Programs and Listings

Introduction

It is the intention in this Addenda to provide the reader of my book "Numerical Methods for Laplace Transform Inversion" with a variety of programs to evaluate the inverse Laplace transform. The only requirement is the ability to write simple Fortran routines to evaluate the transform function $\bar{f}(s)$. As a general rule the programs require the use of double precision arithmetic - this is particularly the case with NAG library routines - but because the extrapolation methods used involve a lot of cancellation it is preferable to work with multi-precision arithmetic. It has been the policy of the author to indicate double/multiple precision variables by using the term **real**, that is real in bold characters, and also to indicate constants of this form by writing them in bold type. Thus **0.01** would represent 1d-2 in double precision or 1q-2 in quadruple precision and $\mathbf{10^{-10}}$ would represent 1d-10 or 1q-10. Similarly, **complex** represents double/multiple precision complex variables. Likewise **exp** represents dexp in double precision and qexp in quadruple precision and indicates that the exponential operation will produce numerical results which are compatible with the variable declaration. Similarly for **sin**, **cos**, **abs**, **float**, and so on. For some functions we have used the generic name in the program. Examples of this are max, min and ifix where, depending on the computer used and the context we might require dmax1, dmin1 or jifix respectively — or some similar function names.

We now give examples of various programs and their specifications. It should be borne in mind that the parameters needed to operate some programs need careful choosing and it would be advisable to rerun the programs with alternative parameter settings. While the author has attempted to write optimal programs for routines which have not been published this objective might not have been achieved in all cases. Additionally, the user has to be aware of the fact, especially when supplying $\bar{f}(s)$ which is to be evaluated for complex values of s, of the need to determine the principal value of $\bar{f}(s)$.

Also included in this Chapter are two Mathematica programs. Some additional Mathematica programs can be found from the package LaplaceAndzTransforms — see Graf [5]. Further details relating to the NAG programs can be found at their website [9].

Program for Weeks's method

This is based on the paper by Garbow et al [4] which determines a Laguerre series approximation $\tilde{f}(t)$ to $f(t)$ of the form

$$\tilde{f}(t) = e^{\sigma t} \sum_{i=0}^{m-1} a_i e^{bt/2} L_i(bt), \quad \sigma > \sigma_0, \quad b > 0$$

where σ_0 (=γ) is the abscissa of the singularity of $\bar{f}(s)$ which is furthest from $\Re s = 0$. NAG routine C06LBF computes the coefficients a_i and the routine C06LCF evaluates the expansion for specified t.

Entry/Exit Parameters

1. f — **complex** function to be supplied by user. f must return the value of the Laplace transform $\bar{f}(s)$ for given s.
2. sigmao — **real**. The value of σ_0.
3. sigma — **real**, the parameter σ of the Laguerre expansion. It is replaced by $\sigma_0 + 0.7$ on exit if $\sigma < \sigma_0$.
4. b — **real**. If $b < 2(\sigma - \sigma_0)$ on entry it is replaced by $2.5(\sigma - \sigma_0)$.
5. epstol — **real**. An upper bound on $|f(t) - \tilde{f}(t)|e^{-\sigma t}$.
6. nmax — integer. An upper bound for the number of coefficients actually computed. It is recommended that nmax is a power of 2 (≥ 8) and 1024 is sufficient in most cases.
7. m — integer. The number of Laguerre expansion coefficients actually computed.
8. acoef(nmax) — **real** array. On exit, the first m elements contain the computed coefficients a_i.
9. errvec(8) — **real** array. On exit contains diagnostic information. In particular, errvec(1) contains an overall estimate for $|f(t) - \tilde{f}(t)|e^{-\sigma t}$. The NAG Library Manual should be consulted for further information.
10. ifail — integer. On exit ifail=0 unless the routine detects an error. The NAG Library Manual should be consulted for details about the setting of ifail on entry and the significance of the output values of ifail.
11. t — **real**. On entry is the value of t for which $f(t)$ is required.
12. finv — **real**. On exit the approximation of the inverse Laplace transform at t.

Full details of the application of the routines can be found in the NAG Library Manual. There follows a sample program to determine the inverse Laplace transform at the values t_i, $i = 1, \cdots, n$ which

are to be read into the computer.

```
c       This is an application of NAG Library Routines C06LBF/C06LCF
        integer nmax, nout
        parameter(nmax=512, nout=6)
        real b, epstol, exact, finv, pserr, sigma, sigmao, t
        integer ifail, j, m
        real acoef(nmax), errvec(8)
        external C06LBF, C06LCF
        complex f
        external f

        sigmao = 0.1
        epstol = 0.00001
        ifail=0
        call C06LBF(f, sigmao, sigma, b, epstol, nmax, m, acoef, errvec, ifail)
        write(nout,999) 'number of coefficients=', m
        write (nout,*) '          t          f(t)          '

        read(5,*)n
c       reads number of values of t at which f(t) required
        do i=1,n
        read(5,*)t
        call C06LCF(t, sigma, b, m, acoef, errvec, finv, ifail)
        write(nout,998)t,finv
        enddo
        stop
999     format(1x, a, i6)
998     format(1x, 1p, d10.2, d15.4)
        end

        complex function f(s)
c       f̄(s) is the transform of J_0(t)
        complex s, z
        real pr, pi, qr, qi
        external A02AAF
c       The NAG routine A02AAF evaluates complex square roots
        z=s*s+1
        pr=real(z)
```

pi=**imag**(z)
call A02AAF(pr, pi, qr, qi)
z=**cmplex**(qr, qi)
f= (**1.0**)/z
return
end

Padé Table Construction by method of Longman
The program LONGPAD

Given a series

$$g(s) = a_0 + a_1 s + a_2 s^2 + \cdots + a_n s^n + \cdots$$

the program LONGPAD determines the Padé approximants

$$\frac{\beta_0 + \beta_1 s + \cdots + \beta_p s^p}{\gamma_0 + \gamma_1 s + \cdots + \gamma_q s^q},$$

for $0 \leq p, q \leq [(n-1)/2]$.

Entry/Exit Parameters
1. n — integer. This should be an odd number and declared as a parameter.
2. a(i) — **real**. These are the coefficients in the series which have to be provided by the user.
3. b(i, j, k) — **real** array. On exit will correspond to the β_k of the $[i/j]$ Padé approximant.
4. g(i, j, k) — **real** array. On exit will correspond to the γ_k of the $[i/j]$ Padé approximant — see Longman [7]

The program below computes the diagonal terms in the Padé table for the function $s\bar{f}(s)$ of Chapter 6, §3.

```
c       Requires the coefficients a_i in the Taylor series expansion
        parameter (n=11)
        real a(0:n), g(0:n, 0:n, 0:n), b(0:n, 0:n, 0:n), sig
        read(5,*)sig
        a(0)=1.0
        a(1)=-1.0
        a(2)=(1+sig)/2
        sig=sig/2
        do i=3, n-1
        call coeff(i,sig,a)
        write(6,*)i, a(i)
        enddo
        call longpad(n, a, b, g)
        m=(n-1)/2
```

```
      do k=0, m
      write(6,*)g(m, m, k), b(m, m, k)
      enddo
      stop
      end

      subroutine longpad(n, a, b, g)
      real a(0:n), g(0:n, 0:n, 0:n), b(0:n, 0:n, 0:n), x, c(0:n)
      if (n.le.25) goto 100
      write(6,*)'n too large (> 25)'
      stop
100   c(0)=1.0
      c(1)=-a(1)/a(0)
      do i=2, n-1
      x=a(i)
      do j=1, i-1
      x=x+c(j)*a(i-j)
      enddo
      c(i)=-x/a(0)
      write(6,*)i, c(i)
      enddo

      do j=0, n-1
      do k=0, j
      g(j, 0, k)=c(k)
      b(0, j, k)=a(k)
      enddo
      enddo

      do i=1, n-1
      m=n-i
      do j=0, m
      b(i, j,0)=a(0)
      enddo
      enddo

      do j=1, n-1
      m=n-j
      do i=0, m
```

```
      g(i, j, 0)=1.0
      enddo
      enddo

      do i=1, n-2
      m=n-i
      do j=1, m
      do k=1, j
      x=b(i-1, j, k-1)*b(i-1, j+1, j+1)/b(i-1, j, j)
      b(i, j, k)=b(i-1, j+1, k)-x
      enddo
      enddo
      enddo

      do j=1, n-2
      m=n-j
      do i=1, m
      do k=1, i
      x=g(i, j-1, k-1)*g(i+1, j-1, i+1)/g(i, j-1, i)
      g(i, j, k)=g(i+1, j-1, k)-x
      enddo
      enddo
      enddo
      return
      end

      subroutine fact(k, fac)
      real fac
      fac=1.0
      if (k.eq.0) goto 100
      do j=1, k
      fac=j*fac
      enddo
100   return
      end

      subroutine coeff(k, sig, a)
      real c(k), a(k), y, sig
      y=1.0
```

```
      call fact(k, fac)
      c(0)=y/fac
      call fact(k-2, fac)
      c(1)=y/fac
      if (k.gt.3) goto 100
      c(2)=(3*y)/2
      goto 200

100   do i=2, k-1
      c(i)=y
      do j=k-i+2, k+i-2, 2
      c(i)=j*c(i)
      enddo
      call fact(k-i-1, fac)
      c(i)=c(i)/fac
      call fact(i, fac)
      c(i)=c(i)/fac
      enddo

200   a(k)=c(k-1)
      do i=k-2, 0, -1
      a(k)=a(k)*sig+c(i)
      enddo
      do i=1, k
      a(k)=-1.0*a(k)
      enddo
      return
      end
```

Program for Sidi's method of Laplace Inversion

This program is based on the variation of Sidi's method given in §5.3 . The user has to supply two function subroutines fun1(c,x,t,f) and fun2(c,x,t,f). The former will work out $\Re \bar{f}(s)$ for given t and place it in location f, where $s = c + it$. The latter likewise works out $\Im \bar{f}(s)$. Numerical integration is performed by means of a Gauss 64-point quadrature formula for $\int_0^1 g(x)dx = \sum_1^{64} w_i g(x_i)$. For smaller t it might be preferable to use a higher order Gaussian quadrature formula or an adaptive approach based on the 64-point formula.

Entry/Exit Parameters.

1. t — a **real** number corresponding to the value at which the inverse transform is required.
2. c — a **real** parameter which has been set at **0.2** but must be increased to ensure that c is greater than the real part of any singularity of $\bar{f}(s)$.
3. x — a **real** variable which equals $\Im s$. This is required for the function subroutines fun1 and fun2.
4. n — integer, the number of points used by the Gauss quadrature rule. The default value is 64. A change in n would require an update of the subroutine gauss(n,x,w).
5. m — integer, the number of terms used in the mW - extrapolation. The default value is 40 but this can be altered as long as it is not too drastically different.
6. a — **real**, on exit should be a good approximation to the inverse transform if it agrees with b.
7. b — **real**, on exit should be a good approximation to the inverse transform if it agrees with a.

If a and b are not equal then the calculation should be repeated with the "c" qualifying the print instruction suppressed in the subroutine extrap. If the successive iterations show some consistency for a, say, then a can be accepted as an approximation to the transform. (See Chapter 10).

```
parameter (n=64, m=40)
real x(n), w(n), a, b, t, c, pi, xa, xb, ga, gb
real psia(0:m), fa(0:m),za(0:m+1)
real psib(0:m), fb(0:m), zb(0:m+1)
parameter (c=0.2)
```

```
      write(6,*)'n=', n, 'm=', m, 'c=', c
      pi=acos(-1.0)
      read(5,*)t
      do i=1, m+1
      za(i-1)=(i*pi)/t
      zb(i-1)=za(i-1)
      enddo
      za(0)=za(0)/2
      call gauss(n, x, w)

      fa(0)=0
      k=0
      psia(k)=0
      do i=1, n
      xa=k*za(0)+za(0)*x(i)
      call fun1(c, xa, t, ga)
      psia(k)=psia(k)+w(i)*ga
      enddo
      psia(k)=(pi*psia(k))/2

      fb(0)=0
      psib(k)=0
      do i=1, n
      xb=k*zb(0)+zb(0)*x(i)
      call fun2(c, xb, t, gb)
      psib(k)=psib(k)+w(i)*gb
      enddo
      psib(k)=pi*psib(k)
c     write(6,*)k, psia(k), psib(k)

      do k=1, m
      psia(k)=0
      psib(k)=0
      do i=1, n
      xa=(2*k-1)*za(0)+2*za(0)*x(i)
      xb=k*zb(0)+zb(0)*x(i)
      call fun1(c, xa, t, ga)
      psia(k)=psia(k)+w(i)*ga
      call fun2(c, xb, t, gb)
```

```
      psib(k)=psib(k)+w(i)*gb
      enddo
      psia(k)=psia(k)*pi
      fa(k)=fa(k-1)+psia(k-1)
      psib(k)=psib(k)*pi
      fb(k)=fb(k-1)+psib(k-1)
c     write(6,*)fa(k), fb(k)
      enddo

      call extrap(m, psia, fa, za, a)
      a=2*a/pi
      a=(a*exp(c*t))/t
      write(6,*)'a=', a
      call extrap(m, psib, fb, zb, b)
      b=2*b/pi
      b=(b*exp(c*t))/t
      write(6,*)'b=', b
      stop
      end

      subroutine fun1(c, x, t, f)
c     This subroutine evaluates Re^{-4sqrt(s)}
      real c, x, t, f, u, v
      complex s, w
      s=cmplex(c, x)
      w=-4.0*sqrt(s)
c     sqrt denotes complex square root
      w= exp(w)
      f= real(w)
      f=f* cos(t*x)
      return
      end

      subroutine fun2(c, x, t, f)
c     This subroutine evaluates Im e^{-4sqrt(s)}
      real c, x, t, f, u, v
      complex s, w
      s=cmplex(c, x)
      w=-4.0*sqrt(s)
```

```
c       sqrt denotes complex square root
        w= exp(w)
        f= imag(w)
        f=-f* sin(t*x)
        return
        end

        subroutine extrap(m, psi, f, z, a)
        real psi(0:m), f(0:m), z(0:m), p(0:m), q(0:m), a, b
        do j=0, m
        p(j)=f(j)/psi(j)
        q(j)= 1.0/psi(j)
        enddo
        do k=1, m
        do j=0, m-k
        p(j)=p(j+1)-p(j)
        q(j)=q(j+1)-q(j)
        b=( 1.0/z(j+k))-( 1.0/z(j))
        p(j)=p(j)/b
        q(j)=q(j)/b
        enddo
        a=p(0)/q(0)
c       write(6,*)a
c       the write statement in previous line should be reinstated
c       if there is a significant difference between the
c       estimates 'a' and 'b' for the Laplace transform
        enddo
        return
        end

        subroutine gauss(n, x, w)
        real x(n), w(n)
        x(1)=0.99965252086788606972845281217 2818
        w(1)=0.00089164036084821647364803957248 98344
        x(2)=0.99817005838597763967346225033 8200
        w(2)=0.00207351663028123381764376786427 6530
        x(3)=0.99550668573837216036969119172 1652
        w(3)=0.00325222898448918142805868019 9989531
        x(4)=0.99166812694231295846564965107 8416
```

w(4)=0.004423379913181973861515457329864311
x(5)=0.986663413894955481870926753676136
w(5)=0.005584069730065564409295246509604289
x(6)=0.980504399826026859459307060948579
w(6)=0.006731523948359321299030383342978218
x(7)=0.973205687429201408031240745673632
w(7)=0.007863015238012359660982997648769673
x(8)=0.964784586065969787910745077279613
w(8)=0.008975857887848671542522651000559224
x(9)=0.955261068539251402878190334004165
w(9)=0.010067411576765104686170158364271 80
x(10)=0.944657722997557052926702019136426
w(10)=0.011135086904191627079649165192077 27
x(11)=0.932999699077046409880391692535079
w(11)=0.012176351284355436669088775204534 30
x(12)=0.920314648126290181375845772347937
w(12)=0.013188734857527329335845896312612 80
x(13)=0.906632657561398779870961669043152
w(13)=0.014169836307129741613755652600118 66
x(14)=0.891986179471670703805110262606884
w(14)=0.015117328536201239433987029909774 21
x(15)=0.876409953630265948305931887442847
w(15)=0.016028964177425776792733752173949 22
x(16)=0.859940925085805413424470108915974
w(16)=0.016902580918570804695782741055362 63
x(17)=0.842618156527116621281779185515688
w(17)=0.017736106628441191905346573357623 11
x(18)=0.824482735627328669928880615996702
w(18)=0.018527564270120023020207550904791 65
x(19)=0.805577677586196625124426485509274
w(19)=0.019275076589307814564481248473404 55
x(20)=0.785947823101317017141939058329594
w(20)=0.019976870566360170693328463064168 00
x(21)=0.765639732009947272829006951772228
w(21)=0.020631281621311764305078148736819 98
x(22)=0.744701572853526478739263153510961
w(22)=0.021236757561826794503669883954408 69
x(23)=0.723183008626732043992473857379458
w(23)=0.021791862264661726684841393048686875

x(24)=**0.70113507898199580184788338563007 9**
w(24)=**0.02229527908187828153006735501547241**
x(25)=**0.67861007916883405797522130752310 1**
w(25)=**0.02274581396370907223988549848563456**
x(26)=**0.65566143599510547807875634928007 8**
w(26)=**0.02314239829065720864797662461613062**
x(27)=**0.63234358110438370818698208625501 0**
w(27)=**0.02348409140810500866266314287729056**
x(28)=**0.60871182187000354207482437449441 1**
w(28)=**0.02377008285741515433114110347211160**
x(29)=**0.58482221021199640901865681487413 5**
w(29)=**0.02399969429822915386406308993567301**
x(30)=**0.56073140964806027723518823174612 4**
w(30)=**0.02417238111740147858488476357900890**
x(31)=**0.53649656089389951972477147097016 9**
w(31)=**0.02428773372075171346739953339198905**
x(32)=**0.51217514633171221625447792142685 8**
w(32)=**0.02434547850456986019168269536737497**
j=n/2
do i=1, j
x(j+i)=1-x(j-i+1)
w(j+i)=w(j-i+1)
enddo
return
end

Implementation of the $d^{(m)}$-transformation via the $W^{(m)}$-Algorithm

The $W^{(m)}$-Algorithm is a way of implementing the $d^{(m)}$ transformation — see §12.4, Ford and Sidi [3] and Sidi [10].

Entry/Exit Parameters.

1. cf(I) — **real** array which contain the elements of the series to be summed. In the program which follows the program is applied to two cases of series summation where the general terms are respectively $(1/n^{3/2} + 2/n^2)$ and $(\cos n)/n$ and also to finding the limit of the Post-Widder sequences relating to $\bar{f}(s) = 1/(s+1)$ and $\bar{f}(s) = 1/\sqrt{(s^2+1)}$ (see §8.1).

2 m — integer. m specifies the transformation being used and must lie between 1 and mdim (mdim is usually chosen to be 6). For the two series considered one should take m=2 and we can take m=1 for the Post-Widder applications.

3. lmax — integer which specifies the number of terms of the series to be computed and which must be less than ldim.

4. kappa — **real**. This must be ≥ 1.0 and is required for arithmetic progression sampling (aps).

5. sigma — **real**. This must be $\geq$1.0 and is required for geometric progression sampling. sigma must be 1 for aps to be invoked

6. epsdiv — **real**. A small number which is used in testing the magnitude of the approximation approx(j,p).

7. approx(j,p) — **real** array with $0 \leq$ j $\leq$ ldim, $0 \leq$ p $\leq$ ldim. These approximations are returned by the $d^{(m)}$ transformation

8. cf(I) — a function subprogram which has to be supplied by the user. In the case of the Post-Widder applications the value of t required in the Laplace inversion must also be supplied.

9. np — integer giving the specific sequence whose limit is required.

```
c       Implementation of the Ford-Sidi W^(m)- Algorithm
        implicit real (a - h, o - z)
        parameter(mdim=6, ldim=50, epsdiv=10^-77)
        parameter (np=2, m=2, lmax=20, incr=1, sigma=1.3)
        dimension g(mdim), psiai(0:ldim, 2, 2), bigpsi(0:ldim, mdim, 2)
        dimension psig(0:mdim, 2:mdim+1, 2), approx(0:ldim, 0:ldim)
        external mltag
```

```
      common /sigkap/sigmap, kappap
      common /np/npp/theta/theta
      common /rl/irl(0:ldim)
      write(6,111)
111   format( 'summation of cf(I)/ limit of Post-Widder sequence')
      npp=np
      theta=1.q0
      sigmap=sigma
      kappap=kappa
      call wmalgm(mdim, ldim, m, lmax, mltag, g, psiai, bigpsi, psig,
     + approx, epsdiv)
      write(6,121)
121   format(/, 3x, 'l', 3x, 'r_l', 3x, 'error(l,0)', 3x,'error(0,l)')
      do l=0, lmax
      er1=approx(l,0)
      er2=approx(0,1)
      write(6,131) l, irl(l), er1, er2
      enddo
131   format(2i4, d25.16)
      stop
      end

      subroutine wmalgm(mdim, ldim, m, lmax, mltag, g, psiai, bigpsi, psig,
     + approx, epsdiv)
      implicit real (a - h, o - z)
      integer cur, temp, p, pm, q, qp
      dimension g(mdim), psiai(0:ldim, 2, 2)
      dimension bigpsi(0:ldim, mdim, 2), psig(0:mdim, 2:mdim+1, 2)
      dimension approx(0:ldim, 0:ldim)
      cur=1
      temp=2
      call mltag(m, 0, t, a, g)
      approx(0, 0)=a
      psiai(0, 1, cur)=a/g(1)
      psiai(0, 2, cur)=1.0/g(1)
      bigpsi(0, 1, cur)=1.0/t
      do k=2, m
      psig(0, k, cur)=g(k)/g(1)
      enddo
```

```
psig(0, m+1, cur)=t
do 80 l=1, lmax
call mltag(m, l, t, a, g)
approx(l, 0)=a
psiai(0, 1, temp)=a/g(1)
psiai(0, 2, temp)=1.0/g(1)
bigpsi(0, 1, temp)=1.0/t
do k=2, m
psig(0, k, temp)=g(k)/g(1)
enddo
psig(0, m+1, temp)=t
sign=-1.0
do 60 p=1, l
if (p.le.m) then
d=psig(p-1, p+1,temp)-psig(p-1, p+1, cur)
do i=p+2, m+1
psig(p, i, temp)=(psig(p-1, i, temp)-psig(p-1, i, cur))/d
enddo
end if

if (p.lt.m) then
bigpsi(p, p+1, temp)=sign/psig(p, m+1, temp)
sign=-sign
end if
pm=min0(p-1, m-1)
do q=1, pm
ps=bigpsi(p-2, q, cur)
dq=ps/bigpsi(p-1, q, cur)-ps/bigpsi(p-1, q, temp)
qp=q+1
bigpsi(p, qp, temp)=(bigpsi(p-1, qp, temp)-bigpsi(p-1, qp, cur))/dq
enddo
if (p.gt.m) then
ps=bigpsi(p-2, m, cur)
d=ps/bigpsi(p-1, m, cur)-ps/bigpsi(p-1, m, temp)
end if
bigpsi(p, 1, temp)=(bigpsi(p-1, 1, temp)-bigpsi(p-1, 1, cur))/d
do i=1, 2
psiai(p, i, temp)=(psiai(p-1, i, temp)-psiai(p-1, i, cur))/d
enddo
```

```
60      continue
        do 70 p=1, l
        j=l-p
        if (abs(psiai(p, 2, temp)).ge.epsdiv) then
        approx(j, p)=psiai(p, 1, temp)/psiai(p, 2, temp)
        else
        approx(j, p)=10**75
        write(6,101)j, p
101     format(1x, 'approx(', i3, ',', i3, ') is not defined')
        end if
70      continue
        jj=cur
        cur =temp
        temp=jj
80      continue
        return
        end

        subroutine mltag(m, l, t, a, g)
        implicit real(a-h, o-z)
        real kappa
        parameter(ldim=50)
        dimension g(m)
        common /sigkap/sigma, kappa
        common /rl/irl(0:ldim)

        if (sigma.eq.1.0) then
        lsump=kappa*l+10**-10
        lsum=kappa*(l+1)+10**-10
        end if
        if (sigma.gt.1.0) then
        if(l.eq.0) then
        lsump=0
        lsum=1
        else
        lsum=1
        lsump=1
        do i=1, l
        do i=0, lsum
```

```
ir= sigma*lsum+10**-10
if (ir.le.lsum) then
lsum=lsum+1
else
lsum=ir
end if
if (i.eq.l-1)lsump=lsum
enddo
end if
end if
irl(l)=lsum
if (l.eq.0) a=0
do i=lsump+1, lsum
a=a+cf(i)
enddo
p=lsum
t=1.0/p
do k=1, m
g(k)=cf(lsum+k-1)
enddo
do i=2, m
do j=m, i, -1
g(j)=g(j)-g(j-1)
enddo
enddo
do k=1, m
g(k)=g(k)*p
p=p*lsum
enddo
do k=1, m/2
st=g(k)
g(k)=g(m-k+1)
g(m-k+1)=st
enddo
return
end

function cf(I)
implicit real(a - h, o - z)
```

```
      common /np/np/theta/theta
      a=float(i)
      if (np.eq.1) cf=1.0/a**1.5+2.0/a**2
      if (np.eq.2) cf=cos(a*theta)/a
      if (np.eq.3) then
      t=10.0
c     value of t for which inverse transform is required
      b=-(a+1.0)
      a1=a-1
      b1=-(a+1)
      if (i.gt.1) goto 150
      cf=(1.0+(t/a))**b
      goto 100
150   cf=(1+(t/a))**b-(1+(t/a1))**b1
100   end if
      if (np.eq.4)then
      t=2.0
      s=a/t
      v=s*s+1
      w=1.0/sqrt(v)
      z=-(s*w)/v
      s1=(a-1)/t
      v1=s1*s1+1
      w1=1.0/sqrt(v1)
      z1=-(s1*w1)/v1
      if (i.gt.1) goto 200
      cf=-s*s*z
      goto 250
200   b=1.0
      do j=1, i-1
      b=j*b
      enddo
      do j=2, i
      y=-((2*j-1)*s*z+(j-1)*(j-1)*w)/v
      w=z
      z=y
      enddo
      if (i.gt.2) goto 300
      y1=z1
```

```
      goto 350
300   do j=2, i-1
      y1=-((2*j-1)*s1*z1+(j-1)*(j-1)*w1)/v1
      w1=z1
      z1=y1
      enddo
350   cf=(-((-s)**(a+1))/(a*b))*y+((-s1)**a/b)*y1
250   end if
      return
      end
```

Program for Crump's Method

Crump's method consists of the application of the epsilon algorithm to the summation in Durbin's Fourier series approximation

$$f(t) \approx \frac{e^{ct}}{T}\left[\tfrac{1}{2}\Re\{\bar{f}(c)\} + \sum_{k=1}^{\infty}\Re\left\{\bar{f}\left(c+\frac{ik\pi}{T}\right)\right\}\cos\left(\frac{k\pi t}{T}\right) - \sum_{k=1}^{\infty}\Im\left\{\bar{f}\left(c+\frac{ik\pi}{T}\right)\right\}\sin\left(\frac{k\pi t}{T}\right)\right]$$

See reference, Crump [2].

Entry/Exit parameters

1. FUN — a subroutine which evaluates the real and imaginary parts of $\bar{f}(s)$ (see sample program). This must be declared as external in the program which calls C06LAF.
2. n — integer. n signifies the number of points at which the inverse Laplace transform is required.
3. t(n) — **real** array. On entry $0.0 \leq t(1) < t(2) < \cdots < t(n)$ and the t(j) represent the points at which $f(t)$ is to be determined.
4. valinv(n) — **real** array. On exit gives an estimate for $f(t)$ for each $t(j)$.
5. errest(n) — **real** array. On exit gives an estimate for the relative errors of the components in valinv(n). If valinv(j) < relerr, then errest(j) estimates absolute error.
6. relerr — **real**. On entry the required relative error in the values of the inverse Laplace transform.
7. alphab — **real**. An upper bound for c. Ideally it should equal c or be slightly larger. If it is less than c the prescribed accuracy might not be attained.
8. tfac — **real**. On entry a factor to be used in calculating the value of T. The suggested value of tfac is **0.8** but it must be greater than **0.5**.
9. mxterm — integer. On entry the maximum number of terms used in the evaluation of the Fourier series. It is suggested that mxterm ≥ 100. Certainly mxterm ≥ 1.
10. nterms — integer. On exit gives the number of terms of the Fourier series actually used.
11. na — integer. The number of values of c used by the routine.
12. alow/ahigh - **real**. Respectively the lowest and highest values values of c used in the algorithm. These can be used for checking

alphab.
13. nfeval — integer. On exit records the number of calls made to the subroutine FUN.
14. work(4*mxterm+2) — **real** array. used as working space.
15. ifail — integer. On entry ifail must be set to 0, -1 or 1. On exit ifail = 0 unless the routine detects an error. See the NAG Library specification for more details regarding error indications.
The NAG Library specification should also be consulted for further details on Accuracy and the choice of alphab and other precautions which are advised.

```
c       Program for Crump's method
c       Uses NAG Library routine C06LAF
        integer nmax, mxterm, nout
        parameter (nmax=20, mxterm=500, nout=6)
        real ahigh, alow, alphab, relerr, tfac
        integer i, ifail, n, na, nfeval, nterms
        real errest(nmax), t(nmax), valinv(nmax), work(4*mxterm+2)
        external C06LAF, FUN

        read(5,*)n
        do i=1, n
        read(5,*)t(i)
        enddo

        write(nout,*)'alphab=?'
        read(5,*)alphab
        write(nout,*)'relerr=?'
        read(5,*)relerr
        write(nout,*)'tfac=?'
        read(5,*)tfac
        write(nout,*)
        write(nout,999)'mxterm=', mxterm, 'tfac=', tfac,
     +  'alphab=', alphab, 'relerr=', relerr
        ifail=-1

        call C06LAF(FUN, n, t, valinv, errest, relerr, alphab, tfac, mxterm,
     +  nterms, na, alow, ahigh, nfeval, work, ifail)
```

```
      if(ifail.gt.0 .and. ifail.lt.5)goto 60
      write(nout,*)
      write(nout,*) '          t          result          '

      do i=1, n
      write(nout,998) t(i), valinv(i)
      enddo
      write(nout,997)'nterms=', nterms,    'nfeval=', nfeval,
     + 'alow=', alow,    'ahigh=', ahigh,    'ifail=', ifail

c     test for larger values of c

      relerr=1.0^-3
      tfac=0.8
      write(nout,*)
      write(nout,999)'mxterm=', mxterm,    'tfac=', tfac,
     + 'alphab=', alphab,    'relerr=', relerr
      ifail=-1
      call C06LAF(FUN, n, t, valinv, errest, relerr, alphab, tfac, mxterm,
     + nterms, na, alow, ahigh, nfeval, work, ifail)

      if (ifail.gt.0 .and. ifail.lt.5)goto 60

      write(nout,*)
      write(nout,*) '          t          result          '
      do i=1, n
      write(nout,998) t(i), valinv(i)
      enddo
60    write(nout,*)
      write(nout,997)'nterms=', nterms,    'nfeval=', nfeval,
     + 'alow=', alow,    'ahigh=', ahigh,    'ifail=', ifail

999   format(1x, a, i4, a, f6.2, a, f6.2, a,1p, d8.1)
998   format(1x, f4.1, 7x, f9.6)
997   format(1x, a,i4, a, i4, a, f7.2, a, f7.2, a, i2)
      stop
      end

      subroutine FUN(pr, pi, fr, fi)
```

```
c       evaluates f(s) = 1/√(s^2 + 1)
        real pr, pi, fr, fi, zr, zi
        external A02AAF
c       NAG routine A02AAF evaluates the square root of a complex number
        external A02ACF
c       NAG routine A02ACF evaluates the inverse of a complex number
        fr=pr*pr-pi*pi+1.0
        fi=2.0*pr*pi
        call A02AAF(fr, fi, zr, zi)
        call A02ACF(1.0, 0.0, zr, zi, fr, fi)
        return
        end
```

Program for Post-Widder method

This program applies the rho-algorithm to extrapolate the Post-Widder sequence. An odd number of terms are required to be calculated.

Entry/Exit Parameters.

1. m — integer variable. In the example provided below if m=1 then the inverse transform of $\bar{f}(s) = 1/(s+1)$ is found while m=2 will give the inverse transform for $\bar{f}(s) = 1/\sqrt{(s^2+1)}$.
2. n — integer variable. The number of terms to be used in the rho algorithm. n must be odd and a maximum of 45.
3. t — **real** variable. The value of t at which the inverse transform is required.
4. rho2 — a **real** array. The contents of the array are the n terms of the Post-Widder sequence. The routine which calculates the sequence must be provided by the user.

Each cycle of the rho-algorithm will be printed out and the final result printed will give the estimate for $f(t)$. The accuracy of the estimate can be gauged by comparing with the alternate cycle which precedes it.

```
      real rho1(45),rho2(45),a,b,c,d
      read(5,*)m
c     m indicates the sequence to be computed
c     m=1, f̄(s) = 1/(s+1);   m=2, f̄(s) = 1/(s^2+1)^1/2
      read(5,*)n
c     n must be odd, n≤ 45
      call rhoalg(m, n)
      stop
      end

      subroutine rhoalg(m, n)
      real rho1(45), rho2(45), a
      do i=1, n
      rho1(i)=0
      enddo
      if (m.eq.1) goto 50
      call sequence2(n, rho2)
```

```
      goto 100
50    call sequence1(n, rho2)
100   do k=1, n-1
      do i=1, n-k
      rho1(i)=rho1(i+1)+float(k)/(rho2(i+1)-rho2(i))
      write(6,*)rho1(i)
      enddo
      write(6,*)'new cycle'
      do i=1, n-k
      a=rho1(i)
      rho1(i)=rho2(i)
      rho2(i)=a
      enddo
      enddo
      return
      end

      subroutine sequence1(n, x)
      real x(45), t, a, b
      read(5,*)t
      do i=1, n
      a=float(i)
      b=-(a+1)
      x(i)=(1+(t/a))**b
      write(6,*)x(i)
      enddo
      return
      end

      subroutine sequence2(n, x)
      real x(45), u(0:45), t, a, b, f, s, v, w
      v=-1.0
      w=1.0
      read(5,*)t
      do i=1, n
      a=float(i)
      b=a+1
      s=a/t
      w=w*v
```

```
      x(i)=w*(s)**b
      call fac(i,f)
      x(i)=x(i)/f
      call deriv(i,s,u)
      x(i)=x(i)*u(i)
      write(6,*)x(i)
      enddo
      return
      end

      subroutine fac(i, f)
      real f
      f=1.0
      do j=1, i
      f=j*f
      enddo
      return
      end

      subroutine deriv(i, s, u)
      real r, s, t, u(0:45)
      t=s*s+1
      u(0)=1.0/sqrt(t)
      u(1)=-(s/t)*u(0)
      if (i.eq.1) goto 100
      do j=2, i
      r=float(j)
      u(j)=(2*r-1)*s*u(j-1)+(r-1)*(r-1)*u(j-2)
      u(j)=-u(j)/t
      enddo
100   return
      end
```

Program for the Honig-Hirdes method for Laplace Inversion

This program is based on the Durbin approach to evaluating the inverse Laplace transform but differs from the Crump method by applying the Korrektur technique of Albrecht and Honig to reduce the discretization error.

Entry/Exit Parameters.

1. f — a **real** subroutine which evaluates $\bar{f}(s)$ given **real** parameters sr($=\Re s$) and si($=\Im s$) the results being indicated by the **real** parameters fr($=\Re\bar{f}(s)$) and fi($=\Im\bar{f}(s)$). This subroutine must be declared as external and supplied by the user.
2. t1, tn — **real** parameters which are respectively lower and upper bounds of the interval in which $f(t)$ is to be approximated.
3. n — integer variable, the number of values $t_k,\ k = 1, \cdots, n$ at which $f(t)$ is to be computed. Note that

$$t_k = \text{t1} + (\text{tn} - \text{t1})k/(n+1),$$

so that if we require $t_k = 10 + k, \quad k = 1, \cdots, 10$ we have to set t1=10 and tn=21.

4. iman — integer variable. If it is set to 0 all further parameters are set automatically (except ns1 which has to be 60). If it is 1 a manual choice of the parameters ilapin, ikonv, ns1, ns2, icon, ikor, con is possible. In the example below iman=0.
5. ilapin — integer variable. If ilapin =1 the approximation formula is applied at $T = t$ while if ilapin =2 it is applied at T=tn (or 2tn for the Korrektur terms).
6. ikonv — integer. If ikonv=1 the minimum-maximum method is invoked while ikonv=2 invokes the ϵ - algorithm.
7. ns1 — integer. The number of function evaluations of $\bar{f}(s)$ which are used in order to estimate $f(t)$. ns1=60 if iman=0.
8. ns2 —integer. The number of function evaluations of $\bar{f}(s)$ used to approximate the Korrektur term.
9. icon — integer. If icon=0 then there is no optimal choice of free parameters. If icon =1 then there is optimal choice of free parameters for $t = t_{[n/2]}$ while if icon=2 there is optimal choice of free parameters for $t = t_k$.
10. ikor — integer. If ikor =0 there is no application of the Korrektur method otherwise it is 1.

11. con — **real**. If icon =0 then the free parameter c is determined from con = cT.
12. h(6,n) — a **real** array which on exit contains $f(t_k)$ in row 1. The second row is a work area. The third row contains the computed optimal parameter $\mathrm{con}_{\mathrm{opt}} = c_{\mathrm{opt}} \cdot T$ for $t = t_k$. The fourth row gives the code for the acceleration method adopted (0 — no acceleration of convergence, 1 — minimax, 2 — epsilon algorithm(epal), 4 — curve fitting method(cfm), P = ns1+2 — indicates overflow in epal if P$\geq$4. The fifth row contains the absolute error calculated by formula (33) in the Honig - Hirdes paper and the sixth row contains a control number of up to 6 digits which is explained in Honig and Hirdes [6].
13. E(3,ns1) — a **real** array used as working space.
14. iout — integer. Refers to the output device used.
15. ier — integer. Parameter which indicates error in the input data. If ier = 0 there is no error. Otherwise the error is indicated by the print-out.

```
      integer n, iman, ilapin, ikonv, ns1, ns2, icon, ier, iout
      parameter (n=10, ns1=60, iman=0)
      real t1, tn, con, h(6,n), e(3,ns1)
      external f
      read(5,*)t1, tn
      iout=6
      call lapin(f, t1, tn, n, iman, ilapin, ikonv, ns1, ns2, icon, ikor,
     + con, h, e, ier, iout)
      do i=1,n
      t=t1+(tn-t1)*i/(n+1)
      write(iout,*)t, h(1,n)
      enddo
      stop
      end

      subroutine lapin(f, t1, tn, n, iman, ilapin, ikonv, ns1, ns2, icon,
     + ikor, con, h, e, ier, iout)

      real abrn, absf, con, conopt, con1, con2, del, e, fn, fns1
      real freal, fimag, h, pi, racc, rnsum, rnsumk, t, ta, tb
      real tk, tn, t0, t1, v1, v2, w
```

```
      integer hmono
      external f
      dimension h(6,n), e(3,ns1)
      common /clapin/ ta, tb, t0, conopt, absf, lval, hmono

c     initialise array h
      do i=1, n
      do j=1, 6
      h(j,i)=0.0
      enddo
      enddo

      ier=0
      if (tn.lt.t1) ier=1
      if (n.lt.1) ier=ier+10
      if (iman.eq.0) goto 100
      if (iman.eq.1) goto 110
      ier=ier+100
      goto 120

c     parameters for iman=0
100   if (ns1.ne.60) ier=ier+100000
      if (ier.ne.0) goto 120
      ilapin=1
      ikonv=2
      icon=1
      ikor=0
      ns2=0
      goto 200

c     iman=1
110   if (ilapin.lt.1 .or. ilapin.gt.2) ier=ier+1000
      if (ikonv.lt.1 .or. ikonv.gt.2) ier=ier+10000
      if (ns1.lt.1 .or. (ns2.lt.1 .and. ikor.eq.1)) ier=ier+100000
      if (icon.lt.0 .or. icon.gt.2 .or. (icon.eq.2 .and. ilapin.eq.2))
     + ier=ier+1000000
      if (ikor.lt.0 .or. ikor.gt.1) ier=ier+10000000
      if (icon.eq.0 .and. con.le.0.0) ier=ier+100000000
      if (ier.eq.0)goto 200
```

```
120     call error(ier, t1, tn, n, iman, ilapin, ikonv, ns1, ns2, icon,
     +  ikor, con, iout)
        return

200     pi=4.0*atan(1.0)
        con1=20.0
        con2=con1- 2.0
        absf=0.0

        j3=(3-icon)/2 + n*(icon/2)
        ta=t1
        tb=tn
        do 830 l3=1, j3
        lval=l3
        hmono=0
        kor1=ikor

c       computation of the optimal parameters
210     if (icon-1) 215,230,220
215     jump=0
        call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
        goto 830
220     ta=t1+float(l3)*(tn-t1)/float(n+1)
        tb=ta
230     nh=n/2
        t=ta+(tb-ta)*float(nh)/float(n+1)
        tk=float(2-ilapin)*t+float(ilapin-1)*tb

c       computation of the truncation error (rnsum)
        t0=t
        con =con1
        jump=1
        call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
240     fn=h(1,13)
        fns1=e(1,ns1)
        con=con2
        jump=2
        call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
250     if (fn.ne.h(1,l3) .and. fns1.ne.e(1,ns1))goto 255
```

```
      conopt=con
      absf=0.0
      goto 320
255   rnsum=tk*(fn-h(1,l3))/exp(con1)-exp(con2)
      if (ilapin.eq.2)goto 260

c     computation of the acceleration factor (del)
      racc=t*(fns1-e(1,ns1))/( exp(con1)-exp(con2))
      del=rnsum/racc

260   if (ikor.eq.1)goto 280

c     optimal parameters (Method A)
      t0=2.0*tk+t
      con=con1/4.0
      jump=3
      call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
270   fn=h(1,l3)
      conopt=-tk/(2.0*tk+t)*log( abs(rnsum/(tk*fn)))
      goto 310

c     optimal parameters for the Korrektur method (Method A)
280   t=4.0*tk+t
      con=con1/4.0
      jump=4
      call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
290   fn=h(1,l3)
      t0=8.0*tk+t
      jump=5
      call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
300   fn=fn-h(1,l3)
      conopt=-tk/( 4.0*tk+t)*log( abs(rnsum/(tk*fn)))

c     optimal parameters (Method B)
310   if (ilapin.eq.1)goto 315
      absf=abs( exp(conopt)*rnsum*2.0/tk)
      goto 320
315   v1=con1/t
      v2=conopt/t
```

```
      w=pi*float(ns1)/t
      call f(v2, w, freal, fimag)
      rnsumk=rnsumk*freal
      call f(v1, w, freal, fimag)
      rnsumk=rnsumk/freal
      abrn=(rnsumk-rnsum)/(v2-v1)
      conopt=-log( abs(abrn/t+rnsumk)/(t*float(ikor*2+2)*fn)))/
     +  float(3+2*ikor)
      v1=v2
      v2=conopt/t
      call f(v2, w, freal, fimag)
      rnsumk=rnsumk/freal
      absf=exp(conopt)/t*abs(rnsumk)+
     +  abs(exp(-2.0*conopt)*fn*float(ikor-1)
     +  exp(-4.0*conopt)*fn*float(ikor))

320   if (conopt.le.0.0) conopt=1.0
      jump=6
      call lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor, con, h, e, jump)
830   continue
      return
      end

      subroutine lapin2(f, t1, tn, n, ilapin, ikonv, ns1, ns2, icon, ikor,
     +  con, h, e, jump)
c     Laplace inversion with optimal parameters

      real a, absf, b, con, conopt, delta, divi, e, e1, e2, e3, eins
      real faktor, fimag, freal, h, pi, pit, pite, ral, suim, sure
      real ta, tb, te, tl, tm, tm1, tn, tt, t0, t1, x1, x2, x3, v, w, y1, y2, y3
      integer richt, richta, hmono
      external f
      dimension h(6,n), e(3,ns1), e3(3)
      common /clapin/ ta, tb, t0, conopt, absf, l3, hmono

      pi=4.0*atan( 1.0)
      if (jump.eq.0) goto 360
      if (jump.lt.6) goto 370
```

```
        t0=ta
        tt=tb
        i1=l3
        j1=(2-icon)*n+(icon-1)*l3
        con=conopt
        kor1=ikor
        goto 380

360     t0=t1
        tt=tn
        i1=1
        j1=n
        kor1=ikor
        jump=6
        goto 380
370     kor1=0
        tt=t0
        i1=l3
        j1=l3

380     delta=(tt-t0)/float(n+1)
        nsum=ns1

c       computation of the t-values from t0, tt
        do 805 k2=1, 2
        if (ilapin.eq.1) goto 420
        te=float(k2)*tt
        v=con/te
        call f(v, 0.0, freal, fimag)
        ral=-0.5*freal
        pite=pi/te

405     do l=1, nsum
        w=float(l-1)*pite
        call f(v, w, freal, fimag)
        e(2,l)=freal
        e(3,l)=fimag
        enddo
```

```
420     do 800 k1=i1, j1

        tl=t0+float(k1)*delta
        if (ilapin.eq.2) goto 440

        if (k2.eq.2) tl=3.0*tl
        v=con/tl
        faktor=exp(v*tl)/tl
        call f(v,0.0,freal,fimag)
        ral=-0.5*freal
        pit=pi/tl
        eins=1.0
        sure=0.0

c       method of Durbin
425     do l=1, nsum
        w=float(l-1)*pit
        call f(v, w, freal, fimag)
        sure=sure+freal*eins
        eins=-eins
        e(1,l)=faktor*(ral+sure)
        enddo
        goto 460

440     if (k2.eq.2) tl=tl+te
        faktor=exp(v*tl)/te
        sure=0.0
        suim=0.0
        do l=1, nsum
        w=float(l-1)*pite
        sure=sure+e(2,l)*cos(w*tl)
        suim=suim+e(3,l)*sin(w*tl)
        e(1,l)=faktor*(ral+sure-suim)
        enddo

c       search for stationary values
460     nmax=nsum*2/3
        monoto=0
        k=0
```

```
      richta=sign( 1.5,(e(1,nsum)-e(1,nsum-1)))
      do l=1, nmax
      j=nsum-1
      richt=sign( 1.5,e(1,j)-e(1,j-1)))
      if (richt.eq.richta) goto 500
      k=k+1
      e3(k)=e(1,j)
      richta=richt
      if (k.eq.3) goto 510
500   enddo
      if (k.eq.0) goto 700
      h(k2, k1)=e(1, nsum)
      if (k2.eq.1) h(4, k1)=0
      goto 790
510   ke=2
      if ((e3(ke)-e(1,j))*float(richta).gt.0) goto 560
      jmin=nsum/3
      jmax=j-1
      do jj=jmin, jmax
      j=j-1
      richt=sign(1.5, (e(1, j)-e(1, j-1)))
      if (richt.eq.richta) goto 540
      richta=richt
      ke=3-ke
      if ((e3(ke)-e(1, j))*float(richta).gt.0.0)goto 560
540   enddo
550   monoto=1
      if (ikonv.eq.2) goto 630

c     minimum-maximum method (minimax)
560   h(k2, k1)=(e3(1)+e3(3))/4.0+e3(2)/2.0
      if (k2.eq.1) h(4, k1)=1
      goto 790

c     epsilon algorithm (epal)
630   k=0
      nsumm1=nsum-1
      e2=e(1, 1)
      do l=1, nsumm1
```

```
      e1=e(1,1)
      tm=0.0
      lp1=l+1
      do m=1, l
      mm=lp1-m
      tm1=e(1, mm)
      divi=e(1, mm+1)-e(1, mm)
      if (abs(divi).gt.1.0**-20) goto 640
      k=1
      goto 670
640   e(1, mm)=tm+1.0/divi
      tm=tm1
      enddo
      e2=e1
      enddo

670   if ( abs(e1).gt.abs(e2) e1=e2
      if ( abs(e(1,1)).gt.abs(e1)) e(1, 1)=e1
      h(k2, k1)=e(1, 1)
      if (k2.eq.1) h(4, k1)=k+2
      goto 790

c     curve fitting (cfm)
700   x1=float(nsum)-2.0
      x2=float(nsum)-1.0
      x3=float(nsum)-0.0
      y1=e(1, nsum-2)
      y2=e(1, nsum-1)
      y3=e(1, nsum)
      b=((y3-y1)*x3*x3*(x1+x2)/(x1-x3)
     + -(y2-y1)*x2*x2*(x1+x3)/(x1-x2))/(x3-x2)
      a=((y2-y1)-b*(x1-x2)/(x1*x2))*(x2*x2*x1*x1)/(x1*x1-x2*x2)
      h(k2,k1)=y1-(a/x1+b)/x1
      monoto=1
      if (k2.eq.1) h(4, k1)=3

790   h(3, k1)=con
      h(5, k1)=absf
      hmono=hmono+k2*monoto*10**(6-jump)
```

```
      if (jump.lt.6) goto 800
      h(6, k1)=float((2-k2)*hmono+
     + float(k2-1)*(float(2*monoto)+h(6, k1))
      hmono=hmono/10*10

800   continue
      if (kor1.eq.0) return
      if (k2.eq.2) goto 810
      nsum=ns2
805   continue

c     Korrektur method
810   faktor=-exp(-2.0*con)
      do k=i1, j1
      h(1, k)=h(1, k)+faktor*h(2, k)
      enddo

      return
      end

      subroutine error(ier, t1, tn, n, iman, ilapin, ikonv, ns1, ns2, icon, ikor,
     * con, iout)
      real t1, tn, con
      write(iout,1)ier, t1, tn, n, iman, ilapin, ikonv, ns1, ns2, icon, ikor, con
1     format(///14h *** error ***, 8x, 5hier =, i12//
     * 10h t1 * , d10.3, 18x, 12h1 * tn < t1/
     * 10h tn * , d10.3/
     * 10h n * , i10, 17x, 11h10 * n < 1/
     * 10h iman * , i10, 16x, 29h100 * iman < 0 or iman > 1/
     * 10h ilapin * , i10, 15x, 34h1000 * ilapin < 1 or ilapin > 2/
     * 10h ikonv * , i10, 15x, 33h10000 * ikonv < 1 or ikonv > 2/
     * 10h ns1 * , i10, 13x, 38h100000 * ns1 < 1 or (ns2 < 1 and ,
     * 41hikor = 1) or (ns1 <> 60 and iman = 0)/
     * 10h ns2 * , i10/
     * 10h icon * , i10, 12x, 36h1000000 * icon < 0 or icon > 2 or,
     * 29h (icon = 2 and ilapin = 2)/
     * 10h ikor * , i10, 11x, 34h10000000 * ikor < 0 or ikor > 1/
     * 10h con * , d10.3, 10x, 21h100000000 * con <= 0//)
      return
```

```
      end

      subroutine f(sr, si, fr, fi)
c     evaluates function 1/(s^2 + 1)
      real sr, si, fr, fi, a
      fr=sr*sr-si*si+1.0
      fi=2*sr*si
      a=fr*fr+fi*fi
      fr=fr/a
      fi=-fi/a
      return
      end
```

Program for Talbot's method

This is a sub-program of Algorithm 682 by Murli and Rizzardi [8]. It omits a section of the Murli-Rizzardi program which deals with essential singularities. Readers should be warned that some of the variable names used by Murli and Rizzardi have been changed to be more consistent with the values used in the text and in the original Talbot paper. Examples are conlam, consig, and connu and tvalue which have become lam, sig, nu and tval.

Entry/Exit parameters

1. flap — a **complex** subroutine which evaluates $\bar{f}(s)$ for complex arguments. This subroutine must be supplied by the reader.
2. finv — a **real** function which evaluates the function $f(t)$ at the given data points for some known test function. This will give a measure of confidence when operating the program with $f(t)$ unknown. In this situation take $f(t)$ equal to **1.0**.
3. sings — this routine must return the number of singularities, their location and multiplicity. There is a restriction as the number of singularities may not exceed 24.
4. nt — integer. The number of data points at which the Laplace transform is required.
5. vt — **real** array of dimension nt. Data to be supplied by the user. In the example below data values are $\mathbf{0.5}, \cdots, \mathbf{64.0}$.
6. nosing — integer. The number of singularities of the function $\bar{f}(s)$ with non-negative imaginary parts.
7. singre — real array of dimension ≤ 24. This array contains the real parts of the singularities.
8. singim — real array of dimension ≤ 24. This array contains the imaginary parts of the singularities.
9. multsi — integer array. This array records the multiplicity of each singularity.

For example, if $\bar{f}(s) = 1/(s-1)^3[(s-0.5)^2+4]$, we would set
nosing=3,
singre(1)=1, singim(1)=0, multsi(1)=2;
singre(2)=0.5, singim(2)=2,multsi(2)=0;
singre(3)=0.5, singim(3)=-2, multsi(3)=0;
in the subroutine sings.

10. invf — **real**. The computed value of $f(t)$.
11. nopts — integer. The number of points used in the Talbot summation as computed by the subroutine tapar.

12. ic — integer. The machine decimal precision used by Talbot's method.
12a. id — an integer between 1 and ic+2.
13. ier — integer . This is an error indicator. ier=0 indicates normal termination. ier =1 indicates that the output value of the subroutine finv causes overflow and a scaled value of $f(t)$ is returned.
14. pserr — real. This is the ratio of absolute error/$(e^{\sigma t})$.
15. error — real. This is the relative (R) error when the value produced by finv is greater than 1. Otherwise it is the absolute error (A).

```
      implicit real(a-h, o-z)
      real lam, nu, invf
      real finv, fv, tt, error, pserr, dinvf, demax, d2mach
      real r1mach(3), r1
c     the machine constants r1mach(*) are defined by Murli and Rizzardi
c     — this program gives the IEEE 754 specification for these constants
      external flap
      complex flap
      character*1 alfa
      dimension singre(24), singim(24), multsi(24), vt(8)
      data ndim/24/
      data vt/0.5, 1.0, 2.0, 4.0, 8.0, 16.0, 32.0, 64.0/
      data d2mach/Z'7fefffffffffffff'/
c     d2mach is the double length version of r1mach(2)
      data r1mach(2)/Z'7f7fffff'/
      data r1mach(3)/Z'33800000'/
      nt=8
      ic=-ifix(log10(r1mach(3))*3./4.)
      demax=log(d2mach)
      emax=log(r1mach(2))
      call sings(ndim, singre, singim, multsi, nosing)
      write(6,200)(singre(k), singim(k), multsi(k), k=1, nosing)
      do 2 it=1, nt
      tval=vt(it)
      tt=dble(tval)
      if (tval.gt.demax) then
          fv=0.0
```

```
      else
           fv=finv(tval)
      endif
      if (abs(fv).gt.1.0) then
           alfa='R'
      else
           alfa='A'
      endif

      do 3 id=1, ic+2
      call tapar(tval, id, nosing, singre, singim, multsi, ic,
     *     lam, sig, nu, nopts)
      if (id.eq.1) then
           if (tval.gt.demax) then
                write(6,*)tval, lam, sig, nu
      else
           write(6,*)tval, lam, sig, nu, fv
           endif
      endif

      call tsum(flap, lam, sig, nu, nopts, tval, invf, ier)
      if (ier.eq.0) then
           pserr=(fv-invf)/exp(tt*sig)
           if (alfa.eq.'R') then
                error=(fv-invf)/fv
      else
           error=fv-invf
      endif
      write(6,*)id, nopts, invf, pserr, error, alfa
      else
           write(6,*)id, nopts, invf, ier
      if (tval.gt.emax.and.tval.lt.demax) then
           dinvf=lam/nopts*exp(tt*sig)*invf
           pserr=(fv-dinvf)/exp(tt*sig)
           error=(fv-dinvf)/fv
           write(6,*)dinvf, pserr, error, alfa
      endif
      endif
3     continue
```

```
2       continue
        stop
200     format('singularities and their multiplicities'/
     *  (2(2x, e12.6), 2x, I2))
        end

        subroutine tapar(tval, decdig, nosing, singre, singim, multsi, ic,
     *  lam, sig, nu, nopts)
c       This subroutine evaluates the geometrical parameters
c       lam(bda), sig(ma) and nu for the Talbot contour and the
c       accuracy parameter (nopts)
        implicit real (a-h, o-z)
        real lam,nu
        real pi
        logical caso1
        integer ic, decdig, dd
        dimension singre(nosing), singim(nosing), multsi(nosing)

        data pi/0.0/
        if (pi.eq.0.0) pi=4.*atan(1.0)
        caso1=.true.
        omega=0.4*(ic+1)

        sig0=0.0
        pmax=singre(1)
        kmax=1
        if (nosing.eq.1) goto 3
        do j=1, nosing
        if (singre(j).gt.pmax) pmax=singre(j)
        if (multsi(j).gt.multsi(kmax)) kmax=j
        enddo
3       if (pmax.gt.sig0) sig0=pmax

        rd=0.0
        sid=0.0
        tetd=pi
        do 7 j=1, nosing
        if (singim(j).le.0.0) goto 7
        caso1=.false.
```

```
      etaj=atan2(-singre(j)+sig0,singim(j))
      tetj=etaj+pi/2.
      rj=singim(j)/tetj
      if (rd.ge.rj) goto 7
      rd=rj
      tetd=tetj
      id=j
7     continue

      if (caso1) goto 8
      srd=singre(id)-sig0
      sid=singim(id)
      md=multsi(id)
      v=sid*tval
      omega=min(omega+v/2.0,2.0*(ic+1)/3.0)
      if (1.8*v.gt.omega*tetd) goto 9

      caso1=.true.
8     lam=omega/tval
      sig=sig0
      nu=1.0
      goto 10
9     caso1=.false.
      pk=1.6+12.0/(v+25.0)
      fi=1.05+1050./max(553.0, 800.0-v)
      pmu=(omega/tval+sig0-pmax)/(pk/fi-1.0/tan(fi))
      lam=pk*pmu/fi
      sig=pmax-pmu/tan(fi)
      nu=sid/pmu

10    n1=0
      e=( 2.3*decdig+omega)/(lam+tval)
      if (e.gt.4.4) goto 11
      unro=( 16.0+4.3*e)/(24.8-2.5*e)
      goto 13
11    if (e.gt.10.0) goto 12
      unro=(50.0+3.0*e)/(129.0/e-4.0)
      goto 13
12    unro=(44.0+19.0*e)/(256.0/e+0.4)
```

```
13      n1=lam*tval*(unro+(nu-1.0)/2.0)
        n1=n1+1
        n0=0
        n2=0

        if (sid.eq.0.0) goto 15
        dd=decdig+2*min(md-1,1)+md/4
        if (.not.caso1.or.(md.eq.0)) goto 14
        p=srd/lam
        q=sid/lam
        call invpc(p, q, tetd, u, pi)
        if (u.le.0.0)goto 14
        n0=(2.3*dd+srd*tval)/u
        n0=n0+1
14      gamm=(sig-sig0)/lam
        y=v/1000.0
        eta=min(1.78,1.236+0.0064*(1.78**dd))
        eta=eta*(1.09-y*(0.92-0.8*y))
        n2=eta*nu*(2.3*dd+omega)/(3.0+4.0*gamm+1.0/exp(gamm))
        n2=n2+1
        nopts=max(n0, n1)
        nopts=max(n2, nopts)
        return

15      dd=decdig+2*min(multsi(kmax)-1,1)+multsi(kmax)/4
        n2=(2.3*dd+omega)/2.0
        n2=n2+1
        icm1=ic-1
        if (dd.lt.icm1) goto 17
        q=0.0
        do 16 l=1, nosing
        if (multsi(l).eq.0) goto 16
        dd=decdig+2*min(multsi(l)-1, 1)+multsi(l)/4
        srd=singre(l)-sig0
        if (.not.(dd.ge.icm1.and.srd.lt.0.0)) goto 16
        p=srd/lam
        call invpc(p, q, tetd, u, pi)
        if (u.le.0.0)goto 16
        n0l=(2.3*dd+tval*srd)/u
```

```
      if (n0l.gt.n0) n0=n0l
16    continue
      n0=n0+1
17    nopts=max(n0, n1)
      nopts=max(n2, nopts)
      return
      end

      subroutine tsum(flap, lam, sig, nu, nopts, tval, finv, ier)
c     This routine gives the trapezoidal rule approximation
c     for the inverse Laplace transform
      implicit real (a-h, o-z)
      real lam, nu
      real pi
      external flap
      complex flap, s, ff
      real r1mach(2), r1
      data r1mach(1)/Z'00800000'/
      data r1mach(2)/Z'7f7fffff'/
      data pi/0.0/
      elim=log(r1mach(1))
      if (pi.eq.0.0) pi=4.0*atan(1.0)
      piovn=pi/float(nopts)
      ier=0
      tau=lam*tval
      psi=piovn*tau*nu
      c=cos(psi)
      br=0
      bi=0
      dbr=0
      dbi=0
      nm1=nopts-1
      finv=0
      if (nm1.eq.0) goto 5

c     Reinsch algorithm
      if (c.gt.0) goto 2
c     Case cos(psi)≤ 0
      u=4.0*cos(psi/2)**2
```

```
      do k=nm1, 1, -1
      theta=k*piovn
      alpha=theta*cos(theta)/sin(theta)
      beta=theta+alpha*(alpha-1.0)/theta
      sr=lam*alpha+sig
      si=lam*nu*theta
      s=cmplx(sr, si)
      ff=flap(s)
      g=real(ff)
      h=imag(ff)
      arg=alpha*tau
      if (arg.le.elim) then
           eat=0.0
      else
           eat=exp(arg)
      endif
      br=dbr-br
      bi=dbi-bi
      dbr=u*br-dbr+eat*(g*nu-h*beta)
      dbi=u*bi-dbi+eat*(h*nu+g*beta)
      enddo
      br=dbr-br
      bi=dbi-bi
      dbr=u*br-dbr
      goto 4

c     Case cos(psi)>0
2     u=-4.0*sin(psi/2)**2
      do k=nm1, 1, -1
      theta=k*piovn
      alpha=theta*cos(theta)/sin(theta)
      beta=theta+alpha*(alpha-1.0)/theta
      sr=lam*alpha+sig
      si=lam*nu*theta
      s=cmplx(sr, si)
      ff=flap(s)
      g=real(ff)
      h=imag(ff)
      arg=alpha*tau
```

```
      if (arg.le.elim) then
           eat=0.0
      else
           eat=exp(arg)
      endif
      br=dbr+br
      bi=dbi+bi
      dbr=u*br+dbr+eat*(g*nu-h*beta)
      dbi=u*bi+dbi+eat*(h*nu+g*beta)
      enddo
      br=dbr+br
      bi=dbi+bi
      dbr=u*br+dbr
c     end of Reinsch algorithm

4     finv=dbr-br*u/2.0-bi*sin(psi)
5     sr=lam+sig
      s=cmplx(sr)
      ff=flap(s)
      g=real(ff)
      finv=(finv+nu*exp(tau)*g/2.0)
      if (finv.eq.0.0) return
      if (finv.gt.0.0) then
           signum=1.0
      else
           signum=-1.0
      endif

      cost=log(r1mach(2))
      of=sig*tval+log(lam*abs(finv)/nopts)
      if (of.gt.cost) then
      ier=1
      write(6,*)'ier=', ier
      else
           finv=exp(of)*signum
      endif
      return
      end
```

```
      subroutine invpc(p, q, theta, u, pi)
      implicit real (a-h, o-z)
      r=p*p+q*q
      r=sqrt(r)
      x=13.0/(5.0-2*p-q-0.45*exp(p))
      y=2*pi-x
      k=1
100   continue
      ang=y-theta
      s=sin(ang)
      c=cos(ang)
      b=q-y
      e=-b/s
      arg=e/r
      if (arg.le.0.0) then
      u=-1.0
      goto 200
      endif
      u=log(arg)
      g=p+b*c/s+u
      dy=b*g/(1.0+e*(e-2.0*c))
      if (abs(dy).lt.0.0001) goto 200
      if (k.ge.100) goto 200
      k=k+1
      y=y+dy
      goto 100
200   continue
      return
      end

      complex function flap(s)
      complex s, a, b, I
      flap=log(s)/s
      return
      end

      subroutine sings(ndim, singre, singim, multsi, nosing)
      implicit real (a-h, o-z)
      dimension singre(ndim), singim(ndim), multsi(ndim)
```

```
nosing=1
singre(1)=0.0
singim(1)=0.0
multsi(1)=0
return
end

real function finv(t)
real a
write(6,*)'t=', t
a=dble(t)
finv=-0.57721566490153286-log(a)
return
end
```

Mathematica program for Talbot's method
This program is taken from Abate and Valkó [1]. We reproduce this program and give the results when $\bar{f}(s) = 1/(s^3 - 8)$ in Chapter 10.

```
FT[F_ , t_ , M_]:=Module[{np, r, S, theta, sigma},
np=Max[M, $ MachinePrecision]; r=SetPrecision[2M/(5t), np];
S=r theta (Cot[theta]+I);
sigma=theta+(theta Cot[theta]-1)Cot[theta];
(r/M) Plus@@Append[Table[Re[Exp[tS](1+I sigma) F[S] ],
     {theta,  Pi/M,  (M-1)Pi/M,  Pi/M}],
(1/2) Exp[rt] F[r]] ]
```

When $\bar{f}(s) = 1/(s^3 - 8)$ and we require the inverse transform for $t = 0.5,\ 1,\ \cdots,\ 2^6$ we need to implement the above with the instruction

```
Do[Print[FT[1/(#^3-8) &, 2^k,  300]], {k, -1, 6}]
```

Mathematica program for Gaver's method
Abate and Valkó give a web reference for their GWR (Gaver Wynn rho) algorithm. The program below was written independently by my colleague Dr. R. Behrend.

```
F[f_ , n_ , t_ , M_]:=Module[np, a, np=Max[M, $MachinePrecision];
a=SetPrecision[Log[2]/t, np];
(2n)!/(n!(n-1)!) a Sum[(-1)^k Binomial[n, k] f[(k+n)a],  {k, 0, n}]]
R[f_ , i_ , 0, t_ , M_]:=R[f, i, 0, t, M]=0;
R[f_ , i_, 1, t_ , M_]:=R[f, i, 1, t, M]=F[f, i, t, M];
R[f_ , i_ , k_ , t_ , M_]:=R[f, i, k, t, M]
     =R[f, i+1, k-2, t, M] +(k-1)/(R[f, i+1, k-1, t, M]-R[f, i, k-1, t, M])
```

Thus if we wanted to evaluate the inverse transform of $1/(s^2 + 1)$ for $t = 0.5, 1, \cdots, 2^6$ we need to implement the above with the instruction

$$\text{Do}[\text{Print}[\text{R}[\tfrac{1}{\#^2+1}\ \&,\ 1,\ 75,\ 2^k,\ 300]],\ \{\text{k},\ \text{-1},\ 6\}\ ]$$

Program for Application 1
Gaver's method

```
c       Uses Gaver method in conjunction with the ρ-algorithm
c       Requires input of odd integer n ≤ 45
c       Requires input of variable t
c       Requires a function subroutine funct(s,f) to
c       evaluate the Laplace transform f̄(s) for real s.

        integer n
        read(5,*)n
        call gaver(n)
        stop
        end

        subroutine gaver(n)
        real rho1(45), rho2(45), a
        m=n/2
        if (n.eq.2*m) goto 100
        do i=1, n
        rho1(i)=0
        enddo
        call sequence (n, rho2)
        do k=1, n-1
        do i=1, n-k
        rho1(i) =rho1(i+1)+ float(k)/(rho2(i+1)-rho2(i))
        write(6,*)rho1(i)
        enddo
        write(6,*)'new cycle'
        do i=1, n-k
        a=rho1(i)
        rho1(i)=rho2(i)
        rho2(i)=a
        enddo
        enddo
        goto 200
100     write(6,*)'Error,  n not odd'
200     return
```

```
      end

      subroutine sequence(n, x)
      real x(45), u(0:45,0:90), t, a, b, c, s, f, v, w
      read(5,*)t
      a=log(2.0)
      a=a/t
      m=2*n
      do k=1, m
      s=k*a
      call funct(s, f)
      u(0,k)=k*a*f
      enddo
      do k=1, n
      do i=1, k
      do j=k, 2*k-i
      b=float(j)/float(i)
      u(i, j)=(1+b)*u(i-1, j)-b*u(i-1, j+1)
      enddo
      enddo
      x(k)=u(k, k)
      write(6,*)x(k)
      enddo
      return
      end

      subroutine funct(s, f)
c     Evaluation of this function requires the
c     subroutine newton(s, z)
      real s, f, z
      f=1.0/s
      call newton(s, z)
      f=-f/(1.0-z)
      return
      end

      subroutine newton(s, z)
      real x, s, t, z, y0, y1
      x=(s+4)/3
```

```
100     y0=x*x*x-((s+4)/3)*x*x+(1.0/3.0)
        y1=3*x*x-2*((s+4)/3)*x
        z=-y0/y1
        if (abs(z).lt.1.^-16) goto 200
        x=x+z
        goto 100
200     z=x+z
        return
        end
```

Bibliography

[1] Abate, J. and Valkó, P.P. Multi-precision Laplace transform inversion, Int. J. Num. Meth. Eng., **60** (2004) 979-993.

[2] Crump, K.S. Numerical Inversion of Laplace Transforms using a Fourier Series Approximation, J. Assoc. Comput. Mach., **23** (1976) 89-96.

[3] Ford, W.F. and Sidi, A. An algorithm for the generalization of the Richardson process, SIAM J. Numer. Anal., **24** (1987) 1212-1232.

[4] Garbow, B.S., Giunta, G., Lyness, J.N. and Murli, A. Software for an implementation of Weeks' method for the Inverse Laplace Transform, ACM Trans. Math. Software (TOMS), **14** (1988) 163-170.

[5] Graf, Urs *Applied Laplace Transforms and z-Transforms for Scientists and Engineers*, Birkhäuser Verlag, Basel, (2004).

[6] Honig, G. and Hirdes, U. A method for the Numerical Inversion of Laplace Transforms, J. Comput. Appl. Math., **10** (1984) 113-132.

[7] Longman, I.M. Computation of the Padé Table, Intern. J. Comp. Math., **B3** (1971) 53-64.

[8] Murli, A. and Rizzardi, M. Algorithm 682: Talbot's method for the Laplace inversion problem, ACM Trans. Math. Softw. (TOMS), **16** (1990) 158-168.

[9] www.nag.co.uk

[10] Sidi, A. *Practical Extrapolation Methods: Theory and Applications*, Cambridge University Press, Cambridge, (2003).