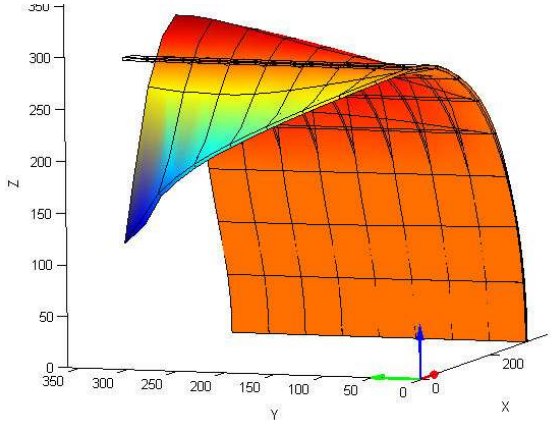


Fig. 15.25. Pinched cylinder. Shape under the load magnified 100,000 times

The shape of the structure under the load is displayed in Fig. 15.25 and it is possible to discern the interplay between membrane and bending action. Also note that the elements are curved in their reference form.

15.5.8 Pinched sphere

This benchmark problem is defined in Fig. 15.26: Note that the structure (boundary value problem) as defined is free-floating, loaded with self-equilibrated forces. The input data is fully described in the Matlab script `pinchsphere`[12]. This benchmark is used to measure the capability of finite elements to model inextensional bending in shell structures. Due to symmetry, only 1/4 of the full structure is discretized with the hexahedron H20, with only a single element through the thickness. The geometry of the spherical shell is captured in two ways: (i) "flat", where only the corner nodes have been forced to lie on co-spherical surfaces; and (ii) "fully curved", where the element nodes have all been moved so that all nodes are located on co-spherical surfaces.

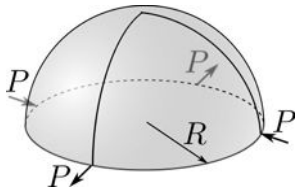


Fig. 15.26. Pinched sphere: description of the problem

The deformed shape of the structure is shown for the two element configurations in Fig. 15.27. Clearly, the fully curved elements afford better accuracy, which is not entirely surprising: when the elements are assembled into a faceted surface, the structure is effectively stiffened ("corrugated"). However, modeling curved surfaces by faceting is routinely done, since in the limit the faceting becomes less important. Indeed, we may observe in Fig. 15.28 that both configurations lead to satisfactory convergence, and that the curving of the elements is important only for coarse meshes.

15.5.9 Beam deflection revisited

Finally, we return to the problem of the (almost) incompressible beam. We study a few models with the goal of understanding their properties concerning both dilatational and shear locking (including

[12] Folder: `FAESOR/examples/stress/3-D`

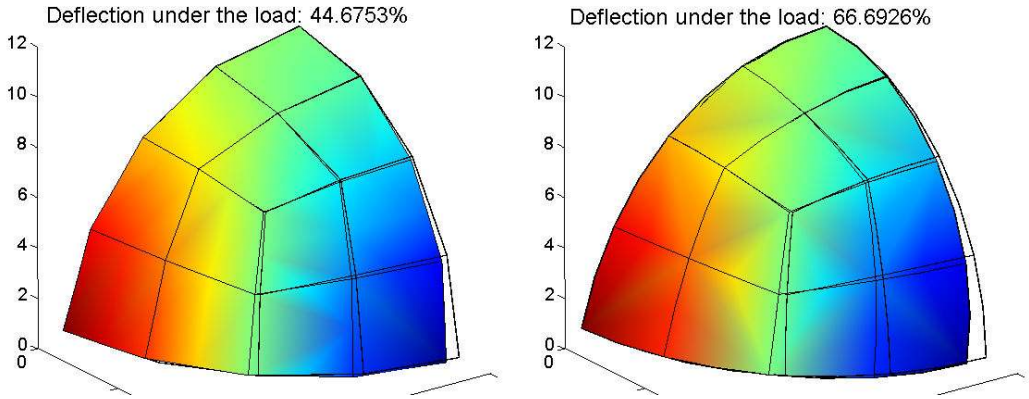


Fig. 15.27. Pinched sphere. Shape under the load magnified 10 times. Configuration of elements: flat (left), fully curved (right)

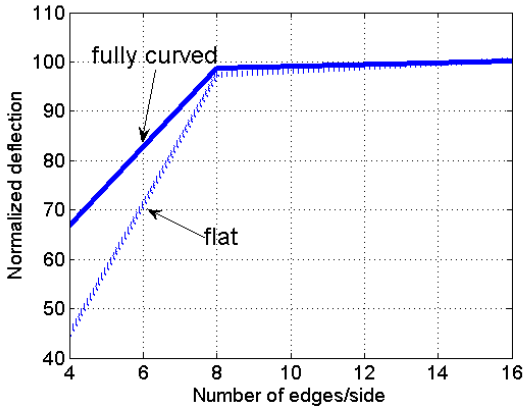


Fig. 15.28. Pinched sphere. Convergence of the deflection under the load for the two different configurations of the elements.

sensitivity to the height versus length ratio). (The input data is to be found in the Matlab script `rltb`[13].) The meshes have different aspect ratios, see Fig. 15.29, and correspondingly we may expect to be able to test the bending response: elements insensitive to aspect ratio are preferable to those that are sensitive. At the same time, we are able to ascertain robustness with respect to dilatation locking since the Poisson's ratio is close to 1/2.

The results are summarized in Fig. 15.29 (note that the vertical scale is different in each graph). The hexahedron H20 with the reduced quadrature $2 \times 2 \times 2$ points is a uniformly accurate element, apparently performing well with respect to dilatation locking, and insensitive to aspect ratio. Full integration for H20 produces an element slightly more sensitive to element elongation, and generally stiffer. The hexahedron H8 with selective reduced integration (SRI) deals well with dilatational locking, but is very sensitive to element elongation: its bending response is poor. The tetrahedron T10 is apparently well-behaved with respect to dilatation locking, but it is quite sensitive to the aspect ratio.

15.6 Errors, validation, and verification

Modeling physical events (for instance, the deflection of an airplane wing when the aircraft is turning is a physical event) on the computer may be described as shown in Fig. 15.30. In the first step,

[13] Folder: `FAESOR/examples/stress/3-D`

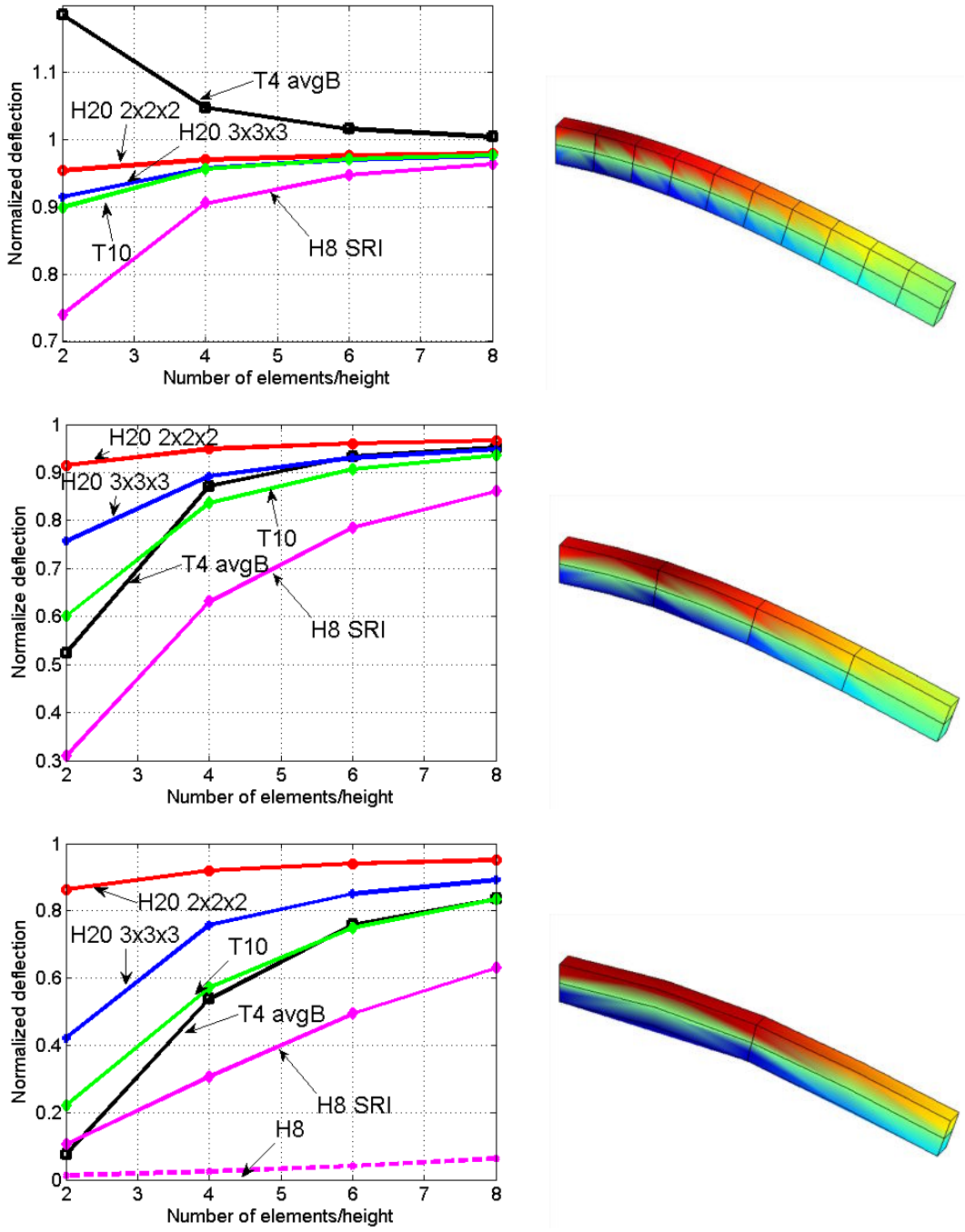


Fig. 15.29. Comparison of normalized deflections for the clamped beam with different aspect ratios. Top to bottom: 1:2, 1:5, 1:10 (height:length).

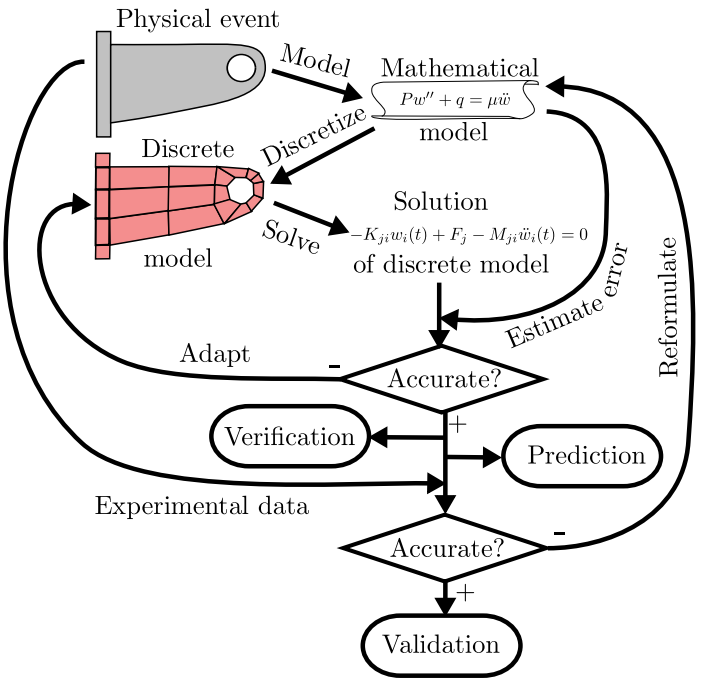


Fig. 15.30. Diagram of the modeling pipeline.

a ***physical event*** is idealized into a ***mathematical model***. The mathematical model has rarely exact (analytical) solutions, which gives rise to the need for a ***discrete model*** (the Galerkin finite element method in this book, but there are many others: finite difference and finite volume methods, boundary element methods, and so on). The unknowns in the discrete model are solved for using various numerical methods – solvers for systems of linear and nonlinear algebraic equations, eigenvalue solvers, integrators for systems of ordinary differential equations, ... – resulting in the ***solution*** of the discrete model. The accuracy of the solution to the discrete model may be then assessed with respect to the mathematical model with various methods of ***error estimation*** (Richardson extrapolation, for instance). If the accuracy is not sufficient, a better discrete model is produced by ***adaptation***, else the solution is deemed acceptable for either one of two actions: prediction or verification.

15.6.1 Verification and Prediction

There are two uses for the solution of the discrete model, depending on whether a reference (exact, or very accurate) solution of the mathematical model is available or not. If it is available, we obtain a ***verification*** of the way the discrete model is implemented in the computer: the ingredients that go into the discrete model are correct, and the equations are solved right, therefore we observe convergence towards (closeness to) the reference solution of the mathematical model. The verification should be addressed thoroughly by the designers of the computational software, but users also tend to perform verifications to gain confidence in the software. Physical events are typically very idealized to produce mathematical models that can be solved very accurately or analytically. Such mathematical models are called ***benchmarks***.

On the contrary, if the reference solution of the mathematical model is not known, the solution to the discrete model will make ***predictions*** of various quantities possible (deflections, natural frequencies, strains or stresses, energy, and so on).

15.6.2 Validation

Finally, if comparison of some quantities from the solution with ***observations*** of the corresponding quantities in the physical event is possible, and provided the agreement is good, we call the

modeling pipeline ***validated*** for this particular physical event; otherwise, if an improvement to the mathematical model may be made, for instance by including aspects of the physical event that have been deliberately neglected before, we perform yet another adaptation and another pass through the modeling pipeline. If there is a significant mismatch, we may decide that a completely different mathematical model needs to be formulated, perhaps even requiring a new theory. In this way, we get the chance to ***falsify*** a theory.

15.6.3 Errors

The contributions to the detected mismatch are associated with the arrows in the graph of Fig. 15.30: there's the ***modeling error*** that accompanies the idealization of the physical event into a mathematical model, the ***discretization error*** when converting the mathematical model into a discrete model, the ***solution error*** produced by the numerical algorithms, and there may also be an ***observation error*** due to inaccurate or erroneous measurements. Finally, an important source of mismatch may also be ***data uncertainty***, as all the input quantities will only be known with a certain margin of error (material parameters, geometric dimensions, and so on).

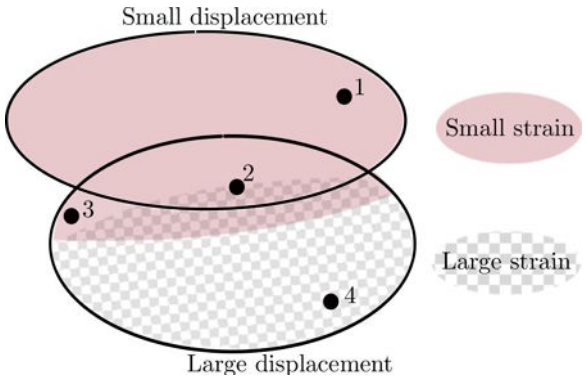


Fig. 15.31. Classes of stress analysis problems based on the magnitude of the displacement, and the magnitude of the strain. Examples: 1: concrete dam; 2: thin clamped plate under a transverse load; 3: steel measuring tape; 4: penetration of a steel slab by a high-velocity tungsten projectile.

15.6.4 Using modeling to make predictions

Comparing the solution of the discrete model with experimental observations allows us to call the solution validated with respect to the specific physical event. If we needed to observe the physical event each time we ran a simulation so that it could be validated, there would not be many incentives for using the simulation in the first place. However, in practice we use validations for a series of specific events to sample the "event space" to establish a ***range of validity***. If a particular physical event $\mathcal{E}$ resembles other events for which the validity of the model has been established, we assume that the model will be likely validated also for $\mathcal{E}$. As an example, consider the classification of the physical events with respect to the magnitude of displacements and strains (Fig. 15.31). If the physical event is the deformation of a structure that works roughly as a steel measuring tape, we assume that it is in the class of events for which models based on large displacements but small strains are validated. Note that the two model classes, small and large displacement, overlap. That is because there's no sharp division of the physical events that can be modeled with either depending on the required accuracy and the importance of capturing the effect of large displacements. For instance, deflections of clamped plates under transverse loads depend to a certain degree on the tension that develops in the structure when it deforms; if the contribution is negligible, small-displacement model could be adequate, otherwise a large-deflection model may be required.

15.6.5 Using benchmarks

The value of benchmarks for the verification of numerical models is clear, but there is another reason analysts should take advantage of benchmark problems: they need to build up a feel for the relative accuracy and robustness of finite elements, and such data is readily accessible in convergence studies performed on benchmark problems because of the availability of the reference solution. The convergence graphs displayed in the previous sections of this chapter are a source of valuable insights.

Eventually, analysts develop intuition to help them assess the suitability of a particular discretization for a particular physical event. For example, performing a series of convergence analyses for different values of the Poisson ratio will help us create a map of the performance of a particular element as it depends on this parameter. Figure 15.32 shows such a map for the fully-integrated brick element H8.

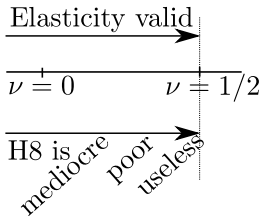


Fig. 15.32. The effect of the Poisson ratio on the accuracy of the fully-integrated brick element H8.

Figure 15.33 illustrates on the example of plate-like structures how their thickness would affect the choice of the mathematical model as well as the choice of the discretization. The 3-D elasticity theory on which the solid elements H20 and T10 are based is valid for all thickness to span ratios. We see that for relatively thicker structures the solid elements might be an effective discretization, while for really thin plates a specialized plate element (based either on the shear-free Kirchhoff theory for very thin plates, or based on the Mindlin theory that incorporates transverse shear for moderately thick plates) would be more a better choice. Finite element analysts construct maps to effective modeling strategies and techniques, such as the one given in Figure 15.33, by experience: running benchmarks, reading manuals and technical papers, and so on.

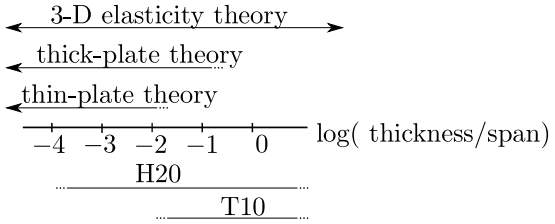


Fig. 15.33. Schematic classification of stress and deflection analysis finite elements for plate-like structures. Ranges for which the finite elements H20 and T10 could be effective.

Exercises

1. Chart 4.34 from Peterson's Stress Concentration Factors [P97] shows the stress intensity factors for an infinite thin slab (membrane) with an infinite row of circular holes loaded with biaxial tensile distributed forces. Consider $d = 10$mm, and thickness of the slab 2mm; assume that the slab is of AISI 1005 Steel.

a) Use as many planes of symmetry as possible and the St. Venant's principle to reduce the domain to finite size. Justify your choices, and illustrate the boundary conditions with a sketch. Hint: Apply the tractions in the direction perpendicular to the symmetry planes by prescribed displacements.
b) Compute the largest tensile stress using Richardson's extrapolation applied to a series of solutions with varying mesh sizes for $d/\ell = 0.1, 0.2, 0.4$. Plot the computed results in the chart 4.34 to compare with the analytical solutions.
c) Reduce the size of the domain so that it fits into a cube of side $\ell/2$, and recompute the largest tensile stress for $d/\ell = 0.4$. Evaluate the effect of the size of the domain on the computed tensile stress.

16

Analyzing the Stresses

Stresses are often of primary interest when designing structures. However, there are some inherent limitations to what is computable and how.

For example, when analyzed with the elasticity model some features in structures generate stresses that are infinite at some points. Obviously, there is little merit in attempts aimed at computing the stress at such points. Even away from these points, without special models the stress is often very inaccurate. Therefore, it behooves us to understand which features lead to infinite stresses so that we can formulate solution strategies appropriately.

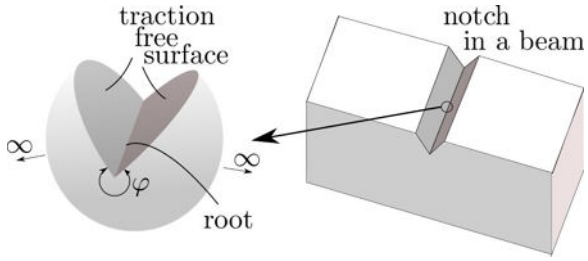


Fig. 16.1. Notch and the idealized geometry of an infinite wedge

16.1 Singularities

Some geometric features resemble notches, scratches, cracks, or put more generally, concave corners with a sharp root. Such geometries lead to states of stress which can be analyzed with two dimensional models (the displacements are functions of two variables). The idealization of such geometries is the ***wedge*** as a locus of points where two bounding surfaces meet at an angle. Figure 16.1 shows a geometry with a notch. Away from the locations where the notch root runs out into the side surfaces, the state of stress may be analyzed using the idealization shown on the left, the *wedge* [B99]. The material surrounding the edge of the root extends to infinity in all directions, but the solution is of interest only in the immediate vicinity of the edge. The analytical solution for symmetric and anti-symmetric loadings possesses for angles $\varphi > 180°$ a singularity in the stress components of the form

$$\sigma \sim r^{\alpha} \ , \tag{16.1}$$

where r is the radial distance from the edge, and $\alpha \leq 0$ measures the strength of the singularity. Figure 16.2 provides a sketch of the dependence of the exponent α on the angle φ. For symmetric loads, even a very slight notch will lead to singular stresses, and the strongest singularity occurs for

$\varphi = 360°$ (infinitely sharp crack). Similarly, anti-symmetric loading will produce the same strength of singularity for a crack configuration. In the same figure on the right, the graph shows that the stronger the singularity, the wider the general area around the wedge root where the stresses are high (they are infinite at the edge $r = 0$ for all strengths).

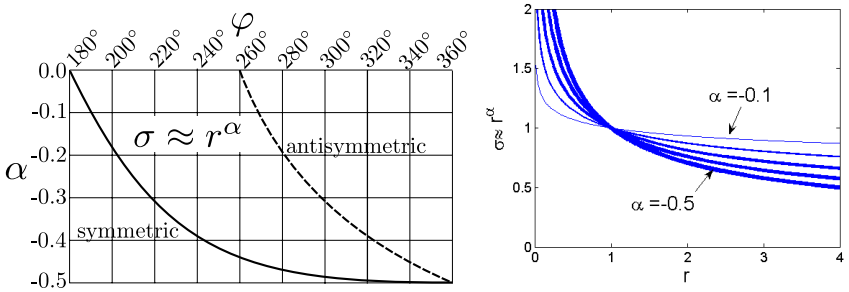


Fig. 16.2. On the left: Strength of the singularity in the stress. Symmetric (lower, solid, curve) and anti-symmetric (upper, dashed, curve) loading. On the right: Variation of the stress in the radial direction as it depends on the strength of the singularity. The stronger the singularity (i.e. the larger $|\alpha|$), the larger the volume of material that experiences high stress values

There are also other sources of singularities in the elasticity model. Singular stresses accompany edges along ***multi-material interfaces***, as shown in Fig. 16.3, or in Fig. 16.5. Singular stresses are also generated at sudden transitions from prescribed displacement to prescribed traction (***discontinuity in boundary conditions***), see Fig. 16.4 .

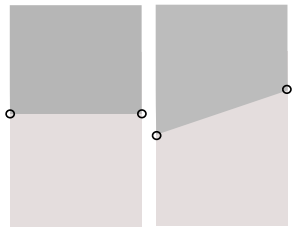


Fig. 16.3. Singularity at a multi-material interface: butt joint and scarf joint

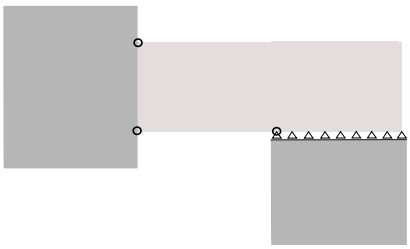


Fig. 16.4. Singularities due to sudden change in displacement boundary conditions. Corner between a clamped edge and the free surface. Location where the roller support ceases to apply.

We also pointed out in Section 13.6.4 that ***concentrated loads*** at points or along curves generate infinite stresses. As discussed in detail there, the energy in the model is infinite in the limit. Therefore, this kind of singularity is "even worse" than the singular stresses near the tip of a wedge. Also,

point supports and supports along curves leads to infinite stresses with similar characteristics as concentrated loads (unless the associated reactions are identically zero).

The most complex singularities to analyze are the true ***3-D singularities at corners***. Examples include (call for Fig. 16.5) the intersections of wedge fronts (crack fronts) with free surfaces (marked 1), corners of multi-material wedges (2), corners on multi-material interfaces (3), or corners in homogeneous geometries (4,6), conical points (marked 5), or corners in wedges (on crack fronts) (marked 7).

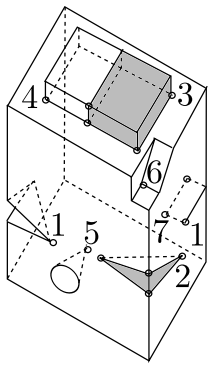


Fig. 16.5. 3-D singular points

16.2 Interpretation of stresses

In stress analysis the stress distribution is linked to the strains. The strains are defined using the spatial derivatives of the displacements. The displacements are continuous within each element (because the basis functions are continuous), and their derivatives are also continuous within each element (because the derivatives of the basis functions are continuous). For instance, across an element the strains may be constant (T4 tetrahedra), or vary linearly (T10 tetrahedra). At the interfaces between the elements the displacements are continuous (because along the finite element interfaces the basis functions guarantee continuity), but the strains are in general discontinuous.

It is important to realize that as a consequence of the strains being discontinuous across inter-element boundaries the finite element solution for the stresses is also going to be discontinuous across element interfaces. That is unphysical since we know that the traction vector (13.21) changes continuously in dependence on the normal to the surface on which it acts and the stress. Observe Figure 16.6: One material is on the bottom, another material is on the top. The point A is on the interface between the two materials. We can separate the two materials along this interface, and write the traction vector on the side of the bottom material as

$$\boldsymbol{t}(A_-) = \mathcal{P}_{\boldsymbol{n}(A_-)}\boldsymbol{\sigma}(A_-)$$

and the traction vector on the side of the top material

$$\boldsymbol{t}(A_+) = \mathcal{P}_{\boldsymbol{n}(A_+)}\boldsymbol{\sigma}(A_+)\ .$$

Since the two normals are related as $\boldsymbol{n}(A_-) = -\boldsymbol{n}(A_+)$, and since by principle of action and reaction we have $\boldsymbol{t}(A_-) = -\boldsymbol{t}(A_+)$, we must have

$$\boldsymbol{t}(A_-) = \mathcal{P}_{\boldsymbol{n}(A_-)}\boldsymbol{\sigma}(A_-) = -\boldsymbol{t}(A_+) = -\mathcal{P}_{\boldsymbol{n}(A_+)}\boldsymbol{\sigma}(A_+)$$

and therefore because

$$-\mathcal{P}_{\boldsymbol{n}(A_+)} = \mathcal{P}_{-\boldsymbol{n}(A_+)} = \mathcal{P}_{\boldsymbol{n}(A_-)}$$

we can conclude

$$\boldsymbol{\sigma}(A_-) = \boldsymbol{\sigma}(A_+) \ .$$

In words, the stress vector is continuous across the material interface. Let us recall that this holds for the mathematical model of elasticity, but the finite element quantities may behave differently!

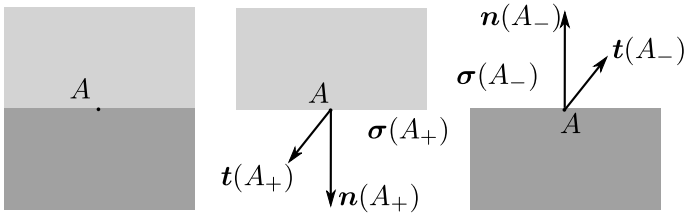


Fig. 16.6. Continuity of stress across a multi-material interface.

In what follows unless we say otherwise when we talk about displacements, strains, and stresses we mean the quantities computed by the finite element model. Here is what we know about the finite element stress: The stress is related to the strain through the constitutive equation. The constitutive equation is invoked only at the quadrature points. Therefore, the stress is consistent with the strains at the quadrature points. In the present model of elasticity, it may seem attractive to use the constitutive equation at any point within the element to compute the stresses. However, it needs to be realized that such stresses are an *interpretation* of the solution, not the solution itself. In other commonly used models in stress analysis, for instance in plasticity, computing the stresses from the displacements at other points than the quadrature points would be ill defined (and ill advised).

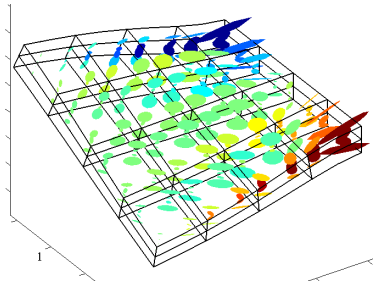


Fig. 16.7. Interpretations of the calculated stresses. Stress τ_{yz}. Stress ellipsoids at integration points

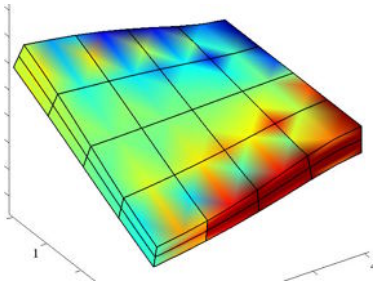


Fig. 16.8. Interpretations of the calculated stresses. Stress τ_{yz}. Stress computed at the nodes of each element from the displacement field

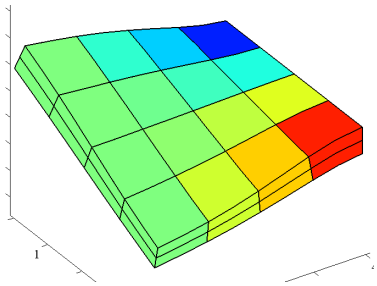


Fig. 16.9. Interpretations of the calculated stresses. Stress τ_{yz}. Stress averaged over all integration points in the element

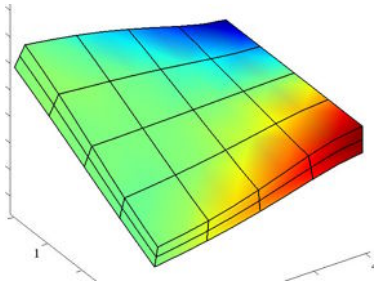


Fig. 16.10. Interpretations of the calculated stresses. Stress τ_{yz}. Stress extrapolated from the integration points to the nodes

Stress ellipsoid

As the base case consider Fig. 16.7: the stresses consistent with the finite elements solution are displayed at the $2 \times 2 \times 2$ quadrature points as the so-called Lamé stress ellipsoids that are visual representations of the principal stresses and directions of the principal stresses. The directions of the axes of the ellipsoid are the directions of the principle stresses, the radii of the ellipsoid are the principal stresses. Each ellipsoid is also colored to represent the magnitude of the stress component τ_{yz}. This is the honest representation of our knowledge of stress in the finite element model.

Stress from element displacements

In Fig. 16.8 we present an interpretation of the stress that is based on the displacements within each element, which are differentiated not at the quadrature points, but rather at the nodes. The strains derived in this way of the nodes are used to compute the stress that the node of each element. The magnitudes of the stress at the nodes are then interpolated across the elements as continuous color. The stress shown in Fig. 16.8 is not very smooth, which may make us suspicious. In fact the stresses and strains in the finite element solution are best behaved at the quadrature points, and using the displacements anywhere else will produce wildly oscillating and untrustworthy stresses. For this reason, the stress interpretation based on displacements at the nodes is normally not used in finite element analysis.

Average element stress

Figure 16.9 presents a relatively well behaved interpretation of the stress which is the average of the stresses at the quadrature points within each element. Therefore, each element gets a single color to represent the stress within it.

Recovered nodal stress

Stress recovery is a procedure in which values of the stress at the nodes are computed from the values of the stress in the elements around the node. Various principles are used to arrive at the nodal stress,

for instance simple average may be used, or a more sophisticated inverse-distance-weighted average, or the highly successful Superconvergent Patch Recovery (SPR) [ZT89]. In Figure 16.10 we show the interpretation of the stress which is the result of the inverse-distance-weighted average recovery: The magnitude of the stress at a node is computed by extrapolating from the magnitude of the stress at the quadrature points nearby. The nodal stress magnitudes are then interpolated using the element basis functions and smooth coloring is applied. The usefulness of such a smooth visual representation of the stress distribution for the evaluation of the results is evident. Nevertheless such a continuous stress field is an interpretation of the finite element results (often performed in a so-called *postprocessing* step). It should be realized that *extrapolating* the stress may be fraught with significant errors. In particular, the stresses are typically *smoothed* in these postprocessing procedures and spikes in stress may get wiped out. On the other hand, in general the recovered stress is an improvement of the stress representation, if for no other reason than because it is continuous across element interfaces.

Convergence in stress

Often one can use the plots of average element stress and the recovered nodal stress for quick visual assessment of the convergence of the stress. When the differences between the two plots become negligible we can interpret that as evidence of the stress being close to converged. For instance, consider Figure 16.11 where we compare the recovered nodal stress plot with the average element stress plot on the right. We can appreciate the indications of sufficient resolution being applied to the stress field as compared to the pair of Figures 16.9 and 16.10 where the blocky nature of the average element stress plot is evident in contrast to the smooth recovered nodal stress plot.

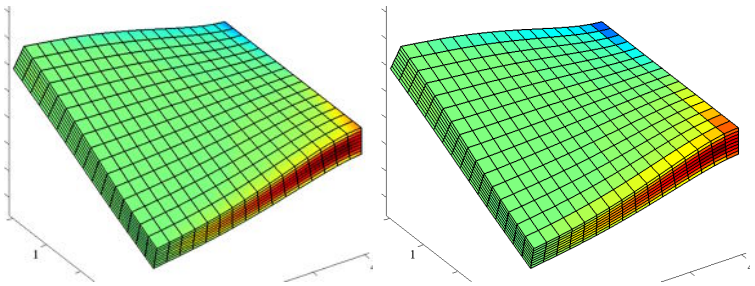


Fig. 16.11. Interpretations of the calculated stresses. Stress τ_{yz}. Left: recovered, continuous field; Right: average element stress.

16.3 Stress concentrations

The finite element formulations in this book may be applied to the solution of the so-called stress concentration problems. These are produced by stress raisers, which are features that cause local or global increases in the stress, but which do not lead to infinite stresses. The book [P97] is an extremely useful resource; a mode of operation where the solution of a finite element model is cross-checked with this book, or vice versa, is to be generally recommended.

To be aware of stress raisers is highly advisable for all modelers, since stress concentrations where the stress changes rapidly are generally locations of high error in the finite element solution: compare with Section 12.2.3. Therefore, appropriately constructed mesh will reflect the presence of stress raisers by suitable refinement (reduction in mesh size) around the stress concentrations. Figure 16.12 presents examples of stress raisers.

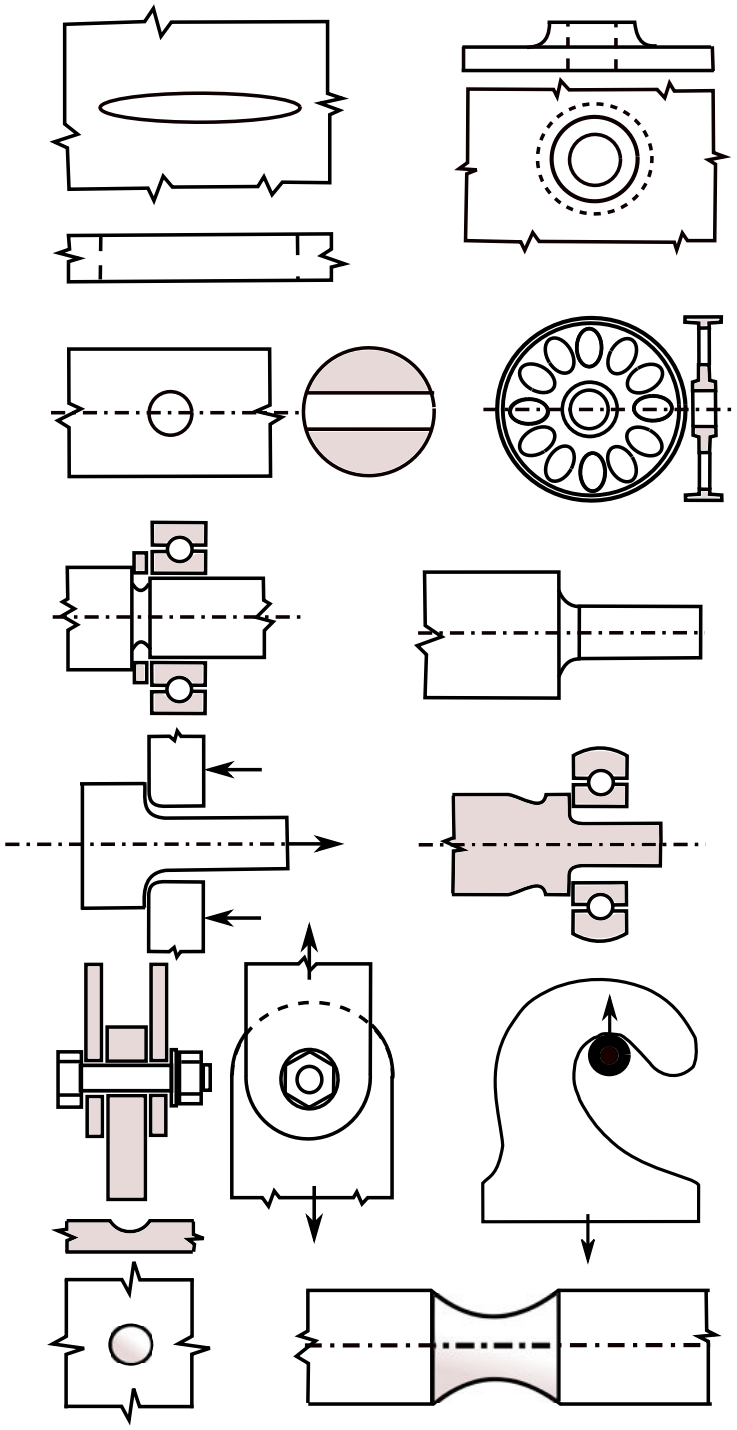


Fig. 16.12. Stress raisers. Left to right, top to bottom: Elliptical hole, stiffened hole, through-hole in a shaft, mass-reduction holes in a flywheel, snap-on ring groove, shoulder fillet, T-head suspension, stress relief groove, clevis and lug joint, curved hook, depression in a plate, cross-section reduction in a shaft

16.4 Adaptive refinement

The difference between the exact stress distribution $\sigma_{\text{ex},j}$ and the stress computed from the finite element solution σ_j can be measured using the so-called L_2 norm. In words, the difference between $\sigma_{\text{ex},j}$ and σ_j is squared and integrated over the domain. The contributions for all the components are summed, and square root is taken. The integration domain may be a single element e, in which case we measure the error of the stress over a single element:

$$\|E_{\sigma_{\text{ex}},e}\| = \left[\sum_j \int_{V_e} \left(\sigma_{\text{ex},j} - \sigma_j\right)^2 \mathrm{d}V\right]^{1/2}, \quad \text{where } j = xx, xy, yy \tag{16.2}$$

where the index j that indicates the stress components corresponds to plane strain or plane stress; the volume of element e is V_e.

When we wish to compute the error over the entire mesh, the contributions from the individual elements are added together (in other words, the integration is performed over all the elements) and we obtain

$$\|E_{\sigma_{\text{ex}}}\| = \left[\sum_j \int_{V} \left(\sigma_{\text{ex},j} - \sigma_j\right)^2 \mathrm{d}V\right]^{1/2} = \left[\sum_e \|E_{\sigma_{\text{ex}},e}\|^2\right]^{1/2} \quad \text{where } j = xx, xy, yy \tag{16.3}$$

The error norm gives us the ability to talk about the distribution of error within the computational mesh. Some elements contribute a large error $E_{\text{ex},e}$ whereas others contribute little. This has to do, as shown for instance in equation (12.11), with both the characteristics of the solution (where is the error large, where is it small?), and with the size of the elements. An element could contribute a large error because the element itself was very large and/or poorly shaped (γh large) and therefore it collected a large amount of error, and/or because even though the element was small the density of error within the element was very large ($C\left(\partial^2 T\right)$ large).

One possible strategy of constructing a mesh that is in some sense optimal is to make the elements of appropriate sizes so that they all contribute the same error. For a mesh with m finite elements that all contribute the same amount of error we write

$$\|E_{\sigma_{\text{ex}}}\| = \left[\sum_e \|E_{\sigma_{\text{ex}},e}\|^2\right]^{1/2} = \left[\sum_e \overline{\|E_{\sigma_{\text{ex}},e}\|}^2\right]^{1/2} = \left[m\overline{\|E_{\sigma_{\text{ex}},e}\|}^2\right]^{1/2} \tag{16.4}$$

where

$$\overline{\|E_{\sigma_{\text{ex}},e}\|}$$

is the desired error per element that each finite element contributes when the error is equally distributed across the mesh.

We may turn this around, set m to be the desired number of elements, and given the total error computed from a given finite element solution we can solve for the corresponding desired error per element $\overline{\|E_{\sigma_{\text{ex}},e}\|}$ from

$$\overline{\|E_{\sigma_{\text{ex}},e}\|} = \left[\frac{\|E_{\sigma_{\text{ex}}}\|^2}{m}\right]^{1/2} \tag{16.5}$$

The question then is how to use this information to guide the sizing of the elements? Let us say the element e with mesh size h_e contributes error which for the sake of the argument we can assume is larger than the desired error per element

$$\|E_{\sigma_{\text{ex}},e}\| > \overline{\|E_{\sigma_{\text{ex}},e}\|}$$

How large should an element constructed in the same location be so that it contributes only an error of $\overline{\|E_{\sigma_{\text{ex}},e}\|}$? We can reduce the error by constructing an element which is smaller then the currently present element $\widehat{h}_e < h_e$, but how to determine the new mesh size? We could invoke again the assumption of equation (12.12): the error is proportional to some power of the mesh size. We can write

$$\|E_{\sigma_{\text{ex}},e}\| \approx C h_e^{\beta} \,, \tag{16.6}$$

and also

$$\overline{\|E_{\sigma_{\text{ex}},e}\|} \approx C \widehat{h}_e^{\beta} \,. \tag{16.7}$$

Solving for the constant C from those equations allows us to combine them

$$\frac{\|E_{\sigma_{\text{ex}},e}\|}{h_e^{\beta}} = \frac{\overline{\|E_{\sigma_{\text{ex}},e}\|}}{\widehat{h}_e^{\beta}}$$

and finally to express

$$\widehat{h}_e = \left(\frac{\overline{\|E_{\sigma_{\text{ex}},e}\|}}{\|E_{\sigma_{\text{ex}},e}\|} \right)^{1/\beta} h_e \tag{16.8}$$

Remeshing

The process of designing a new mesh (remeshing) could be described as follows.

1. For a given finite element mesh and the solution for this mesh compute the total error from equation (16.3).
2. Choose the desired number of finite elements in the new mesh m, and compute the desired error per element (16.5).
3. For all elements e in the mesh compute the new mesh size of elements that should be constructed in the location of the current element e from (16.8).

To make this work we need to resolve two difficulties. One minor, and one major. The minor difficulty is that we have to estimate or guess the exponent β. The estimation could go as explained on the following example: Consider the linear triangle T3 used in stress analysis. It can represent exactly constant stresses. Therefore using the Taylor series and the order-of notation we can predict that the error of the stresses on the element e will be

$$(\sigma_{\text{ex},j} - \sigma_j) \in O(h_e)$$

where as above h_e is the size of the element e. Therefore, we get from (16.2)

$$\|E_{\sigma_{\text{ex}},e}\| = \left[\sum_j \int_{V_e} (\sigma_{\text{ex},j} - \sigma_j)^2 \, \mathrm{d}V \right]^{1/2} = \left[\sum_j \int_{V_e} (O(h_e))^2 \, \mathrm{d}V \right]^{1/2} =$$

$$\left[\sum_j \int_{V_e} O(h_e^2) \mathrm{d}V \right]^{1/2} = \left[O(h_e^2) O(h_e^2) \right]^{1/2} = O(h_e^2)$$

Consequently, for this case $\beta = 2$. Often this gets a little bit more complicated since the error of the stress components may not be $O(h_e)$ for domains with stress concentrations (especially with strong singularities, such as cracks or sharp concave corners). Then the error would converge slower, and we would take $1 < \beta \leq 2$.

The major difficulty we have to address is that starting with equation (16.2) onwards our discussion so far dealt with *true errors*. Of course, the true errors are in general unknowable. In other

words, $\sigma_{\text{ex},j}$ is unknown. Consequently we cannot use this framework as is. It is possible to avoid a wreck by thinking about *approximations* to the true stresses that are both reasonably accurate and computable. One approach that we are going to discuss is often called "stress recovery based", for the simple reason that instead of the true stresses the so-called recovered stresses are used (refer to Section 16.2).

In all the formulas above we would replace the true stress $\sigma_{\text{ex},j}$ (unknown) with the recovered stress σ_j^* (post-processed from the current finite element solution). We have warned in Section 16.2 against trusting the recovered stress too much for decisions based on the values of stress at any given location. In this case the recovered stress serves only as a guide to the design of the proper mesh to be used for the analysis. Only after an analysis had been performed on a mesh and the quality of the results needed to be assessed we would need to worry about the reliability of the recovered stress. For the present purpose to recovered stress is perfectly adequate.

Adaptive analysis

We could describe the resulting procedure as outlined here. First we create an initial reasonably designed mesh. It needn't be perfect, but it doesn't hurt to use one's experience as much as possible to make the initial mesh as effective as we can. We choose the number of elements that we can afford m, and the desired target error $\overline{\|E_{\sigma_e^*}\|}$. Then we repeat:

1. Compute the solution on the current mesh;
2. Post-process the stress;
3. Compute the approximate error distribution $\|E_{\sigma_e^*}\|$; If the error is acceptable, then stop and use the results; otherwise
4. Compute the desired mesh size $\widehat{h}_e$, and redesign the mesh. Repeat from step 1.

Of course, in order to redesign the mesh efficiently we need an automatic software tool (the so-called mesh generator) to do this. In `FAESOR` such a mesh generator is available for meshes composed of triangles. The above procedure is called ***adaptive analysis***. We call it "adaptive" because the solution that we use to make decisions is obtained from a discrete model that is adapted to the idiosyncrasies of the problem at hand. More precisely we could refer to $h-$adaptive analysis, because we use the mesh size to adapt the computation to the properties of the solution.

Example of adaptive heat conduction solution

In the above discussion we have been referring to stress analysis, and specifically to the stress error to guide the adaptive analysis. The stress is in fact a flux (of force), and therefore we could just as well be talking about the heat conduction model and the error of the heat flux. The process works exactly the same for both analyses.

For instance, here is the example discussed in Figure 12.11 in Section 12.2. Figure 16.13 presents in the top row the solutions for the temperature computed with three successive T3 triangle meshes. In the bottom row the color-coded distribution of error is presented. The goal of the adaptive remeshing in this case is to design a mesh that distributes the error equally to all the elements and consists of the target number of elements (650 in the present case). As we can see, the error is strongly concentrated at the re-entrant corner (as indicated by the dark color), and the adaptive procedure reflects this by constructing graded meshes with small elements near the reentrant corner. We can take the decrease of the largest element error, and increase of the smallest element error, as the evidence of the equilibration of the error on the mesh.

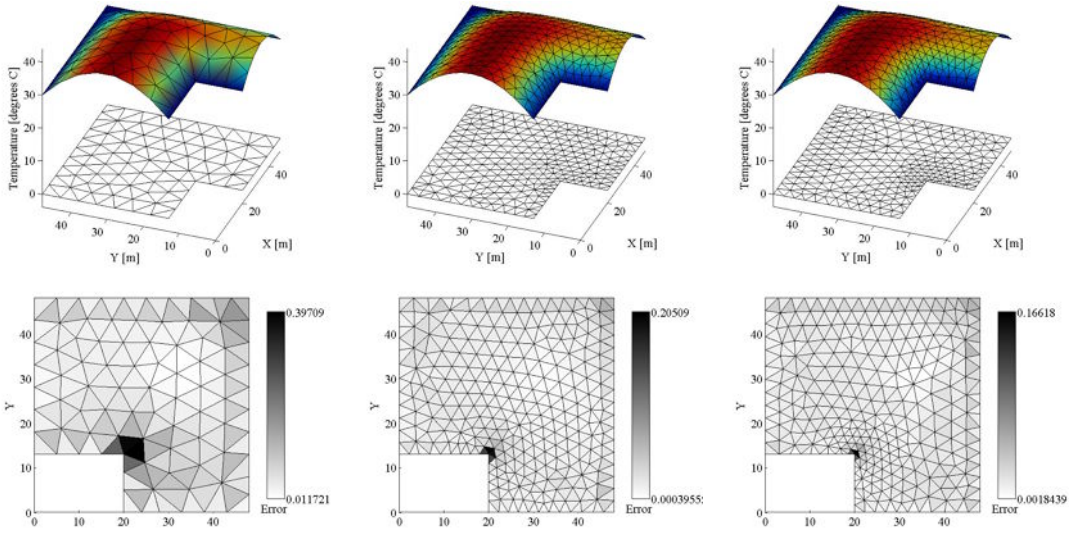


Fig. 16.13. L-shaped domain. Solution with adaptive refinement, target of 650 elements. Left to right: Initial solution, 138 elements, second solution, 470 elements, the third solution, 480 elements. Top row the temperature as a surface; the bottom row the mesh with color-coded distribution of error (dark color corresponds to large error). Matlab script `lshape3_t3_had`.

17

Plane Strain, Plane Stress, and Axisymmetric Models

In this chapter we will reduce the three-dimensional elastodynamic model to require solutions in terms of only two space variables (and the time). This is possible by introducing various assumptions for strains or stresses (and some restrictions on the form of the constitutive equation and loads).

17.1 Plane strain model reduction

The starting point is the observation that for some solids of uniform cross-section (right angle cylinders), such as the one shown in Fig. 17.1, the set of *assumptions* that **(i)** $u_z(x, y, z) = 0$, and $u_x = u_x(x, y)$, $u_y = u_y(x, y)$ seems to approximate the deformation well. As examples consider a long pipe under internal pressure (well removed from any valves or caps), or a straight concrete dam, or the midsection of a wall panel. The reduction to two space variables will be possible if the third

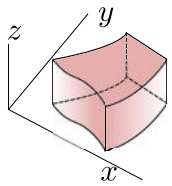


Fig. 17.1. Right-angle cylinder

balance equation is satisfied by design

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \overline{b}_z = \rho \ddot{u}_z \ .$$

The identically vanishing u_z also implies $\epsilon_z = 0$. Furthermore, we have for the shear strains

$$\gamma_{zx} = 0, \quad \gamma_{zy} = 0 \ .$$

Provided the constitutive equation allows, the conjugate shear stresses may also vanish. The condition for τ_{zx} is **(ii)**

$$\tau_{zx} = [\boldsymbol{D}]_{5,(1:6)}[\boldsymbol{\epsilon}] = 0 \ ,$$

where $[\boldsymbol{D}]_{5,(1:6)}$ is taken to mean columns 1 through 6 of row 5. This condition is satisfied provided the coefficients of the material stiffness that multiply the nonzero strains are zero,

$$D_{51} = D_{52} = D_{54} = 0 \ .$$

Symmetry also implies $D_{15} = D_{25} = D_{45} = 0$. Analogously, for the stress τ_{zy} the conditions on the material stiffness coefficients applies to row 6 (column 6). The situation is illustrated in Fig. 17.2: coefficients which could be nonzero are filled, coefficients which must be zero are marked as such, and coefficients which are probably best set to zero are blanks (those would couple τ_{xy} and σ_z).

The general *orthotropic* material of Section 14.5.2 complies with these conditions provided **(iii)** one of the orthotropy axes is aligned with the $z-$axis. A more general material model could be devised, but for practical purposes it is sufficient to consider the orthotropic material.

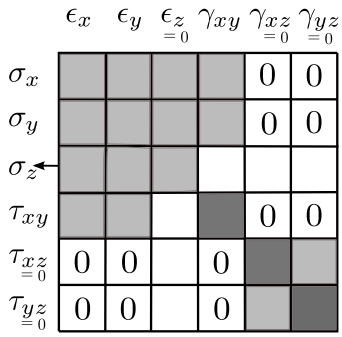


Fig. 17.2. Entries of the material stiffness matrix for plane strain

As a consequence of the assumptions on the displacements, and of the constraints on the form of the material stiffness, the third equilibrium equation becomes

$$\frac{\partial \sigma_z}{\partial z} + \overline{b}_z = 0 \, .$$

But since we already have $\epsilon_x = \epsilon_x(x, y)$ and $\epsilon_y = \epsilon_y(x, y)$, we can write

$$\sigma_z = [\boldsymbol{D}]_{3,(1:2)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \, ,$$

and we see that $\sigma_z = \sigma_z(x, y)$ provided **(iv)** all the coefficients $[\boldsymbol{D}]_{3,(1:2)}$ and $[\boldsymbol{D}]_{(1:2),3}$ are functions of x, y but not of z. Then, the equilibrium equation simply becomes **(v)** a definition of the admissible loads

$$\overline{b}_z = 0 \, .$$

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(v)** are

- top and bottom plane: $\overline{u}_z = 0$, $\overline{t}_x = 0$, $\overline{t}_y = 0$;
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the prescribed components depend on z.

The initial conditions satisfy the modeling assumptions and constraints **(i)-(v)** provided the initial displacements and velocities do not depend on z.

Since $\epsilon_z = 0$, the constitutive equation is written for the nonzero normal stresses and the shear stress as

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:2),(1:2)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{bmatrix} \, . \tag{17.1}$$

The stress-divergence operator for the plane strain model with just two balance equations and three stresses assumes the form

$$\mathcal{B}^T = \begin{bmatrix} \partial/\partial x & 0 & \partial/\partial y \\ 0 & \partial/\partial y & \partial/\partial x \end{bmatrix} . \tag{17.2}$$

The model for the fully three-dimensional elasticity resulted in the ODE system (14.32). In the present case, at this point we are still dealing with the same three-dimensional problem. However, when we substitute all the assumptions and constraints into the Eqs. (14.23), (14.25), (14.26), (14.27), (14.29), and (14.31), the following observations are made. Firstly, the values of all the degrees of freedom in the direction of the z-axis are known to be zero. Secondly, all the loads in this direction are also zero. Consequently, the third equilibrium equation may be ignored, and the test and trial function will have only two components, x and y. Thirdly, all the volume integrals may be precomputed in the z direction as all the integrands are independent of z. Thus, for instance the stiffness matrix integral (14.30) may be written as

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T \left(N_j(\boldsymbol{x})\right) \boldsymbol{D} \mathcal{B} \left(N_k(\boldsymbol{x})\right) \, \mathrm{d}V \right]_{im} = \tag{17.3}$$

$$\left[\int_S \mathcal{B}^T \left(N_j(\boldsymbol{x})\right) \boldsymbol{D} \mathcal{B} \left(N_k(\boldsymbol{x})\right) \, \Delta z \, \mathrm{d}S \right]_{im} . \tag{17.4}$$

Here Δz is the thickness of the slab in the z direction, S is the cross-sectional area of the cylinder. Only σ_x, σ_y, τ_{xy} (and correspondingly ϵ_x, ϵ_y, γ_{xy}) matter, and the strain-displacement matrix is the transpose of (17.2).

17.2 Plane stress model reduction

The plane stress model idealizes the following situation: imagine a thin slab of uniform thickness (the technical term would be ***membrane***) with both plane surfaces at the top and bottom (see Fig. 17.3) traction free, and the boundary conditions on the cylindrical surface independent of the z coordinate. Based on energy considerations we may presume that only the in-plane stresses would play a major role.

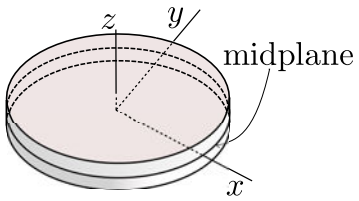


Fig. 17.3. Right-angle cylinder/thin slab/membrane

Therefore, we formulate a model starting from **(i)** the assumptions $u_x = u_x(x, y)$, $u_y = u_y(x, y)$, and u_z symmetric with a respect to the midplane of the membrane. To reduce the problem to two dimensions, we have to try to get rid of the third equilibrium equation.

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \overline{b}_z = \rho \ddot{u}_z \, .$$

Neither of the stresses in it will be exactly zero, since the corresponding strains are in general nonzero. Nevertheless, making an inference from the uniform, and small, thickness of the membrane, we adopt yet another assumption, **(ii)** $|\tau_{zx}| \ll 1$, $|\tau_{zy}| \ll 1$, and $|\sigma_z| \ll 1$. Next, **(iii)** the through-the-thickness body load component is assumed to be zero, $\overline{b}_z = 0$. Finally, in order for the third equilibrium equation to be truly negligible, **(iv)** we invoke the symmetry with respect to the midplane, and establish that the integral of this equation through the thickness of the membrane will vanish, i.e. the resultant perpendicular to the plane of the membrane will vanish

$$\int_{-h/2}^{h/2} \frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \overline{b}_z \, \mathrm{d}z = \int_{-h/2}^{h/2} \rho \ddot{u}_z \, \mathrm{d}z = 0 \, .$$

Furthermore, in order to comply with the symmetry requirement, we shall assume that **(v)** the material is orthotropic, with one orthotropy axis perpendicular to the midplane. The material stiffness matrix will therefore have the appearance shown in Fig. 17.4.

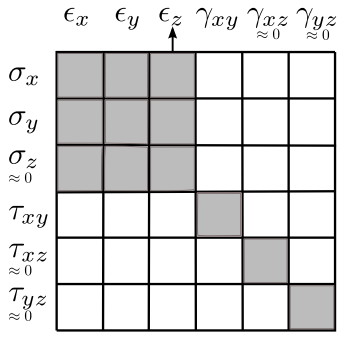


Fig. 17.4. Entries of the material stiffness matrix for plane stress

As the last step, the strain ϵ_z is eliminated from the constitutive equation. Assuming $\sigma_z \approx 0$, we have

$$\sigma_z = [\boldsymbol{D}]_{3,(1:3)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} \approx 0 \, ,$$

which gives

$$\epsilon_z = -D_{33}^{-1} [\boldsymbol{D}]_{3,(1:2)} \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \, .$$

Therefore, we obtain

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = \left[[\boldsymbol{D}]_{(1:2),(1:2)}, [\boldsymbol{D}]_{(1:2),3} \right] \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \end{bmatrix} = \left([\boldsymbol{D}]_{(1:2),(1:2)}, -D_{33}^{-1} [\boldsymbol{D}]_{(1:2),3} [\boldsymbol{D}]_{3,(1:2)} \right) \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \, , \tag{17.5}$$

which together with $\tau_{xy} = D_{44} \gamma_{xy}$ may be substituted into the first two equilibrium equations, rendering them functions of x, y only, provided **(vi)** the body load is $\overline{b}_x = \overline{b}_x(x, y)$, $\overline{b}_y = \overline{b}_y(x, y)$.

The boundary conditions consistent with the modeling assumptions and constraints **(i)-(vi)** are

- top and bottom plane traction-free ($\overline{t}_z = 0$, $\overline{t}_x = 0$, $\overline{t}_y = 0$);
- cylindrical surface: mixture of essential and natural boundary conditions, where none of the prescribed components depend on z.

The initial conditions satisfy the modeling assumptions and constraints **(i)-(vi)** provided the initial displacements and velocities do not depend on the z coordinate.

17.3 Model reduction for axial symmetry

The last model reduction approach to be discussed in this chapter, is the case of *torsionless* axial symmetry (Fig. 17.5). The main assumption is that **(i)** all planes passing through the y axis are

symmetry planes. (The y axis will be referred to as the axis of symmetry.) Therefore, points in any particular cross-section move only in the radial and axial direction, and do not leave the plane of the cross-section (the circumferential displacement is identically zero). Furthermore, points on circles in planes perpendicular to the axis of symmetry, with centers on the axis of symmetry, experience the same radial and axial displacements.

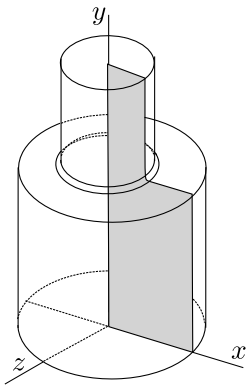


Fig. 17.5. Axially symmetric geometry

Let us consider one particular cross-section, for instance as indicated in Fig. 17.5. Since the displacements in the plane of the cross-section are symmetric with respect to the axis of symmetry, we will consider only the part to one side of the axis of symmetry (hatched in Fig. 17.5). The reduction from three equations of equilibrium to two equations in the plane of the cross-section, where x is the radial direction, and y is the axial direction, will succeed provided the equilibrium equation in the direction perpendicular to the plane of the cross-section (z) is satisfied. Thus, we consider

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{zy}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + \overline{b}_z = \rho \ddot{u}_z \ ,$$

where as the first simplification we note $u_z = 0 \rightarrow \ddot{u}_z = 0$. Furthermore, by symmetry we must have $\overline{b}_z = 0$.

Now for the stresses: As indicated in Fig. 17.6, by assumption the circles in planes perpendicular to the axis of symmetry are transformed by the deformation again into circles in planes perpendicular to the axis of symmetry. Therefore, right angles between the plane of the cross-section and the tangent to the circle where it intersects the cross-section will remain right angles, and the shear strains are $\gamma_{zx} = 0$, and $\gamma_{zy} = 0$. If the material is **(ii)** orthotropic, with one axis of orthotropy perpendicular to the cross-section (for instance a wound composite, with the reinforcing fibers running approximately in circles around the axis of symmetry), we may conclude that $\tau_{zx} = 0$, and $\tau_{zy} = 0$. Finally, since the material on a given circle experiences the same stress in all cross sections, $\partial \sigma_z / \partial z = 0$ (while in general $\sigma_z \neq 0$). Conclusion: the equation of motion in the z direction for each cross-section plane (and in particular the one in which the two-dimensional model is formulated) is satisfied exactly.

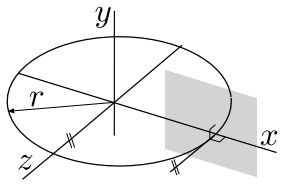


Fig. 17.6. Axially symmetric geometry: geometry of circles in planes perpendicular to the axis of symmetry

It remains to express the circumferential strain in terms of the displacements in the plane of the cross-section. By inspection of Fig. 17.6, displacement of the circle in the y direction does not change its circumference, while displacement radially means that the circle of radius x will experience strain

$$\epsilon_z = \frac{2\pi(x+u_x)-2\pi x}{2\pi x} = \frac{u_x}{x} \ .$$

The strain-displacement operator for this model may therefore be written as

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ 1/x & 0 \\ \partial/\partial y & \partial/\partial x \end{bmatrix} \ . \tag{17.6}$$

The stress divergence operator (the transpose of (17.6)) gives the reduced form for the equations of equilibrium (13.32)

$$\begin{aligned} \frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\sigma_z}{x} + \overline{b}_x &= \rho \ddot{u}_x \\ \frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \overline{b}_y &= \rho \ddot{u}_y \end{aligned}$$

where the presence of σ_z should be noted: recall that the equation of motion in the radial direction holds for a wedge-shaped element, hence the contribution of the circumferential stress to the radial direction must be included.

The constitutive equation is obtained from the full three-dimensional relationship by extracting appropriate rows and columns

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \end{bmatrix} = \begin{bmatrix} [\boldsymbol{D}]_{(1:3),(1:3)} & 0 \\ 0 & D_{44} \end{bmatrix} \begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z \\ \gamma_{xy} \end{bmatrix} \ . \tag{17.7}$$

17.4 Material stiffness for two-dimensional models

The material stiffness matrices are (17.1) for plane strain, equation (17.5) for plane E stress, and (17.7) for axial symmetry. While all are for models reduced to two displacement components, the stiffness matrices are clearly different. The material class `mater_defor_ss_linel_biax` provides the tangent moduli by transforming the material stiffness of a three-dimensional material to the form suitable for a particular reduced two-dimensional model. The class method `tangent_moduli` first retrieves the three-dimensional material stiffness from the property object (line `0014`), and then the switch statement implements the computation (or subsampling) expressed in the three formulas.

```
0013 function D = tangent_moduli[1](self, context)
0014     D= get(self.property,'D', context);
0015     switch self.reduction
0016         case 'axisymm'
0017             D =D(1:4, 1:4);
0018         case 'strain'
0019             D =D([1, 2, 4],[1, 2, 4]);
0020         case 'stress'
0021             Dt =D(1:2, 1:2)-D(1:2,3)*D(3,1:2)/D(3,3);
0022             D =D([1, 2, 4],[1, 2, 4]);
0023             D(1:2, 1:2)= Dt;
```

[1] Folder: `FAESOR/classes/mater/@mater_defor_ss_linel_biax`

```
0024         otherwise
0025             error([' Reduction '
                        self.reduction ' not recognized']);
0026     end
0027 end
```

Note well that there is no error checking concerning the assumptions in the three models. In particular, all three assume there is no coupling between normal stresses and the in-plane shear stress. If the three-dimensional material stiffness was retrieved from a property class which did not comply with these requirements, incorrect results would not be unexpected.

17.5 Strain-displacement matrices for two-dimensional models

The three models need different strain displacement matrices. The plane strain/ planes stress defines the stress-displacement matrix as the transpose of (17.2)

$$\mathcal{B} = \begin{bmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{bmatrix} . \tag{17.8}$$

while the axially symmetric model works with (17.6). Consider the discretization of a particular structure with the triangles T3. In general, it is not possible to tell whether the model is axisymmetric, plain strain, or plain stress just by looking at the mesh. The finite elements (geometric cells) need to be told which they are. For instance, an axisymmetric triangle is created by the constructor when a flag `'axisymm', true` is supplied:

```
gcells = gcellset_T3(struct('conn', [2, 4, 7],'axisymm', true));
```

Based on this flag, the two-manifold class `gcellset_2_manifold` decides which type of the private methods defined for the class `feblock_defor_ss` should be used to compute the strain-displacement matrix. For plane-strain and plane-stress models, the matrix is (17.8), and consequently `feblock_defor_ss_Blmat2` gets to be called. To optimize effort we include two cases in the code: the first where no global-to-local transformation is needed and therefore we avoid the multiplication with the transformation matrix `Rm` (it is supplied as empty), and the second where the matrix was given and therefore the multiplication needs to be performed.

```
0024 function B = feblock_defor_ss_Blmat2[2](self, N,Ndersp,c,Rm)
0025     nfn= size(Ndersp,1);
0026     dim =size(c,2);
0027     B = zeros(3,nfn*dim);
0028     if (isempty(Rm))% there is no global-to-local transformation
0029         for i= 1:nfn
0030             B(:,dim*(i-1)+1:dim*i)=...
0031                 [Ndersp(i,1) 0; ...
0032                 0           Ndersp(i,2); ...
0033                 Ndersp(i,2) Ndersp(i,1) ];
0034         end
0035     else% global-to-local transformation is requested
0036         for i= 1:nfn
0037             B(:,dim*(i-1)+1:dim*i)=...
0038                 [Ndersp(i,1) 0; ...
0039                 0           Ndersp(i,2); ...
0040                 Ndersp(i,2) Ndersp(i,1) ]*Rm(:,1:2)';
```

[2] Folder: `FAESOR/classes/feblock/@feblock_defor_ss/private`

```
0041          end
0042      end
0043      return;
0044 end
```

For the axisymmetric model the strain-displacement matrix is (17.6) and `feblock_defor_ss_Blmat2axisymm` is to be called. Note that the term which requires division with the distance from the axis becomes undefined where the distance is zero (in other words when the strain-displacement matrix is evaluated for a point on the axis of symmetry); in that case we arbitrarily set the distance equal to machine precision (a very small number) on line 0027.

```
0025 function B = feblock_defor_ss_Blmat2axisymm[3](self,N,Ndersp,c,Rm)
0026     nfn= size(Ndersp,1);
0027     r=c(1); if r==0,r=eps; end
0028     dim =size(c,2);
0029     B = zeros(4,nfn*dim);
0030     for i= 1:nfn
0031         B(:,dim*(i-1)+1:dim*i)=...
0032             [Ndersp(i,1) 0; ...
0033             0           Ndersp(i,2); ...
0034             N(i)/r 0; ...
0035             Ndersp(i,2) Ndersp(i,1) ]*Rm(:,1:2)';
0036     end
0037     return;
0038 end
```

17.6 Integration for two-dimensional models

As discussed earlier, the volume integrals for plane strain, such as the one required for the stiffness matrix, incorporate the third dimension as a "thickness". Integration through the third dimension reduces to multiplication with the thickness.

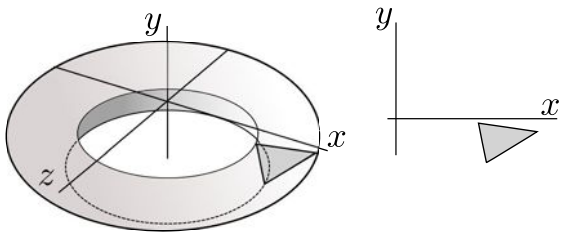


Fig. 17.7. Volume integration for axisymmetric analysis

The volume integrals for the axial-symmetry model reduction approach differ from plane stress or plane strain in that the "for free" integration is performed along the circumference of the rotationally symmetric body, not "through the thickness". For instance, the stiffness matrix integral (14.30) may be written as

$$K_{(j,i)(k,m)} = \left[\int_V \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,\mathrm{d}V\right]_{im} = \left[\int_S \mathcal{B}^T\left(N_j(\boldsymbol{x})\right)\boldsymbol{D}\mathcal{B}\left(N_k(\boldsymbol{x})\right)\,2\pi x\,\mathrm{d}S\right]_{im}, \tag{17.9}$$

[3]Folder: `FAESOR/classes/feblock/@feblock_defor_ss/private`

where $2\pi x\ \mathrm{d}S$ is the volume element of the ring generated by revolving the area $\mathrm{d}S$ around the axis of symmetry y—compare with Fig. 17.7. The volume integration may be performed by numerical quadrature over the area S but using a *volume* Jacobian

$$\int_V f\ \mathrm{d}V = \int_S f 2\pi x\ \mathrm{d}S \approx \sum_{k=1}^{M} f(\xi_k,\eta_k) J_{\mathrm{vol}}(\xi_k,\eta_k) W_k\ ,$$

where the volume Jacobian is defined as ($\det\left[J(\xi_k,\eta_k)\right]$ is the surface Jacobian)

$$J_{\mathrm{vol}}(\xi_k,\eta_k) = 2\pi x(\xi_k,\eta_k)\det\left[J(\xi_k,\eta_k)\right]\ . \tag{17.10}$$

In this way, all volume integrals are performed in exactly the same way, be they defined over a three dimensional manifold (solid), a two-dimensional manifold (surface equipped with a thickness, or swept around an axis of symmetry), a one dimensional manifold (curve equipped with a cross-sectional area), or a zero-dimensional manifold (point endowed with a chunk of volume). For the two-dimensional models of this chapter, the volume Jacobian is computed by the method `Jacobian_volume` defined for the class `gcellset_2_manifold`. The Matlab code closely mirrors the formulas; the method `other_dimension` evaluates the thickness at the given integration point `pc` (which in the plane-stress case could be variable).

```
0019 function Jac = Jacobian_volume[4](self, conn, N, J, x)
0020     if self.axisymm
0021         xyz =N'*x;
0022         Jac = Jacobian_surface(self, conn, N, J, x)*2*pi*xyz(1);
0023     else
0024         Jac = Jacobian_surface(self,conn,N,J,x)*other_dimension(self,conn,N,x);
0025     end
0026 end
```

Traction loads, such as prescribed pressure on part of the bounding surface, needs to be evaluated with surface integrals. Quite analogously to the volume integration, the surface integrals are numerically evaluated along curves, but the Jacobians are surface Jacobians. Compare with Figs. 17.3 and 17.8: for instance, for axial symmetry the surface Jacobian is

$$J_{\mathrm{surf}}(\xi_k) = 2\pi x(\xi_k)\det\left[J(\xi_k)\right]\ . \tag{17.11}$$

Here $\det\left[J(\xi_k)\right]$ is the curve Jacobian (8.72). For the two-dimensional models, the surface Jacobian is computed by the method `Jacobian_surface` defined for the class `gcellset_1_manifold`.

```
0019 function Jac = Jacobian_surface[5](self, conn, N, J, x)
0020     if self.axisymm
0021         xyz =N'*x;
0022         Jac = Jacobian_curve(self, conn, N, J, x)*2*pi*xyz(1);
0023     else
0024         Jac = Jacobian_curve(self,conn,N,J,x)*other_dimension(self,conn,N,x);
0025     end
0026 end
```

17.7 Thermal strains in two-dimensional models

The thermal strains must be of the form (14.34) as the two-dimensional models all require properly aligned orthotropic constitutive relations. In order to define the thermal strain loads (14.40),

[4] Folder: `FAESOR/classes/gcellset/@gcellset_2_manifold`
[5] Folder: `FAESOR/classes/gcellset/@gcellset_1_manifold`

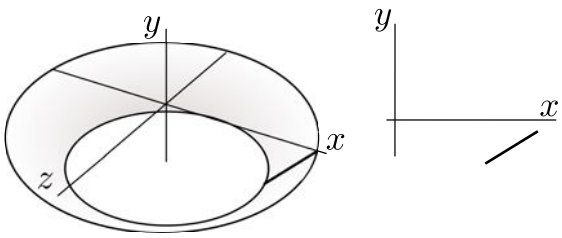


Fig. 17.8. Surface integration for axisymmetric analysis

three ingredients are needed: the strain-displacement matrix, the material stiffness, and the thermal strains. The former two have been discussed before, it remains to outline the calculation of the thermal strains when the model is reduced.

The plane stress model and the axially symmetric model may refer directly to (14.34), because the stress in the former is independent of the strain in the third direction, while all strains are incorporated in the latter. On the other hand, in the plane strain model the stress depends on the strain in the z direction (it must be zero), and nonzero thermal strain in the z direction will therefore have an effect which needs to be incorporated in the reduced constitutive equation.

The relationship between the normal stresses and the normal strains may be written for the plane strain model as a subset of the full three-dimensional relation

$$\begin{bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \end{bmatrix} = [\boldsymbol{D}]_{(1:3),(1:3)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \\ \epsilon_z = 0 \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \\ \epsilon_{\Theta z} \end{bmatrix} \right) \ . \tag{17.12}$$

Since we are not interested directly in σ_z, and since $\epsilon_z = 0$, we may write

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = [\boldsymbol{D}]_{(1:2),(1:2)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} \right) - [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} \ . \tag{17.13}$$

Grouping the strains with the same meaning leads to

$$\begin{bmatrix} \sigma_x \\ \sigma_y \end{bmatrix} = [\boldsymbol{D}]_{(1:2),(1:2)} \left(\begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} - \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} - [\boldsymbol{D}]^{-1}_{(1:2),(1:2)} [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} \right) \ , \tag{17.14}$$

where

$$- \begin{bmatrix} \epsilon_{\Theta x} \\ \epsilon_{\Theta y} \end{bmatrix} - [\boldsymbol{D}]^{-1}_{(1:2),(1:2)} [\boldsymbol{D}]_{(1:2),3} \epsilon_{\Theta z} \ ,$$

represents an effective in-plane thermal strain.

17.8 Examples

17.8.1 Thermal strains in a bimetallic assembly

Consider an assembly of two thin metal slabs. The inset is of aluminum, while the outer plate is of steel: see Fig. 17.9. The assembly is exposed to an increase of 70°C from the stress-free reference state. Of interest are the deformations produced by the unequal coefficients of thermal expansion. Due to the thinness of the plate, we consider plane stress an adequate approximation. Because of two-way symmetry, only a quarter of the plate is discretized.

The Matlab script `alusteel`[6] solves the problem with triangular adaptive meshes. The initial phases are omitted for brevity, we just mention that two sets of properties, materials, and finite element blocks need to be created because the assembly consists of two distinct materials.

[6]Folder: `FAESOR/examples/thermo_mechanical`

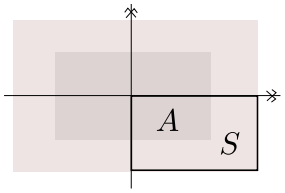


Fig. 17.9. Bimetallic assembly with thermal strain load

The calculation proceeds by assuming that the thermal strains are generated by a temperature distribution described by a field. Therefore, such a field is created, and all its degrees of freedom are set to the prescribed temperature.

```
0063 dT = field(struct ('name',['dT'], 'dim', 1, ...
0064     'data',zeros(length(fens),1)+70));
```

Both finite element blocks contribute to the stiffness matrix, and the assembly is performed on the concatenated arrays of element stiffness matrices produced for these two blocks. The thermal strain loads are again computed for each block separately, and assembled into the global load vector.

```
0066 K = start (sparse_sysmat, get(u, 'neqns'));
0067 K = assemble (K, cat(2,stiffness(febs, geom, u),...
           stiffness(feba, geom, u)));
0068 % Load
0069 F = start (sysvec, get(u, 'neqns'));
0070 F = assemble (F, cat(2,
         thermal_strain_loads(feba, geom, u, dT),...
0071     thermal_strain_loads(febs, geom, u, dT)));
```

It might be of interest to point out how to visualize a particular stress component. The first step is to create the graphic viewer, class `graphic_viewer`. The viewport is reset with the method `reset`.

```
0077 gv=graphic_viewer;
0078 gv=reset (gv,struct ('limits', [0, 100, -8, 40]));
0079 set(gca,'FontSize', 12)
0080 cmap=jet;
0081 cmpn=3;
```

Two fields are created by interpolating the stresses from the integration points to produce continuous representations by estimating stress at the finite element nodes. This is inherently tricky, since we know the stress is in general discontinuous at the nodes. The method `field_from_integration_points` of the class `feblock_defor_ss` uses inverse-distance interpolation. Two distinct fields are used, because the elastic properties of the two materials are different along their common interface.

```
0082 flda = field_from_integration_points(feba, geom, u, dT,
                'Cauchy', cmpn);
0083 flds = field_from_integration_points(febs, geom, u, dT,
                'Cauchy', cmpn);
0084 nvalsa=get(flda,'values');
0085 nvalss=get(flds,'values');
0086 nvalmin =min(min(nvalsa),min(nvalss));
0087 nvalmax =max(max(nvalsa),max(nvalss));
```

An object to map values of the stress to colors, `data_colormap`, is created and initialized with the range of the stress. The stress is then mapped to another field, `colorfield`, whose nodal parameters are colors. The computed color field is used to draw the geometric cells of the block `feba`, using the magnified (scaled) displacement field `u` so that the deformations are readily distinguishable. The same process is then repeated for the block `febs`.

```
0088 dcm=data_colormap(struct ('range',[nvalmin,nvalmax],
       'colormap',cmap));
0089 colorfield=field(struct ('name', ['colorfield'],
       'data',map_data(dcm, nvalsa)));
0090 draw(feba, gv, struct ('x',geom,'u', scale*u,
       'colorfield',colorfield, 'shrink',1));
0092 colorfield=field(struct ('name', ['colorfield'],
       'data',map_data(dcm, nvalss)));
0093 draw(febs, gv, struct ('x',geom,'u', scale*u,
       'colorfield',colorfield, 'shrink',1));
```

The problem is solved for four different meshes, each time scaling the element size down by a factor of 2. The shear stress is shown in Fig. 17.10. As can be seen, the stress peaks in the vicinity of the corner in the assembly where the two materials meet. We realize that the reentrant corner in the steel piece is going to generate a stress singularity. To compute the largest stress then becomes futile: it tends to infinity, and any subsequent refinement would only tend to cost more without any genuine improvements. However, in any real design the corner would not really be sharp, there would be a radius to take care precisely of such stress concentrations.

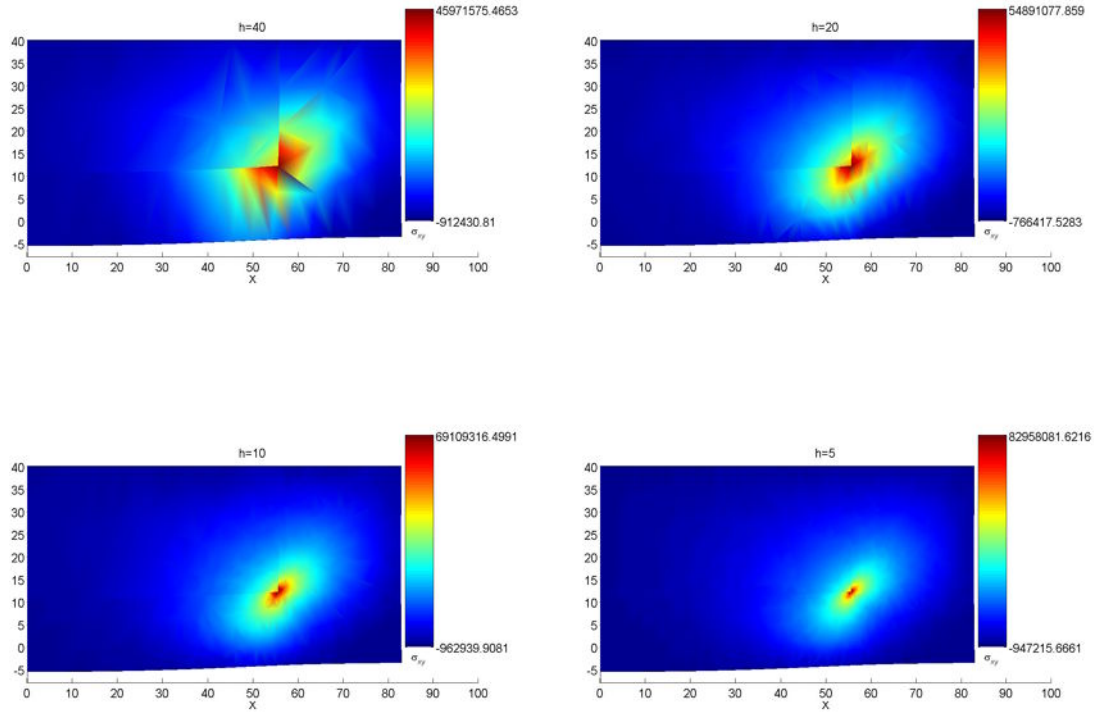


Fig. 17.10. Shear stress in the bimetallic assembly

The Matlab script `alusteelround` [7] solves for deformation and stress in a modified geometry, with the reentrant corner rounded to remove the stress singularity. The solution is displayed in Fig. 17.11, and upon closer inspection we can verify that the shear stress is now convergent.

[7] Folder: `FAESOR/examples/thermo_mechanical`

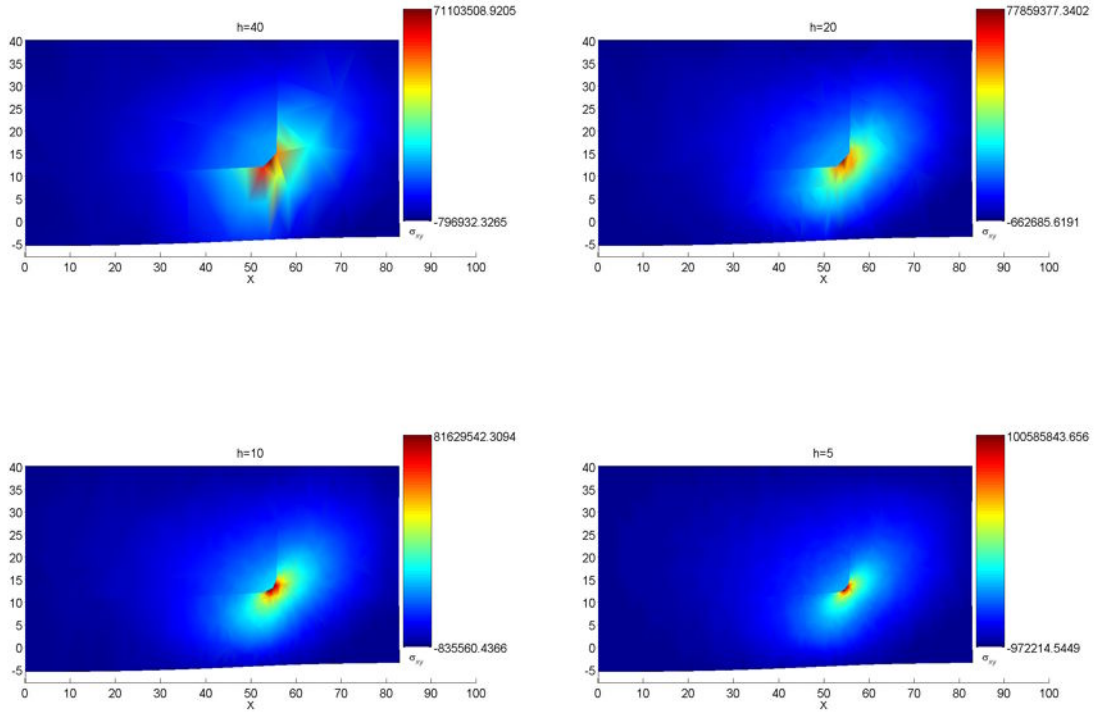


Fig. 17.11. Shear stress in the bimetallic assembly with rounded corner

17.8.2 Orthotropic balloon

In this example we consider an elastic balloon, with orthotropic properties induced by reinforcing fibers along circles in planes perpendicular to the axis of symmetry and centered on the axis of symmetry: see Fig. 17.12. The balloon is loaded by internal pressure. The model is based on the axial symmetry of the expected deformation. The modeled section is hatched, as we use not only the axial symmetry, but also symmetry with respect to the plane through the center of the ball perpendicular to the axis of symmetry.

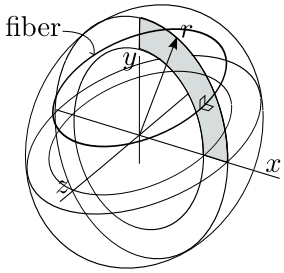


Fig. 17.12. Orthotropic balloon

The mesh is created in a rectangular domain, with the first dimension between zero and the thickness, and the second corresponding to the angle (from zero to the total angle swept out between

the axes x and y, which is $\pi/2$). Note that as an option we pass the flag `'axisymm'` to indicate the two-dimensional model reduction type.

```
0021 [fens,gcells]=block2d(rex-rin,pi/2,5,20,struct('axisymm',true));
```

In order to apply the pressure boundary condition on the inside of the balloon, we extract the boundary of the block, and we select the corresponding line segment cells.

```
0022 bdry_gcells = mesh_bdry(gcells, struct('axisymm', true));
0023 icl = gcell_select(fens, bdry_gcells, struct('box', [0,0,0,pi/2],
          'inflate',rin/100));
```

The preparation of the mesh is concluded by locating the nodes correctly in the x, y coordinates:

```
0024 xy=get (fens,'xyz');
0025 for i=1:length(fens)
0026     r=rin+xy(i,1); a=xy(i,2);
0027     xy(i,:)=[r*cos(a) r*sin(a)];
0028 end
0029 fens=set(fens,'xyz', xy);
```

The material model is based on the orthotropic property class, `property_linel_ortho`, but we choose the material constants so that in the x, y plane the material is isotropic, and the material is therefore effectively transversely isotropic. We must not forget to set the flag `axisymm` so that the material moduli matrix is computed correctly for the axially symmetric model.

```
0031 prop = property_linel_ortho(struct(
         'E1',E1,'E2',E2,'E3',E3,...
0032     'nu12',nu12,'nu13',nu13,'nu23',nu23,...
0033     'G12',G12,'G13',G13,'G23',G23));
0035 mater=mater_defor_ss_linel_biax (struct('property',prop,
0036     'reduction','axisymm'));
```

Two finite element blocks are created: `feb` collects the finite elements that represent a cross-section (quadrilaterals Q4), and `efeb` for the finite elements L2 along the pressure-loaded edge (the inside of the balloon).

```
0038 feb = febf (struct ('mater',mater, 'gcells',gcells,...
0039     'integration_rule',gauss_rule (2, integration_order)));
0040 efeb = febf (struct ('mater',mater, 'gcells',subset(bdry_gcells,icl),...
0041     'integration_rule',gauss_rule (1, 2)));
```

The essential boundary conditions are applied in the form of rollers on the axis of symmetry, and on the transverse plane of symmetry.

```
0048 % First the plane of symmetry
0049 ebc_fenids=fenode_select (fens,
        struct('box',[0 rex 0 0],'inflate',rex/10000));
0050 ebc_prescribed=ones(1,length (ebc_fenids));
0051 ebc_comp=ebc_prescribed*0+2;
0052 ebc_val=ebc_fenids*0;
0053 u = set_ebc(u,ebc_fenids,ebc_prescribed,ebc_comp,ebc_val);
0054 % The axis of symmetry
0055 ebc_fenids=fenode_select (fens,
        struct('box',[0 0 0 rex],'inflate',rex/10000));
0056 ebc_prescribed=ones(1,length (ebc_fenids));
0057 ebc_comp=ebc_prescribed*0+1;
0058 ebc_val=ebc_fenids*0;
0059 u = set_ebc(u,ebc_fenids,ebc_prescribed,ebc_comp,ebc_val);
0060 u   = apply_ebc (u);
```

The force intensity object of class `force_intensity` is given a function handle that corresponds to the internal pressure applied radially. The method `distrib_loads` integrates the force intensity over the block `efeb` of the edges on the internal surface, producing an array of element-load vectors. Note the 2 (last argument of `distrib_loads`): it indicates the manifold dimension to be used in the integration of the distributed loads; 2 for a surface.

```
0068 fi=force_intensity(struct('magn',@(x) (p*x'/norm(x))));
0069 F = start (sysvec, get(u, 'neqns'));
0070 F = assemble (F, distrib_loads(efeb, geom, u, fi, 2));
```

Figure 17.13 displays the deformed shape, which due to the reinforcing fibers running in planes perpendicular to the axis of symmetry assumes the general resemblance of a football. The circumferential stress is displayed on the deformed shape, while the undeformed shape is shown as shrunk outlines of the cross-section elements.

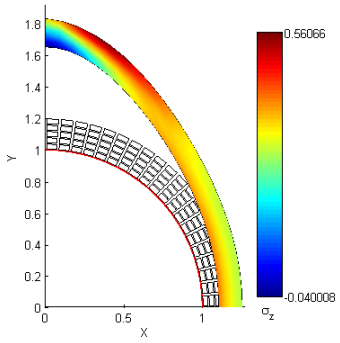


Fig. 17.13. Orthotropic balloon

17.9 Transient dynamic analysis

In this section we introduce the explicit variant of the Newmark algorithm, the ***centered difference*** integrator. It will be convenient to derive the algorithm as if we were thinking of the motion of a particle since to make it work for the finite element model, we just replace scalars with matrices.

17.9.1 Centered difference time stepping

The starting point is the well-known formula for the approximation of the second order derivative of the displacement

$$\ddot{u} \approx \frac{u_{n+1} - 2u_n + u_{n-1}}{\Delta t^2} = a_n \ , \tag{17.15}$$

where we introduce a label for the approximation, a_n, which is the so-called *algorithmic acceleration*; also, we use the time step $\Delta t = t_n - t_{n-1}$. Another way of computing this acceleration is from the equation of motion (Newton's equation)

$$a_n = M^{-1} f(t_n, u_n) = M^{-1} f_n \ , \tag{17.16}$$

where f_n is the resultant force. Since we assume u_n and u_{n-1} as the displacements of the particle at time instants t_n and t_{n-1} are known, we can use these two equations to solve for the next displacement, u_{n+1}. However, it is slightly inconvenient to have to keep track of the previous displacement, u_{n-1}, especially because it requires some fiddling in the first step (what is u_{-1}?).

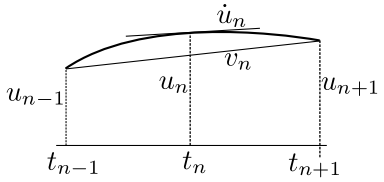


Fig. 17.14. Graphical representation of the centered difference definition of the velocity

A remedy is to use an intermediate variable, the *algorithmic velocity*. We define, again using centered differences (see Fig. 17.14),

$$v_n = \frac{u_{n+1} - u_{n-1}}{2\Delta t} \ . \tag{17.17}$$

Therefore, we can express

$$u_{n-1} = -2\Delta t v_n + u_{n+1} \ , \tag{17.18}$$

and substituting into (17.15), write the displacement

$$u_{n+1} = u_n + \Delta t v_n + \frac{\Delta t^2}{2} a_n \ . \tag{17.19}$$

It remains to determine how to push forward the algorithmic velocity. Combining the two Eqs. (17.18) and (17.19), written for the time instants t_{n+2}, t_{n+1} and t_n

$$v_{n+1} = \frac{u_{n+2} - u_n}{2\Delta t} \ , \quad u_{n+2} = u_{n+1} + \Delta t v_{n+1} + \frac{\Delta t^2}{2} a_{n+1} \ , \tag{17.20}$$

allows us to express u_{n+2} in two ways, and obtain therefore an equation for v_{n+1}

$$v_{n+1} = v_n + \frac{\Delta t}{2}(a_n + a_{n+1}) \ . \tag{17.21}$$

To summarize, the ***centered difference algorithm*** now reads

$$\text{Given:} \quad u_n, v_n, a_n = M^{-1} f(t_n, u_n) \tag{17.22}$$

Compute:

$$\begin{aligned} u_{n+1} &= u_n + \Delta t v_n + \frac{\Delta t^2}{2} a_n \\ a_{n+1} &= M^{-1} f(t_{n+1}, u_{n+1}) \\ v_{n+1} &= v_n + \frac{\Delta t}{2}(a_n + a_{n+1}) \end{aligned}$$

As shown for instance in Reference [H00], this algorithm is only conditionally stable, meaning that it blows up for time steps longer than a ***critical time step***

$$\Delta t_{\text{crit}} = \frac{2}{\max \omega_h} = \frac{\min T_h}{\pi} \ , \tag{17.23}$$

where max ω_h is the highest frequency of vibration in the discrete system, and min T_h is the associated period. Cruder estimates of Δt_{crit} are available also from the vibration frequencies of the individual elements in the mesh, or as the smallest time interval needed to travel the distance between finite element nodes by an elastic wave.

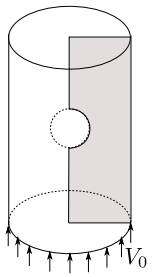


Fig. 17.15. Cylindrical rod with the spherical cavity

17.9.2 Example: stress wave propagation

In this section, we will use the centered difference algorithm to obtain a transient solution for the problem of a wave propagating through a cylindrical rod in which there is a spherical cavity. We would expect to see reflections of the stress wave off the free surface of the cavity. Refer to Fig. 17.15. The Matlab script `pressrecho`[8] solves the problem as axially symmetric, using linear triangles. The issue of convergence is not being addressed, but in any serious investigation it certainly should be: as we are well aware by now, the linear triangles are not very accurate.

The mesh is created in the axially symmetric domain with the automatic mesh generator. Note that an options structure is being passed to make all the generated triangles axisymmetric: `struct('axisymm', true)`.

```
0014 [fens,gcells,groups,edge_gcells,edge_groups]=
                targe2_mesher({...
0015     ...
0025     }, 1.0,struct('axisymm', true));
```

The property, material, and finite element block, geometry field and displacement field objects are created in the usual way. Displacement boundary condition is applied only along the axis of symmetry.

Next, the initial velocity `v` field is created by copying the displacement field; since it is a *copy* of the displacement field, the numbers of equations are now available in the velocity field. These are used later to retrieve the system vector initialized in the proper places with the initial velocity. In this example, the initial velocity is imparted to the nodes on the bottom cross-section to simulate for instance an impact.

```
0050 v = u;% copy everything from displacement field to velocity
0051 fenids=fenode_select (fens,struct('box',[0 15 -25 -25],'inflate',1/100));
0052 for j=1:length(fenids)
0053     v = scatter(v,fenids(j),[0, 1]*vmag);
0054 end
```

The stiffness and mass matrices are assembled: note that the mass is assembled as a diagonal matrix. That affords the best efficiency, since all the solutions of the type `A0 =Mmat\(-Kmat*U0)` are then very quick indeed. The diagonalization is effected by lumping the element mass matrices using the row-sum technique (line `0063`).

```
0056 K = start (sparse_sysmat, get(u, 'neqns'));
0057 K = assemble (K, stiffness(feb, geom, u));
0058 M = start (sparse_sysmat, get(u, 'neqns'));
0059 ems=mass(feb, geom, u);
0060 mat=get(ems,'mat');
0061 for j=1:length(mat)
```

[8] Folder: `FAESOR/stress/2-D`

```
0062     mat{j}=diag(sum(mat{j}));
0063 end
0064 M = assemble (M, set(ems,'mat',mat));
```

The initial values for the integration are established next.

```
0067 Kmat = get(K,'mat');
0068 Mmat = get(M,'mat');
0069 U0 = gather_sysvec(u);
0070 V0= gather_sysvec(v);
0071 A0 =Mmat\(-Kmat*U0);
```

The critical time step is calculated by computing the largest natural frequency of the discrete system. This might not be a good idea for very large meshes, since the eigenvalue solution takes time.

```
0073 o2=eigs(Kmat,Mmat,1,'LM');
0074 dt= 0.99* 2/sqrt(o2)
```

The loop really transcribes the algorithm (17.22) more or less literally. A minor addition is the re-initialization after each step (lines 0087-0089).

```
0082 while t <tfinal
0083     t=t+dt;
0084     U1 = U0 +dt*V0+(dt^2)/2*A0;
0085     A1 = Mmat\(-Kmat*U1);
0086     V1 = V0 +dt/2* (A0+A1);
0087     U0 = U1;
0088     V0 = V1;
0089     A0 = A1;
... omitted graphics
0106 end
```

Figure 17.16 illustrates the pressure at equally spaced time intervals. Given the fairly limited resolution (approximately 30 elements radially), the results are far from smooth, but still afford a qualitative picture of a pressure echo.

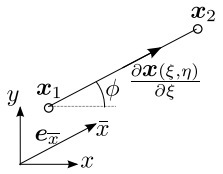
17.10 Solved exercises

Exercise 78.

Derive the global-to-local strain-displacement matrix for truss structural member derived as an L2 finite element. Specialize the expression to two model dimensions (planar truss).

Solution: We define the local coordinate system at the quadrature point using a unit vector tangent to the rod centerline

$$\boldsymbol{e}_{\overline{x}} = \frac{\partial \boldsymbol{x}}{\partial \xi} \Big/ \left\| \frac{\partial \boldsymbol{x}}{\partial \xi} \right\|$$



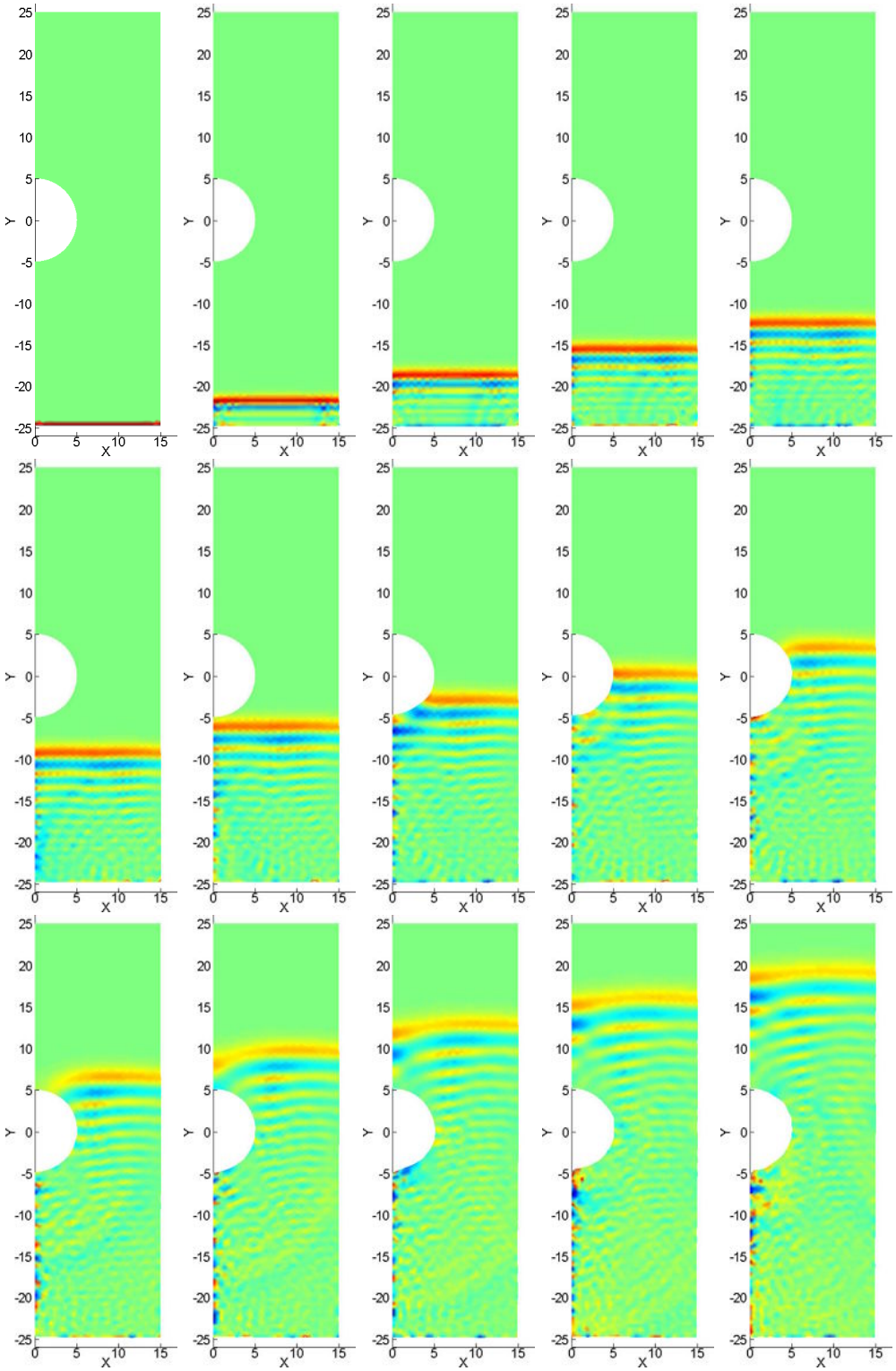


Fig. 17.16. Propagation of a stress wave through an axially symmetric bar with a spherical cavity. Displayed is the pressure at equally spaced time intervals

The axial strain is expressed in the local coordinate system as

$$\overline{\epsilon} = \frac{\partial \overline{u}}{\partial \overline{x}}$$

The derivative with respect to the local coordinate is expressed using the usual chain rule

$$\frac{\partial \spadesuit}{\partial \overline{x}} = \frac{\partial \spadesuit}{\partial \xi}\frac{\partial \xi}{\partial \overline{x}} = \frac{\partial \spadesuit}{\partial \xi}\left(\frac{\partial \overline{x}}{\partial \xi}\right)^{-1} = \frac{\partial \spadesuit}{\partial \xi}\left(\overline{J}\right)^{-1}$$

Here $\overline{J} = \frac{\partial \overline{x}}{\partial \xi}$ is the Jacobian matrix in the local coordinate system. The local coordinate $\overline{x}$ is obtained by projection onto the direction of $\boldsymbol{e}_{\overline{x}}$

$$\overline{x} = \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{x}$$

Therefore the local Jacobian $\overline{J}$ is obtained as

$$\overline{J} = \frac{\partial \overline{x}}{\partial \xi} = \boldsymbol{e}_{\overline{x}} \cdot \frac{\partial \boldsymbol{x}}{\partial \xi}$$

where the column vector $J = \frac{\partial \boldsymbol{x}}{\partial \xi}$ is the Jacobian matrix in the global coordinate system. The local displacement is also obtained by projection

$$\overline{u} = \boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}$$

Now we introduce the finite element interpolation

$$\boldsymbol{u} = N_1(\xi)\boldsymbol{u}_1 + N_2(\xi)\boldsymbol{u}_2$$

which also implies

$$\overline{u} = N_1(\xi)\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}_1 + N_2(\xi)\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}_2$$

and therefore we have for the local-coordinate strain

$$\overline{\epsilon} = \frac{\partial \overline{u}}{\partial \overline{x}} = \frac{\partial \overline{u}}{\partial \overline{x}} = \frac{\partial N_1}{\partial \overline{x}}\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}_1 + \frac{\partial N_2}{\partial \overline{x}}\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}_2$$

In matrix form we replace the dot product with $\boldsymbol{e}_{\overline{x}} \cdot \boldsymbol{u}_1 = [\boldsymbol{e}_{\overline{x}}]^T[\boldsymbol{u}_1]$ and analogously for the second nodal displacement to obtain

$$\overline{\epsilon} = \frac{\partial \overline{u}}{\partial \overline{x}} = \frac{\partial \overline{u}}{\partial \overline{x}} = \frac{\partial N_1}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T[\boldsymbol{u}_1] + \frac{\partial N_2}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T[\boldsymbol{u}_2]$$

Therefore we can rearrange the relationship between the global displacements and the local strain as

$$\overline{\epsilon} = \left[\, \tfrac{\partial N_1}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T \,,\, \tfrac{\partial N_2}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T \,\right] \begin{bmatrix} [\boldsymbol{u}_1] \\ [\boldsymbol{u}_2] \end{bmatrix}$$

which introduces the global-to-local strain-displacement matrix as

$$\boldsymbol{B}^e = \left[\, \tfrac{\partial N_1}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T \,,\, \tfrac{\partial N_2}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T \,\right]$$

Note that each nodal strain-displacement submatrix $\boldsymbol{B}^e_k = \frac{\partial N_k}{\partial \overline{x}}[\boldsymbol{e}_{\overline{x}}]^T$ complies with the general formula (14.56)

$$\boldsymbol{B}^e_k = \mathcal{B}^{(\overline{x},x)}\left(N_k(\boldsymbol{x})\right) = \mathcal{B}^{(\overline{x})}\left(N_k(\boldsymbol{x})\right)\left[\boldsymbol{R}_m\right]^T$$

Specializing the strain-displacement matrix to two model coordinates we have

$$[\boldsymbol{e}_{\overline{x}}] = \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}$$

which gives with the derivatives of the basis functions with the respect to the local coordinate (L^e is the length of the truss element)

$$\frac{\partial N_1}{\partial \overline{x}} = -1/L^e, \quad \frac{\partial N_2}{\partial \overline{x}} = 1/L^e$$

finally

$$\boldsymbol{B}^e = \frac{1}{L^e}\left[-\cos\phi, -\sin\phi, \cos\phi, \sin\phi\right]$$

Exercise 79.

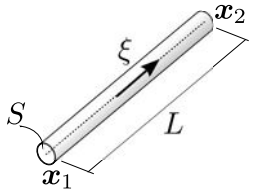
Derive the stiffness matrix for truss structural member of uniform cross-sectional area and homogenous material derived as an L2 finite element. Specialize the expression to two model dimensions (planar truss).

Solution: In exercise 78 we have derived the global-to-local strain-displacement matrix as

$$\boldsymbol{B}^e = \frac{1}{L}\left[-\cos\phi, -\sin\phi, \cos\phi, \sin\phi\right]$$

The stiffness matrix of one truss member will be the result of the integral (14.48)

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^{eT}\boldsymbol{D}\boldsymbol{B}^e \, \mathrm{d}V \ ,$$



The L2 element has one parametric coordinate, and the volume integral is therefore evaluated as

$$\boldsymbol{K}_e = \int_0^L \boldsymbol{B}^{eT}\boldsymbol{D}\boldsymbol{B}^e S(\overline{x}) \, \mathrm{d}\overline{x} \ ,$$

where $S(\overline{x})$ is the cross-sectional area at $\overline{x}$. The constitutive relation between the axial stress and the axial strain is expressed in the local coordinate system as

$$\overline{\sigma} = E\overline{\epsilon}$$

where E is the Young's modulus, so the (constant) material stiffness matrix is

$$\boldsymbol{D} = [E]$$

The numerical quadrature is carried out in the parametric domain $-1 \leq \xi \leq +1$

$$\boldsymbol{K}_e = \int_{-1}^{+1} \boldsymbol{B}^{eT}\boldsymbol{D}\boldsymbol{B}^e S(\xi) \det[J] \, \mathrm{d}\xi \ ,$$

which considerably simplifies since $\boldsymbol{D}, \boldsymbol{B}^e$ are independent of the integration variable:

$$\boldsymbol{K}_e = \boldsymbol{B}^{eT}\boldsymbol{D}\boldsymbol{B}^e \int_{-1}^{+1} S(\xi)\det[J]\,\mathrm{d}\xi\ ,$$

The expression $S(\xi)\det[J]$ is the volume Jacobian for the L2 finite element. The curve Jacobian $\det[J]$ is expressed as

$$\det[J] = \left|\left| \frac{\partial \boldsymbol{p}(\xi)}{\partial \xi} \right|\right|$$

as discussed in Section 8.13. As also worked out in the same section, the curve Jacobian specializes for the L2 element to $\det[J] = L/2$. Furthermore the cross-section is uniform and the integral simplifies to

$$\int_{-1}^{+1} S(\xi)\det[J]\,\mathrm{d}\xi\ , = S\frac{L}{2}2 = SL$$

where SL is the volume of the truss member. So finally we arrive at the stiffness matrix of the trust member as

$$\boldsymbol{K}_e = \frac{ES}{L}\begin{bmatrix} -\cos\phi \\ -\sin\phi \\ \cos\phi \\ \sin\phi \end{bmatrix}\begin{bmatrix} -\cos\phi, -\sin\phi, \cos\phi, \sin\phi \end{bmatrix}$$

which is identical to the well-known expression from structural analysis.

Exercises

1. How would damping forces be included in the centered difference time stepping? Consider the equation of motion $\boldsymbol{M}\ddot{\boldsymbol{u}} + \boldsymbol{C}\dot{\boldsymbol{u}} + \boldsymbol{K}\boldsymbol{u} = \boldsymbol{L}$, where the damping forces are velocity-proportional (viscous).

18

Consistency + Stability = Convergence

Let us assume the elasticity problem we wish to solve has a nice and smooth "exact" solution. What are the conditions for a finite element model to converge to this exact solution in the limit of the mesh size approaching zero? Even though we are never going to be able to reach this limit except in special situations, we need to know that it is possible to approach it.

For the simple finite element models that are considered in this book, there are two sufficient conditions for convergence to occur: *consistency* and *stability*.

18.1 Consistency

Consistency is really just a front for two individual requirements: Completeness + Compatibility. For the sake of the argument the following discussion will be centered around the model of elasticity (statics), but the conclusions may be extended to other problems more or less readily.

18.1.1 Completeness

A key quantity in the elasticity model are the strains. The completeness requirement aims to ensure that all possible constant strains within an element are represented exactly. (When we say all possible constant strains, we also mean zero strains – rigid body motion.) Let us build on our intuition:

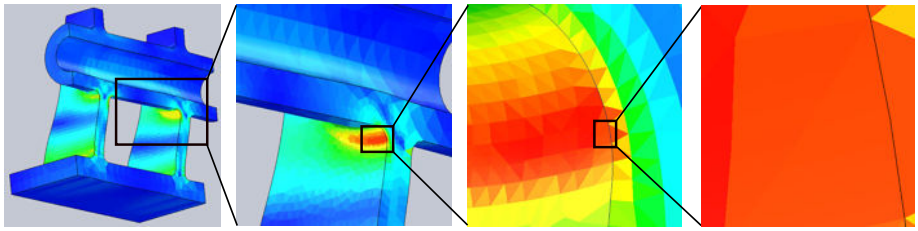


Fig. 18.1. Zooming in on the strain with a series of progressively stronger magnifying glasses

imagine being able to observe the strains on a deforming structure as a continuous field (Fig. 18.1). When we observe the whole structure, the distribution of the strains is in general very complex, but as we look at smaller and smaller areas on the structure, with progressively stronger and stronger magnifying glasses, the variations in the strains become less and less evident. This is the same principle as in the truncated Taylor series: the smaller the excursion from a given point, the more important the lower-order terms, and the less important the higher-order terms. Looking at the point itself, only the *constant* term remains. In other words, looking close enough we see material experiencing ***constant strain***. *If this material was enclosed in a finite element, we would want the finite element to be able to represent this state.*

Since the strains are obtained as combinations of the first derivatives of the displacements, we conclude that for the element to be able to represent constant strain, it must be able to reproduce displacements that are *linear polynomials exactly*. Clearly, the simplex elements (L2, T3, T4) satisfy this requirement by construction. However, it is easy to verify that *all* the iso-parametric elements discussed in this book reproduce exactly linear functions: Assume a linear function in this form

$$f(\boldsymbol{x}) = \boldsymbol{a}\boldsymbol{x} + b \ ,$$

where $\boldsymbol{a}$ is a constant row matrix, and b a constant. Now write a finite element interpolation of the function f as

$$\Pi_h f(\boldsymbol{x}) = \sum_{k=1}^{M} N_k(\boldsymbol{x}) f_k \ .$$

Provided the degrees of freedom are set as

$$f_k = \boldsymbol{a}\boldsymbol{x}_k + b \ ,$$

where $\boldsymbol{x}_k$ are the position vectors of the nodes, we have

$$\Pi_h f(\boldsymbol{x}) = \sum_{k=1}^{M} N_k(\boldsymbol{x}) f_k = \sum_{k=1}^{M} N_k(\boldsymbol{x}) \left[\boldsymbol{a}\boldsymbol{x}_k + b\right] \ .$$

But due to the partition of unity property (8.24) and to the interpolation of the Cartesian coordinates (8.25) we have

$$\Pi_h f(\boldsymbol{x}) = \sum_{k=1}^{M} N_k(\boldsymbol{x}) \left[\boldsymbol{a}\boldsymbol{x}_k + b\right] =$$

$$\boldsymbol{a} \sum_{k=1}^{M} N_k(\boldsymbol{x}) \boldsymbol{x}_k + b \sum_{k=1}^{M} N_k(\boldsymbol{x}) = \boldsymbol{a}\boldsymbol{x} + b = f(\boldsymbol{x}) \ .$$

Therefore, all the finite elements that satisfy (8.24) and (8.25) are complete in the sense discussed in this paragraph.

18.1.2 Compatibility

The elasticity mathematical model describes a material which does not separate or fracture. Therefore, also the finite element model must preserve the continuity of the material. The basis functions within an element describe how particles within the confines of the element move. Since all the basis functions we discussed were continuous inside the element, the compatibility requirement is satisfied inside the element. However, we also need to insure that the material does not experience overlaps or fissures where the individual elements meet. Therefore, we formulate the compatibility requirement along the edges (faces or vertices): all elements meeting along an edge (along a face, or at a vertex) of the mesh must describe the displacements in the same way. Thus, mixing together three-node and six-node triangles could result in gaps opening in the material (compare with Fig. 18.2), because along an edge the displacement is a linear function from one side, and a quadratic function from the other. And if such gaps proliferated in the mesh, there would be a portion of energy improperly accounted for, since these gaps do not exist in the mathematical model of the structure.

18.2 Stability

Stability determines the uniqueness of the finite element solution. The mathematical model of elasticity does not admit solutions for the displacement that do not produce positive energy other than

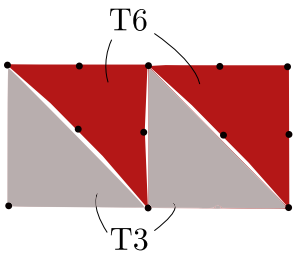


Fig. 18.2. Incompatible arrangement of elements. T3 triangles with linear variation of displacement along an edge cannot be matched with T6 triangles with quadratic variation of displacement along an edge. Uncontrollable gaps would open up.

rigid body motion. (Provided the material is stable: the material stiffness matrix must be positive definite.) Therefore, the finite element model must not have built-in ***zero-energy modes*** other than rigid-body modes. However, in order for most linear algebra algorithms and solvers to work, we would remove all rigid body modes before a solution is attempted on the level of the finite element system: the global stiffness matrix should not be singular.

Therefore, the stability requirement is essentially reduced to demanding that the *element stiffness matrices* be of *proper rank*, which means of rank that allows only for zero-energy modes in the form of rigid body motion. The element stiffness matrix $\boldsymbol{K}_e$ which links together n degrees of freedom (all the displacement components at all the nodes) should have the rank

$$\text{rank}\boldsymbol{K}_e = n - n_{\text{RBM}} \ ,$$

where n_{RBM} is the number of rigid body modes the element possesses (=6 in 3-D, and so on). To dig deeper, we recall Eq. (14.48), which we supplement with numerical quadrature

$$\boldsymbol{K}_e = \int_{V_e} \boldsymbol{B}^{eT} \boldsymbol{D} \boldsymbol{B}^e \ \mathrm{d}V \approx \sum_k \boldsymbol{B}^e(\boldsymbol{\xi}_k)^T \boldsymbol{D}(\boldsymbol{\xi}_k) \boldsymbol{B}^e(\boldsymbol{\xi}_k) J(\boldsymbol{\xi}_k) w_k \ . \tag{18.1}$$

Assuming the full rank of the material stiffness matrix (positive definite!), the requirement that the element stiffness matrix have a proper rank may be translated into a counting procedure for the integration points: each quadrature point adds the matrix

$$\boldsymbol{B}^e(\boldsymbol{\xi}_k)^T \boldsymbol{D}(\boldsymbol{\xi}_k) \boldsymbol{B}^e(\boldsymbol{\xi}_k) J(\boldsymbol{\xi}_k) w_k \ ,$$

which has the rank of $\text{rank}\boldsymbol{D}(\boldsymbol{\xi}_k)$, with the provisions $J(\boldsymbol{\xi}_k) \neq 0$, and $w_k \neq 0$. If each quadrature point increases the rank of $\boldsymbol{K}_e$ by $\text{rank}\boldsymbol{D}(\boldsymbol{\xi}_k)$, the requisite number of integration points will be

$$n_{qp} \geq \frac{n - n_{\text{RBM}}}{\text{rank}\boldsymbol{D}} \ . \tag{18.2}$$

For instance, for the quadratic tetrahedron T10, the number of quadrature points should be

$$n_{qp} \geq \frac{30 - 6}{6} = 4 \ ,$$

and that is indeed satisfied by the four-point rule from Table 11.2.

It should be noted that the Jacobian could cause some trouble. Figure 18.3 shows a few possible shapes of a quadratic triangle obtained by shifting the mid-side nodes from the ideal shape of the equilateral triangle (a). The triangles (b) and (d) represent shapes that are commonly used to better approximate curved boundaries, either concave or convex. Moderate excursions of the nodes from the ideal locations at the midpoints of the edges typically do not lead to major difficulties. The triangle (c) is produced by shifting the mid-side nodes inwards excessively so that the surface of the triangle is turned inside out, resulting in a negative Jacobian in the corner regions (zero Jacobian where the edges intersect). The triangle (e) is obtained by shifting the mid-side node outwards and along the

edge towards the corner. Beyond a certain point (compare with triangle (f)), the Jacobian again becomes negative in the vicinity of the corner. (To realize what is happening, note that a parabolic arc needs to be passed through the three nodes along an edge.) Negative or zero Jacobians are a problem, both in terms of the rank of the element stiffness matrix, and in terms of the interpretation of the strains produced on such elements.

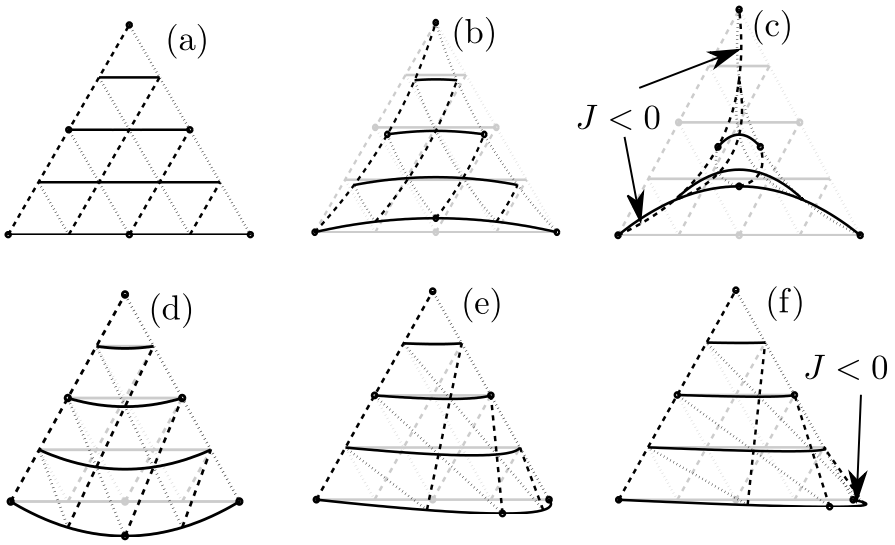


Fig. 18.3. Shapes of quadratic triangles. (a) undistorted triangle with a constant positive Jacobian, (b), (c) produced by moving mid-side nodes inwards, (d) one edge node moved outwards, (e), (f) one edge node moved outwards and towards the right hand side corner. Note that the Jacobian is negative at some points for cases (c) and (f).

18.2.1 Conclusion

The following formal equality has been coined for finite difference methods, but will also work for the finite elements here:

$$\boxed{\text{Consistency} + \text{Stability} = \text{Convergence.}}$$

We have shown, that convergence for properly defined quantities (not for infinite stresses!) is available with the methods discussed in this book: they possess the property of consistency, and with proper integration, they may be endowed with stability.

Some of the above requirements may be mollified or relaxed, and there may be good reasons for doing so: for instance, improving performance on coarse meshes by relaxing the constraints on the compatibility. However, as a general guide towards convergent finite element approximations, the above dictum will be a safe starting point.

Exercises

1. Consider a T6 element which is mapped to the physical space with the identity map, $x = \xi$, $y = \eta$. Move the node 6 to $x = \overline{\xi}$, $y = 1/2$. Find $\overline{\xi}$ such that the Jacobian becomes zero in at least one point.
2. Deduce how many Gauss integration points will be needed to ensure proper rank of the conductivity matrix for the element Q4.

3. Compute the rank of the conductivity matrix of a well-shaped T6 element using a one-point quadrature. Is the number of (nearly) zero eigenvalues consistent with the argument in Eq. (18.2)?
4. Deduce how many Gauss integration points will be needed to ensure proper rank of the conductivity matrix for the element Q8.

References

[B99] Barber, J. R. (1999). Elasticity, Kluwer Academic Publishers, reprint edition.

[B01] Blevins, R. D. (2001). Formulas for Natural Frequency and Mode Shape, Krieger publishing company, reprint edition.

[BS89] Bogacki, P. and L. F. Shampine (1989). A 3(2) pair of Runge-Kutta formulas, Appl. Math. Letters, Vol. 2, pp 1-9.

[CCS94] Cameron, A. D. and J.A. Casey, G.B. Simpson (1994). Benchmark Tests for Thermal Analyses, NAFEMS Documentation .

[G91] Graff, K. F. (1991). Wave motion in elastic solids. Dover publications, New York.

[H00] Hughes, Thomas J. R. (2000). The finite element method. Linear static and dynamic finite element analysis. Dover publications, New York.

[H98] Hyer, M. W. (1998). Stress Analysis of Fiber-Reinforced Composite Materials, WCB/McGraw-Hill.

[I05] Infolytica (2005). `http://www.infolytica.com/en/coolstuff/ex0047/`

[L05] Lienhard IV, John H. and John H. Lienhard V (2005). A Heat Transfer Textbook, 3rd edition, `http://web.mit.edu/lienhard/www/ahtt.html`.

[Matlab] Matlab online documentation.
`http://www.mathworks.com/access/helpdesk/help/helpdesk.html`

[M04] Moler, C. (2004). Numerical Computing with MATLAB, Electronic edition: The MathWorks, Inc., Natick, MA, 2004. `http://www.mathworks.com/moler`
Print edition: SIAM, Philadelphia, 2004.
`http://ec-securehost.com/SIAM/ot87.html`

[P97] Pilkey, W. D. (1997). Peterson's Stress Concentration Factors. John Wiley & Sons, second edition.

[S83] Sokolnikoff, I. S. (1983). Mathematical Theory of Elasticity. Krieger publishing company. Reprint edition 1983.

[T83] Timoshenko, S. P. (1983). History of Strength of Materials. Dover, 1983.

[TW-K59] Timoshenko, S. P. and S. Woinowsky-Krieger (1959). Theory of Plates and Shells. McGraw-Hill, 1959.

[TB97] L. N. Trefethen, D. Bau III, Numerical Linear Algebra, SIAM: Society for Industrial and Applied Mathematics, 1997.

[ZT89] Zienkiewicz, O. C. and R. L. Taylor (1989). The finite element method. Vol. I. Basic formulations and linear problems. London: McGraw-Hill.

Index