Computer Algebra Math0070

Daniel Richardson,
Department of Mathematical Sciences,
University of Bath
email:dsr@maths.bath.ac.uk

May 15, 2001

1 Introduction: Computer Algebra

To see the latest version of these notes, from BUCS, in unix command:

```
xdvi ~masdr/public_html/m70
```

Computer algebra is the use of computers to represent mathematical objects and to compute with them. See *Computer Algebra*, J Davenport for discussion. The mathematical objects include numbers, but also polynomials, sets, differential equations , etc. Examples of computer algebra systems are maple, reduce, axiom. We will start with maple.

1.1 Maple

There are a number of books in the library which are intended to help with use of Maple. See (518.542MAP).

From BUCS network in unix, command

xmaple

and then choose help and do the introductory tour. Try some commands.

Example 1 *You might try the following. (Comments in parentheses.)*

1/3+2/7; (calculator)
a:= sqrt(2); (a now stands for the real number sqrt(2), not an approximation)
a; (what do you expect)
evalf(a); (evalf gives floating point approx with current precision, held as Digits)
Digits:=50; (note that Digits and digits are not the same)
evalf(a); (50 digits precision)
evalf(Pi); (note that Pi and pi are not the same)
p:= x+y; (p now stands for x+y)
p;
*q:=p**5; (You can also use carat for exponentiation)*
*q:= x*q + y*q;*
expand(q); (multiply and collect)
*q/p**10;*
simplify(%); (percent sign means pervious value, simplify tries to simplify)
subs(x=4.5, q); (substitution)
sort(q, [x,y]);
sort(q, [y,x]);
with(linalg); (turns on the linear algebra package)

*b:=matrix([[x,y], [x*z, z]]);*
b;
evalm(b); (displays matrix b)
c:= 1/b; (matrix inverse)
d:= b& c; (&*, ampersand followed by *, is matrix multiplication)*
d;
simplify(d);
simplify(evalm(d)); (only at this stage do we get identity displayed)
subs(x= 2.1, evalm(c)); (matrix substitution, note evalm inside subs)
l1:=[a,b,d]; (list, refer to help)
s1 $:= \{a, b, c\}$; *(set, order independent, no duplicates, as mathematical set)*
sq:= a,b,c; (sequence)
*plot(sin(x)*exp(-x/5), x= 0 .. 4*Pi); (try also 3d plots)*

Exercise 1 *This is an exercise in interactive use of maple. It is harder than it looks. Please persist. The problem is to tell maple what we want it to do. Note that the whole computation is somewhat ambiguous, and there is room for doubt about the correctness of the computation. What is the correct framework? What are* x_1, x_2, x_3*? Real? Which square root is meant? The only way to be sure that this result is correct is to make a proof with pencil and paper.*
First turn on linear algebra package by
with(linalg);
Define x *to be the 1 by 3 row matrix* $[x_1, x_2, x_3]$. *For* j *between 1 and 3, define* s_j *to be* $(\sum_{j\leq k\leq 3} x_k^2)^{1/2}$. *Define* H *to be a 3 by 2 matrix, with*
$H[i,j] = 0$ *if* $i < j$
$H[i,i] = s_{i+1}/s_i$ *for* $i = 1, 2$
$H[i,j] = -x_i x_j/(s_j s_{j+1}$ *for* $1 \leq j < i \leq 3$.
Verify:

1. *transpose*$(H)H$ *is 2 by 2 identity*
2. xH *is 0.*
3. *Let* $p = H * transpose(H)$. *Then* $transpose(p) = p$, *and* $p * p = p$

The matrices H and p are basic parts of the PSLQ algorithm, which will be discussed later.

1.2 Procedures in Maple

Advice: make a directory to hold all maple files and go into this directory before running maple. Create maple source with any editor, e.g. emacs. Save it, e.g. fb.mpl. Start maple. Command.
read “fb.mpl”;
and then use procedure.

Example 2

```
fib := proc(x:: nonnegint)
# Fibonacci function
  option remember;
  if x = 0 then 0
  elif x = 1 then 1
  else fib(x-1) + fib(x-2) fi
end:
```

```
# some tests

fib(1);

fib(6);
fib(7);
fib(8);
```

These notes will gradually expand and will be handed out gradually. A solution to the first exercise is at

```
~masdr/public_html/exer1.mws
```

and you should be able to open it from Maple. For the problems class on Thursday, try the following:

Exercise 2 *Consider the above Fibonacci procedure and the two following which were discussed in class.*

```
cheb := proc(n:: integer)
# returns table of Chebyshev polynomials up to n
  local p,k;
  p[0]:=1;
  p[1]:=x;
  if n <= 1 then RETURN(eval(p))
  fi;
  for k from 2 to n do
    p[k] :=expand(2*x*p[k-1] - p[k-2])
  od;
  eval(p)
end:


lop := proc(f::algebraic,g::algebraic,x::name, a)
#  this procedure attempts to find the limit, as x-->a of
#  f(x)/g(x), using L'Hopital method.  It assumes f,g are expressions
#   for infinitely differentiable
#   real valued functions of a real variable called x
  if eval(subs(x=a,g)) <> 0 then eval(subs(x=a,f/g))
  elif eval(subs(x=a,f)) <> 0 then "no finite limit"
  else lop(diff(f,x), diff(g,x),x,a)
  fi
end:
```

1. *Find* $limit_{x\to 0}(x^3 sin(x)/(1-cos(x))^2$.
2. *Modify the Fibonacci procedure to return a table.*
3. *Modify the Chebyshev procedure to return the nth polynomial*
4. *Modify the L'Hopital procedure so that it does not use recursion.*
5. *Does the L'Hopital procedure always terminate? Can you fix or alleviate this problem?*
6. *Consider the Euclid GCD algorithm given below. Write a procedure for the extended Euclidean algorithm, which returns a,b,g where g is the gcd(x,y) and g=ax+by.*

2 Problems and Algorithms

Notation: N, Z, R, C are, respectively, the natural numbers including 0, the integers, the real numbers and the complex numbers. $Z[x]$ is the set of polynomials in x with integral coefficients, i.e. expressions of the form $a_0x^d + ... + a_d$, where $a_0, ...a_d$ are integers. $Z[x_1, ..., x_n]$ is the set of polynomials in variables $x_1, ..., x_n$ with integral coefficients.

Problem means a family of related problems. Examples:

1. GCD over Z. Given integral x, y, find their greatest common divisor.
2. Factorisation over Z. Given n in Z, express n as a product of prime numbers.
3. Factorisation over $Z[x]$, down to integers. Given $p(x)$ in $Z[x]$, express $p[x]$ as a product of integers and irreducible factors in $Z[x]$.
4. (Hilbert's tenth Problem). Given $p(x_1, ..., x_n)$ in $Z[x_1, ..., x_n]$, decide whether or not there exist integers $z_1, ..., z_n$ so that $p(z_1, ..., z_n) = 0$.
5. (Polynomial System Solving) Given $(p_1, ..., p_n) : C^m \to C^n$, and each p_i in $Z[x_1, ..., x_m]$, decide whether or not there exists α in C^m so that $(p_1, ..., p_m)(\alpha) = 0$.
6. Identity problem for trigonometric and absolute value Functions. Given expression for function $f(x)$ built up from $x, \pi, 1, +, -, *, sin(x), cos(x), \mid x \mid$ and composition, decide whether or not $f(x)$ is identically zero.
7. Integer Relation Finding. Given real numbers $x_1, ..., x_n$ decide whether or not there exist integers $m_1, ..., m_n$ not all zero and with absolute value bounded by M so that $m_1x_1 + ... + m_nx_n = 0$.

An algorithm to solve such a problem is a method which given an instance of the problem will determine a solution after finitely many steps. Assume some measure of the size of the input. We measure the complexity of an algorithm as the number of steps it takes as a function of the size of the input. We will get different versions of complexity, depending on what we allow as a single step.

1. Classical bit complexity. Number of steps of a Turing machine.
2. Complexity over Z. We allow addition, subtraction and multiplications of arbitrary integers in one step.
3. Complexity over R. We allow addition, subtraction, multiplication, and testing of sign for arbitrary real numbers in one step. Some problems, e.g. Integer relation finding as above, only make sense over R.

We can classify problems by difficulty:

1. Recursively unsolvable. No algorithm is possible. Hilbert's tenth problem and the Identity problem for trigonometric and absolute value functions are examples. There is no hope of a uniform solution method.
2. Recursively solvable, but intractable. In this case, an algorithm exists, but the complexity of the solution takes a number of steps which increases faster than any polynomial in the input size. Most people believe that factorisation over Z is in this category (with size of number measured by length of digital representation). Note that even for an intractable problem, a computer program may be useful.
3. Polynomial complexity solvable. These are problems for which an algorithm exists with the number of steps for a solution bounded by some polynomial in the size of the input. Integer relation finding is in this category, with the size of the bound M measured as the number of digits to express M, and with complexity over R. Factorisation over $Z[x]$ down to integers is another problem in this category.

4. Feasible in practice. GCD over Z for example.

See `www.computer.org/cise/cs2000/c1toc/htm`

for a list of "the best algorithms of the century". It is a good idea to read all these articles, but we are only going to study one of them.

If possible we would like to translate algorithms into usable computer programs. A computer algebra system such as Maple is a good framework for this. We will consider integer GCD, Integer relation finding, numerical solving of systems of equations by Newton's method, and factorisation over $Z[x]$.

3 Euclid's GCD for Z

For integers x and y, $x \mid y$ means that x is a factor of y. An integer d that divides both x and y is called a common divisor of x and y. $GCD(x, y)$ is the greatest common divisor of x and y. To find GCD, we can form the remainder sequence:

$n_0 = x$
$n_1 = y$
$n_0 = q_0 n_1 + n_2$, where $0 \leq n_2 <\mid n_1 \mid$
$n_1 = q_1 n_2 + n_3$, where $0 \leq n_3 <\mid n_2 \mid$
...
$n_i = q_i n_{i+1} + n_{i+2}$, where $0 \leq n_{i+2} <\mid n_{i+1} \mid$
...
$n_k = q_k n_{k+1} + 0$

until $n_{k+2} = 0$. Then n_{k+1} is the GCD Of x and y. In each case, $n_{i+2} = n_i \ mod \ n_{i+1}$. In other words, n_{i+2} is the remainder when n_i is divided by n_{i+1}.

In Maple:

```
euc := proc(x::integer, y:: integer)
  if x < y then euc(y,x)              # swap
  elif y = 0 then x                   # terminate
  else euc(y, x mod y) fi             # reduce
end:
```

Theorem 1 *Suppose x and y are integers, not both zero. Let $M = \{ax + by : a, b : Z\}$. Then $GCD(x, y) = Minimum\{g : g \in M, g > 0\}$.*

Proof. The proof uses the fact that $0 \leq x \ mod \ y <\mid y \mid$. □

There is also an extended Euclidean algorithm which finds a and b so that $ax + by = GCD(x, y)$. Try to write this in Maple. It should return the sequence $a, b, GCD(x, y)$.

3.1 efficiency

Complexity of Euclid GCD over Z is $O(n)$, where n is the size of the input, i.e. length in digits.

To show this, consider remainder sequence of length k, and pick x and y as small as possible with such a length. In order to do this, take all the quotients q_i to be 1. We get a backwards Fibonacci sequence as the worst case.

f_i, the i th number in the Fibonacci sequence has a remainder sequence of length $i - 1$ when divided by f_{i-1}, and no smaller number has a remainder sequence so long. Therefore the number of steps taken by the Euclidean algorithm is bounded by a multiple of the least i so that f_i is larger than $max(\mid x \mid, \mid y \mid)$.

We have $f_i < 2f_{i-1}$ for i bigger than 2, and so $f_{i+1} > 3/2 f_i > C(3/2)^i$, for some C, and for i bigger than 2. So $i < O(log(f_i))$, which gives us our result that the length of the remainder sequence starting with x and y is bounded by $O(n)$, where n is the size of (x, y) measured in digits.

It is possible to get a better estimate for the Fibonacci sequence, using standard techniques for solving recurrence relations. It can be shown that $f_i = c_1\phi^i + c_2(-\phi)^{-i}$, where $\phi = (1 + 5^{1/2})/2$, and c_1, c_2 are $5^{-1/2}, -5^{-1/2}$ respectively.

4 Continued Fractions

See H. Davenport, *Higher Arithmetic*, (512.41 DAV).

Definition. 1 *$CF(q0, q1, ..., qn)$, the continued fraction with partial quotients $q0, q1, ..., qn$, is*

```
q0 + 1
    ----------------
    q1 + 1
         ------------
          ...
                +1
                ---
                 qn
```

$CF(q_0, ..., q_n) = P_n(q_0, ..., q_n)/Q_n(q_0, ..., q_n)$ where P_n and Q_n are polynomials in $Z[q_0, ..., q_n]$. $CF(q_0, ..., q_n)$ is called the n th convergent of the sequence of partial quotients $q_0, ..., q_n, ...$.

$floor(x)$ is the largest integer n with $n \leq x$.

Definition. 2 *Suppose $\alpha : R$. Define $cf(\alpha) = floor(\alpha)$ if $\alpha = floor(\alpha)$, and otherwise let*
$cf(\alpha) = floor(\alpha) + 1/(cf(1/(\alpha - floor(\alpha))))$.

This defines a sequence of partial quotients $q_0, ...$ The sequence terminates if α is rational and otherwise it goes on forever. The convergents are good approximations to α.

Lemma 1 $Q_n(q_0, ..., q_n) = P_{n-1}(q_1, ..., q_n)$
$P_n(q_0, ..., q_n) = q_0 P_{n-1}(q_1, ..., q_n) + P_{n-2}(q_2, ..., q_n)$

Proof. $CF(q_0, ..., q_n) = P_n/Q_n = q_0 + 1/CF(q_1, ..., q_n) = q_0 + Q_{n-1}(q_1, ..., q_n)/P_{n-1}(q_1, ..., q_n)$. Put over common denominator.
□

Theorem 2 $P_{n+1}(q_0, ..., q_{n+1}) = q_{n+1}P_n(q_0, ..., q_n) + P_{n-1}(q_0, ..., q_{n-1})$
$Q_{n+1}(q_0, ..., q_{n+1}) = q_{n+1}Q_n(q_0, ..., q_n) + Q_{n-1}(q_0, ..., q_{n-1})$

Proof. By induction on n.
□

Theorem 3 *For any real α there exists p/q in Q so that $\mid \alpha - p/q \mid < 1/q^2$*

Proof. Clear if α is rational. Suppose not. Define x mod 1 to be $x - floor(x)$. Consider

$$M_n = \{q\alpha \ mod \ 1 : 0 < q \leq n\} \cup \{0, 1\}$$

M_n consists of $n+2$ distinct numbers, all in $[0,1]$. By the pigeon hole principle, there are two distinct numbers x and y in M_n which differ by no more than $1/n$.

$\mid q_1\alpha - p_1 - (q_2\alpha - p_2) \mid \leq 1/n$

So $\mid \alpha - (p_2 - p_1)/(q_1 - q_2) \mid \leq 1/(n \mid q_1 - q_2 \mid) < 1/(q_1 - q_2)^2$

$\square$

The proof shows something stronger: infinitely many of these good approximations. How to find them?

Theorem 4 $\mid P_n/Q_n - \alpha \mid = Minimum\{\mid p/q - \alpha :\mid q \mid \leq Q_n \mid$

$\mid P_n/Q_n - \alpha \mid < 1/(Q_nQ_{n+1}) < 1/Q_n^2$

where P_n/Q_n is n the convergent to α.

Proof omitted. See Davenport.

This theorem means that the n convergents are optimal rational approximations to α.

4.1 Coursework 1: M70

Write a procedure which computes the partial quotients for the continued fraction representation for given real α. Also give a procedure to compute the nth convergent to α. Consider $\alpha = (n)^{1/m}$, for n and m in the range given to you by dsr. Find the first 51 partial quotients, $q_0..q_{50}$ for each such number. Find the best available rational approximation for each α, and tabulate the differences between each α and its best approximation. Marking: Simplicity and apparent correctness of code 50; Explanation 25; Tests and results 25.

Due Friday March 9. Hand in to Mathematics Departmental Office before 3.30 pm.

5 Diophantine Equations

A Diophantine problem is one in which we are looking for whole number solutions to some condition.

5.1 Pell Equation

$x^2 - Dy^2 = 1$

5.2 Fermat's last theorem and the ABC conjecture

$x^n + y^n = z^n, n > 2$

Conjecture 1 *ABC conjecture. There exists a real number κ so that if A, B, C are natural numbers which are relatively prime and such that $A + B = C$, then*

$C \leq (\prod_{p|ABC,p:prime} p)^\kappa$

6 LLL and PSLQ algorithms

6.1 Gram Schmidt

Gram Schmidt orthogonalisation process

Let x^T mean the transpose of vector or matrix x.

Recall that two non zero vectors a and b in R^n are orthogonal if and only if the vector product (i.e. dot product, scalar product) $a \circ b = ab^T$ is zero. For example, $(1,1) \circ (1,-1) = 0$. Orthogonality is a nice property of a basis of a vector space.

Let $v_1, ..., v_k$ be a list of vectors in R^n which is a basis for a vector space $S(v_1, ..., v_k)$. The Gram Schmidt process produces an orthogonal basis $v_1*, ..., v_k*$ for this vector space as follows.

Define $v_1* := v_1$, and $B_1 := v_1 * \circ v_1*$, where $\circ$ means vector product.
Now continue recursively,
for all $i > 1$,
for all $j < i$
set $\mu_{ij} := v_i \circ v_j * / B_j$
(μ_{ij} is the projection of v_i in direction v_j*, scaled by v_j*.)
Then
$v_i* := v_i - \sum_{j<i} \mu_{ij} v_j*$,
and $B_i := v_i * \circ v_i*$.

So v_i* is what is left when we take away whatever can be expressed in terms of $v_1*, ..., v_{i-1}*$. Note that the output $(v_1*, ..., v_k*)$ need not be unit vectors.

6.2 Correction to proof of theorem 3

Add 0 and 1 to M_n.

$$M_n = \{q\alpha \ mod \ 1 : 0 < q \leq n\} \cup \{0, 1\}$$

Then M_n consists of $n+2$ numbers all in $[0, 1]$.

6.3 More about the coursework 1

You will be finding continued fractions for $n^{1/m}$. Get your number n from the class list pinned to door of 1w2.2a. Pick bound M. If you are not on the list, pick a new number less than 50. You should hand in

1. Code for procedures which compute continued fraction for n quotients for real α, and kth convergent for same.
2. Some tests of the above, attempting to demonstrate correctness
3. For your n and for m from 2 to M, find partial quotients $q_0, ..., q_{50}$. Find largest q_k, and report $k-1$ convergent, which should be the best approximation for $n^{1/m}$, and the number of decimal places of accuracy of the approximation.
4. What precision do you need? Discuss and explain.
5. Decide how to measure goodness of approximation. (See Thue Siegel Roth theorem.) Give best overall approximation which you found.

6.4 Quick review of linear algebra

Let $v_1, ..., v_k$ be vectors in R^n.

Definition. 3 *$v_1, ..., v_n$ are linearly independent if*

$$c_1 v_1 + ... + c_n v_n = 0$$

is impossible with $c_1, ..., c_n$ real numbers which are not all zero.

Definition. 4 *$V \subseteq R^n$ is a vector space if V is closed under vector addition and subtraction, and V is closed under multiplication by real scalars.*

Example: plane through origin.

Definition. 5 *$Span(v_1, ..., v_n)$ is the set of vectors which can be written in the form*
$c_1v_1 + ... + c_nv_n$
where $c_1, ..., c_n$ are real numbers.

Definition. 6 *$(v_1, ..., v_k)$ is a basis for a vector space V if $V = Span(v_1, ..., v_k)$ and $v_1, ..., v_k$ are linearly independent.*

Notation: If $v_1 = (x_1, ..., x_n)$ and $v_2 = (y_1, ..., y_n)$ then $v_1 \circ v_2 = x_1y_1 + ... + x_ny_n$.
$\mid v \mid = (v \circ v)^{1/2}$. This is Euclidean length.

6.5 Lattice Reduction

Try ?lattice; in maple.

Advice: search for Lenstra AND LLL in yahoo, and read around the subject.

Let $(b_1, ..., b_k)$ be a list of linearly independent vectors in R^k.

A lattice in R^k is a subset $L(b_1, ..., b_k)$ of the form $\{z_1b_1 + ... + z_kb_k\}$, with $z_1, ..., z_k$ are integers. We will say that $(b_1, ..., b_k)$ is a basis of $L(b_1, ..., b_k)$.

The problem of finding a shortest non zero vector in a given lattice is considered to be very difficult. There is, however, a good method, the LLL algorithm, for finding an approximation to a shortest vector. This is based on the Gram Schmidt method, which was discussed earlier. The essential geometric idea is that if the basis were orthogonal, the smallest vector in it would also be the smallest in the lattice. So the algorithm follows the Gram Schmidt methods as closely as possible, but using integral coefficients. (We can not subtract arbitrary multiples of one vector from another since that might take us outside the lattice.)

The following definition (due to Lenstra, Lenstra, Lovasz) uses the notation of the Gram Schmidt method.

Definition. 7 *A basis $(b_1, ..., b_k)$ is LLL reduced if it satisfies the conditions:*

1. $\mid \mu_{ij} \mid \leq 1/2$ *for all* $1 \leq j \leq i \leq k$
2. $\mid b_i^* + \mu_{ii-1}b_{i-1}^* \mid^2 \geq (3/4) \mid b_{i-1}^* \mid^2$, *for* $1 < i \leq k$.

It is intended to capture the idea that the vectors are approximately orthogonal.

Theorem 5 *Suppose basis $(v_1, ..., v_n)$ has Gram Schmidt basis $(v_1^*, ..., v_n^*)$. Let m be any non zero vector in the lattice generated by $(v_1, ..., v_n)$. Then $\mid m \mid \geq Minimum(\mid v_i^* \mid : 1 \leq i \leq n)$*

Proof. $m = a_1v_1 + ... + a_nv_n$, where $a_1, ..., a_n$ are integers, not all zero. Also $m = c_1v_1^* + ... + c_nv_n^*$. Pick j maximal so that $a_j \neq 0$.

$(v_1, ..., v_j)$ Gram Schmidt's into $(v_1^*, ..., v_j^*)$. Thus
$a_1v_1 + ... + a_jv_j = m = c_1v_1^* + ... c_jv_j^*$.
We also have $v_j = v_j^* + \sum_{k<j} \mu_{jk}v_k^*$
$(v_1, ..., v_{j-1})$ Gram Schmidt's into $(v_1^*, ..., v_{j-1}^*)$. Thus
$m = a_jv_j^* + Terms\ in\ (v_1^*, ..., v_{j-1}^*) = c_jv_j^* + Terms\ in\ (v_1^*, ..., v_{j-1}^*)$, and since the basis vectors are linearly independent, $a_j = c_j$. Thus c_j is a non zero integer, and $\mid m \mid \geq \mid v_j^* \mid$.
□

Another proof can be given via taking the dot product of m and v_j^*.

The result we need is:

Theorem 6 *Let L be a lattice and let $(b_1, ..., b_k)$ be a reduced basis of L. Let x be any non zero vector in L. Then $\mid x \mid \geq 2^{-(n-1)/2} \mid b_1 \mid$.*

Proof.

Perform Gram Schmidt on $(b_1, ..., b_k)$ to get $(b_1^*, ..., b_k^*)$. We know (by the previous theorem) that the lengths of the orthogonal vectors give us a lower bound on the length of any vector in the lattice. Since the basis is reduced, we also have

$\mid b_i^* + \mu_{ii-1} b_{i-1}^* \mid^2 \geq (3/4) \mid b_{i-1}^* \mid^2$, for $1 < i \leq k$.

Thus $\mid b_i^* \mid^2 \geq (1/2) \mid b_{i-1}^* \mid^2$

So for each i

$\mid b_i^* \mid \geq \mid b_1 \mid 2^{(n-1)/2}$, and therefore this is also a lower bound for length of non zero vectors in the lattice. □

Here is a version of LLL.

LLL algorithm

Input $(v_1, ..., v_n)$

Apply Gram Schmidt to get $(v_1^*, ..., v_n^*)$ and (μ_{ij}) (so that we can test if $(v_1, ..., v_n)$ is reduced)

While (not reduced $(v_1, ..., v_n)$)

1. Reset each v_i to $v_i - \sum_{j<i} nint(\mu_{ij}) v_j$, for $i = 2..., n$
2. Do Gram Schmidt process again getting $(v_1*, ..., v_n*)$, and (μ_{ij})
3. If $\mid v_i^* + \mu_{ii-1} v_{i-1}* \mid^2 < 3/4 \mid v_{i-1}^* \mid^2$, for some i, then swap v_i and v_{i-1} for first such i
4. Apply Gram Schmidt again, to get $(v_1^*, ..., v_n^*)$ and (μ_{ij})

Note similarity with Euclid GCD. Essentially we take little things away from big things as far as possible, and then we swap, and continue until finished. Proof of termination (omitted here) is much harder for LLL than for GCD , but LLL does always finish in polynomial time. Complexity is $O(n^6 (\log \mid v_{ij} \mid)^3$. Much slower than GCD over Z, but still usable on medium size problems. Dependence on dimension n is worse than on size of coefficients, as we would expect. Many optimizations and variations of the above technique are possible.

6.6 Integer relation finding

See papers in:

```
www.farcaster.com/papers/sm-thesis/node5.html

(LLL exposition)

 www.nersc.gov/~dhb/

(Analysis of PSLQ, an integer relation finding algorithm)

www.cecm.sfu.ca/personal/borwein/

www.cecm.sfu.ca/Projects/IntegerRelations/index.html

(Examples in Maple)

www.computer.org/cise/cs2000/c1toc.htm

(Integer relation finding)
```

Definition. 8 *Suppose $x = (x_1, ..., x_n)$ is a vector in R^n. An integer relation for x is a vector $m = (m_1, ..., m_n)$ of integers, not all zero, so that $xm^T = 0$.*

An n by n matrix Q is unitary if $QQ^T = I$. Let $U(n, R)$ denote the set of unitary n dimensional matrices with real coefficients.

Let $GL(n, Z)$ denote the set of n by n matrices with integral coefficients, and with determinant 1. $GL(n, Z)$ is a group.

A matrix is lower trapezoidal if it has more rows than columns and if it is all 0 above the diagonal.

Definition. 9 *Suppose $x = (x_1, ..., x_n)$ is a unit vector in R^n, and that no coordinate entry of x is 0. For $1 \leq j \leq n$, define*
$s_j^2 = \sum_{j \leq k \leq n} x_k^2$
Define the n by $n-1$ lower trapezoidal matrix $H_x = (h_{ij}$ by
$h_{ij} = 0$ *if* $1 \leq i < j \leq n-1$
$h_{ij} = s_{i+1}/s_i$ *if* $1 \leq i = j \leq n-1$
$h_{ij} = -x_i x_j/(s_j s_{j+1})$ *if* $1 \leq j < i \leq n$

Lemma 2

1. *$H_x^T H_x = I_{n-1}$, i.e. the columns of H_x are orthogonal, and unit.*
2. *$\mid H_x \mid = \sqrt(n-1)$*
3. *$xH_x = 0$*

The columns of H_x are a basis for $x^\perp$, the space orthogonal to x.

Lemma 3 *Let $P_x = H_x H_x^T$. Then*

1. *$P_x^T = P_x$*
2. *$P_x = I_n - x^T x$*
3. *$P_x^2 = P_x$*
4. *$\mid P_x \mid = \sqrt(n-1)$*
5. *$P_x z^T = z^T$ for any z in $x^\perp$*
6. *$P_x m^T = m^T$ for any integer relation for x.*

Theorem 7 *Suppose that m is an integer relation for x. Let $H = AH_xQ$, where A is in $GL(n, Z)$ and Q is in $U(n-1)$, and suppose that H is lower trapezoidal, and each diagonal element $h_{j,j} \neq 0$. Then*
$\mid m \mid \geq Minimum(1/ \mid h_{ii})$

Proof.
$P_x = H_x H_x^T = I_n - x^T x$.
$P_x m^T = m^T$, as in lemma above.
$AP_x m^T = Am^T$
$AH_x H_x^T m^T = Am^T$.
We have $AH_x = HQ^T$, since $QQ^T = I_{n-1}$.
So $HQ^T H_x^T m^T = Am^T$, and Am^T is a column of integers, not all 0.
Let $Q_{H,j}$ be the j th row of $Q^T H_x^T$. This is a unit vector since $Q^T H_x^T H_x Q = I_{n-1}$.
Let A_j be the j th row of A. Pick j minimal with $A_j m^T \neq 0$.
$Q_{H,k} m^T = 0$ for $k < j$, since j was chosen minimal.
$A_j m^T = h_{jj} Q_{H,j} m^T$. So $\mid m \mid \geq 1/h_{jj}$.
□

The idea of this theorem is that H is a transformation of H_x, using A and Q. We get a lower bound on the size of any integer relation for x by looking at the diagonal elements of H. Any matrices A and Q can be used here, provided only that H is lower trapezoidal. As long as we choose the A and Q so that the lower bound is increasing, we are making progress.

PSLQ algorithm

Pick $\gamma > 2/(3)^{1/2}$.

Set $H = H_x$

While (no element of x or diagonal of H is zero)

1. Hermite reduce H, by integral row operations, (subtracting integral multiples of the i th row from rows below i), so that in each column the diagonal has at least twice absolute value of the other elements. $H := AH, x := xA^{-1}$, where A in $GL(n, Z)$ represents the reduction. The reduced matrix is still lower trapezoidal.
2. Find r so that $\gamma^r \mid h_{rr} \mid$ is maximal. Interchange row r and row $r+1$ of H. Let R represent this interchange. $H := RH, x := xR^{-1}$. Let $\alpha = H(r,r), \beta = H(r+1,r), \lambda = H(r+1,r+1), \delta = (\beta^2+\lambda^2)^{1/2}$
3. H may not be lower trapezoidal at this point, since the interchange may have created a corner above the diagonal. Do column operations, represented by unitary matrix Q to get rid of this corner. $H := HQ$. Q is in $U(n-1)$, and may be defined by the following: $Q(r,r) = \beta/\delta, Q(r,r+1) = -\lambda/\delta, Q(r+1,r) = \lambda/\delta, Q(r+1,r+1) = \beta/\delta$, and otherwise Q is the $n-1$ by $n-1$ identity.

At the beginning and end of each loop, H is lower trapezoidal. Termination implies that an integer relation has been found. Non termination, means that any relation must satisfy $\mid m \mid \geq Min(1/(\mid h_{ii} \mid)$, and this number increases at each loop. Note similarity with Euclidean algorithm.

An implementation in maple , written by Simon Langley, can be found (together with some examples):

```
~masdr/public_html/pslq.ms

~masdr/public_html/pslq.mws
```

6.7 A sample application of PSLQ

Suppose we have the ability to approximate a real number α with high accuracy, and we believe α is a root of a polynomial with integral coefficients of degree n.

$a_n\alpha^n + \ldots a_0 = 0$

with $(a_n, \ldots, a_0)$ integers. Then $(a_n, \ldots, a_0)$ is an integer relation for $x = (\alpha^n, \ldots, \alpha, 1)$. So we can use PSLQ to search for the coefficients.

On the other hand if we believe that α is not algebraic of degree n, we can use PSLQ to exclude the possibility of polynomials with coefficients which are all below some bound, M. To do this, just do PSLQ until the bound produced by the theorem above exceeds nM.

6.8 Programming exercises

These will be discussed in the problems classes. The object is to learn how to use matrices in maple, and, in general, the linear algebra package. An important point: if B is a matrix, A:=B sets address of A to address of B, so that afterwards A and B are not independent. If you want an independent copy, you need to declare A and then write a loop or so to copy B into A.

1. Write Gram Schmidt procedure. (There is one already in the linear algebra package. To see the code for this and other procedures in maple, use interface(verboseproc=2), print(procedurename). Refer to help.)

2. Write procedure which computes H_x from x for pslq.

7 Computation in $Z(x_1, \ldots, x_n)$

7.1 Some Algebra

7.2 Introduction

Definition. 10 *Suppose K is a set. A polynomial in x with coefficients in K is an* expression *of the form*
$a_d x^d + a_{d-1} x^{d-1} + ... + a_0$
where d is a natural number, and $a_d, ..., a_0$ are all elements of K.

Notation: $K[x]$ is the set of polynomials in x with coefficients in K.

Definition. 11 *A power series in x with coefficients in K is an* expression *of the form*
$\sum_{n=0}^{\infty} a_n x^n = a_0 + a_1 x + a_2 x^2 + ...$
where $a_0, a_1, ...$ are all in K.

Notation: $K[[x]]$ is the set of power series in x with coefficients in K.

Normally it is assumed that K is a type with $+, -, *$ defined, and in that case we can define $+, -, *$ uniformly in $K[x]$ from the definitions in K. We do, however, need to make some assumptions about how $+, -, *$ behave.

7.3 Basic Definitions

Definition. 12 *In the following a ring is a structure $(D, +, *, 0, 1, =)$ where*

- *D is a collection of objects*
- *$+, *$ are operations from $D \times D$ to D*
- *0 and 1 are in D, and $0 \neq 1$*
- *$=$ is the usual equality on D*
- *$(D, +, 0)$ is an Abelian group*
- *$*$ is associative and commutative, and $1 * x = x$ for all x in D*
- *$x * (y + z) = x * y + x * z$ for all x, y, z in D*

The full algebraic name of this idea is "commutative ring with unit", but we will just call it a ring in the following.

Definition. 13 *An integral domain is a ring in which $x * y = 0$ only happens when either $x = 0$ or $y = 0$.*

Definition. 14 *A field is a commutative ring with unit in which every non zero object has a multiplicative inverse.*

Every integral domain extends uniquely to a field of fractions. For example, $\mathbf{Q}$ is the field of fractions of $\mathbf{Z}$.

If I is an integral domain, $I[X]$ is the integral domain of polynomials in X with coefficients in I. $I(X)$ is the field of fractions. Similarly, $K[x_1, ..., x_n]$ is multivariate polynomials in $x_1, ..., x_n$ with coefficients in K; and $K(x_1, ..., x_n)$ is the field of fractions.

Definition. 15 *x is a unit in ring D if there is a y in D so that $x * y = 1$.*

Definition. 16 *Elements x and y in a ring are associates if there is a unit u in the ring so that $x * u = y$*

Definition. 17 *We will say that x is a factor of y in a ring D, and write this as $x \mid y$ if there is a z in D so that $x * z = y$.*

Notation: $\wedge$ means logical AND. $\vee$ means logical OR. $\neg$ means logical negation. $\rightarrow$ means logical implication. $\leftrightarrow$ means if and only if. $\forall$ means for all. $\exists$ means there exists.

Definition. 18 *Suppose K is an integral domain and a and b are in K. An object g in K is a gcd of a and b if*
$g \mid a \wedge g \mid b$, *and*
$(\forall h \in D)((h \mid a \wedge h \mid b) \rightarrow h \mid g)$

In other words if g is a gcd of a and b, then g divides both a and b, and g is as large as it can be with this property in the sense that g is divisible by anything in K which divides both a and b.

According to this definition both -4 and 4 would be gcds of 12 and 16 over Z. When K has an ordering, we usually define the GCD to be the largest object which satisfies the gcd definition given above.

Theorem 8 *If g_1 and g_2 are gcds of x and y in integral domain K, then g_1 and g_2 are associates.*

A little later in the course we will look at how to compute polynomial gcds. At this point, we do not even have a proof that they exist.

Definition. 19 *Let K be an integral domain in which gcds exist. For p in $K[x]$, define $content(p) = gcd(\{Coefficients(p)\})$. Define $PrimitivePart(p) = p/content(p)$.*

A polynomial is *primitive* if its content is 1.

Definition. 20 *A non unit polynomial p is irreducible in $K[x]$ if $p = q * r$ is impossible for q and r in $K[x]$, unless either q or r is a unit.*

Theorem 9 $\mathbf{Z}[x_1, ..., x_n]$ *has unique factorisation.*
(Every non zero non unit polynomial in this domain can be expressed as a product of irreducible factors in this domain. Furthermore the product is unique except for the order of the factors, and multiplication by units.)

This theorem implies that gcds are defined in $\mathbf{Z}[x_1, ..., x_n]$.

7.4 Ring morphisms

Definition. 21 *Suppose $K_1 = (D, +, *, zero(K_1), one(K_1), =)$ and $K_2 = (E, \oplus, \otimes, zero(K_2), one(K_2), =)$ are rings. A ring morphism from K_1 to K_2 is a map*
$f : D \rightarrow E$
so that

1. $f(x + y) = f(x) \oplus f(y)$
2. $f(x * y) = f(x) \otimes f(y)$
3. $f(one(K_1)) = one(K_2)$.
4. $f(zero(K_1)) = zero(K_2)$

Comments: In the first item above, the first addition $x + y$ is going on in K_1, and the second addition $f(x) \oplus f(y)$ is going on in K_2. So this says that if you first add in K_1 and then apply f, you will get the same result as if you first applied f to x and y and then added the results in K_2.

The fourth condition is a consequence of the first three

Ring morphisms are also called ring homomorphisms. Ring morphisms are to rings what group homomorphisms are to groups.

The *category* of rings consists of all rings and all ring morphisms.

Example 3 *The natural map $f : Z \to Z_p$, where $f(n)$ is n modulo p, is a ring morphism.*

7.5 Representation of Rings

A representation of a ring is a collection of expressions which stand for objects in the ring. A representation is normal if 0 has only one representation. A representation is canonical if every object in the ring has only one representation.

If E_1 and E_2 are two expressions in a representation, we will say that $E_1 = E_2$ if E_1 and E_2 stand for the same object. For example, $0 + 0 = 0$, even though $0 + 0$ and 0 are not the same as expressions.

We will say that a representation has decidable identity problem if there is an algorithm to decide whether or not $E_1 = E_2$, given E_1 and E_2.

7.6 Integers of arbitrary size

Choose a base b, and represent integers, canonically, as either

0

or as a string $d_k d_{k-1} ... d_0$

or as a signed string $-d_k d_{k-1} ... d_0$

with $0 \leq d_i < b$.

Theorem 10 *Division Theorem for* $\mathbf{Z}$.

For any a and $b \neq 0$ in $\mathbf{Z}$, there exists unique q and r in $\mathbf{Z}$ so that

$a = q * b + r \wedge \mid r \mid < \mid b \mid \wedge \mid q * b \mid \leq \mid a \mid$

We will write the number r asserted to exist by this theorem as $rem(a, b)$.

Definition. 22 *a and b are relatively prime if $gcd(a, b) = 1$.*

7.7 Rational numbers

We can represent rational numbers as p/q where $gcd(p, q) = 1$ and $q > 0$. This is a canonical representation.

Addition, subtraction, multiplication and division of rational numbers requires computation of gcds in Z, in order to put the result into canonical form.

7.8 Representation of Polynomials

7.8.1 Polynomials in $K[x]$

Suppose we have defined a canonical form for ring K. We can write a polynomial $p(x)$, of degree d, in $K[x]$, as an expression

$p(x) = a_0 x^d + ... + a_1 x + a_d$

with the coefficients $a_0, ..., a_d$ in canonical form, and $a_0 \neq 0$.

On the data structure level, we may then either represent $p(x)$ as an array of length $d+1$ of the coefficients $(a_0, a_1, ..., a_d)$

or as a list of pairs of coefficients and powers, for the non zero coefficients, i.e.

$\{(a_0, d), ...\}$.

The array representation is called dense and the list representation is called sparse.

7.9 Multivariate Polynomials

Suppose we are in $K[x_1, ..., x_n]$. There are a lot of choices as to how to write these polynomials.

7.9.1 Recursive sparse

The *recursive sparse* representation does this as

$K[x_1, ..., x_{n-1}][x_n]$.

That is, we consider a polynomial to be in x_n with coefficients in $K[x_1, ..., x_{n-1}]$.

So p in $K[x_1, ..., x_n]$ is written as

$$p = a_0 x_n^d + ... + a_d$$

where the coefficients $a_0, ..., a_d$ are all in canonical form in $K[x_1, ..., x_{n-1}]$, and a_0 is not 0, unless $p = 0$.

This gives us a canonical form, which depends upon initially ordering the variables by importance

$x_1 \prec x_2 \prec ... \prec x_n$

Of course, if we change the order of importance of the variables, the canonical forms change.

This representation is also called lexicographic recursive.

If y is the most important variable which actually occurs in polynomial p, we can write p as

$a_0 y^d + a_1 y^{d-1} ... + a_d$

with $a_0 \neq 0$. In this case we will define

$lv(p) = y$

$lc(p) = a_0$

$deg(p) = d$

$reduct(p) = a_1 y^{d-1} + ... + a_d$

Assume $deg(p) > 0$. p could then be represented as the list

$\{\{\{lv(p), deg(p)\}, lc(p)\}, reduct(p)\}$.

So, in terms of Lisp, the car of a polynomial is the leading term and and cdr is the reduct.

7.9.2 Other canonical forms in $\mathbf{Q}[x_1, ..., x_n]$

Definition. 23 *A power product is a term obtained by multiplying together the variables to various non negative powers.*

$x_1^{d_1} ... x_n^{d_n}$

with $d_i \geq 0$ for $i = 1, ..., n$. The sum of the powers, $d_1 + d_2 + ... + d_n$, is the total degree of a power product.

Definition. 24 *A monomial of $K[x_1, ..., x_n]$ is the product of a power product and an element of the ring K*

Definition. 25 *Suppose we order the variables $x_1 \prec x_2 \prec ... \prec x_n$. Let $a = c_1 x_1^{a_1} x_2^{a_2} ... x_n^{a_n}$ and $b = c_2 x_1^{b_1} x_2^{b_2} ... x_n^{b_n}$ be monomials. Then we will say $a \prec b$ if $a_n < b_n$ or if $a_n = b_n$ but $a_{n-1} < b_{n-1}$ or if $a_n = b_n$ and... $a_1 = b_1$ but $a_1 < b_1$. This is called lexicographic ordering on the monomials.*

Each ordering of the variables induces a different lexicographic ordering.

Definition. 26 *Another common ordering on monomials is by total degree, and then, in case of a tie, lexicographically.*

We can get various canonical forms by writing our polynomials as sums of monomials, in decreasing order, depending on what order we adopt. This can be represented as lists of representations of monomials. This is called a distributed representation.

A common form is obtained by using the total degree and then lexicographic order on the monomials.

Another common form is to use total degree and then reverse lex.

7.9.3 Fractions of polynomials

We can represent fractions of polynomials as ratios of polynomials with gcd 1.

In an integral domain, any two gcds of p and q are associates. We will normally talk about *the* gcd in this case, not distinguishing between associates.

7.10 Power Series

In terms of expressions, we can write a power series as

$\sum_{n=0}^{\infty} a_n x^n$

where a_n is some expression depending on n.

A truncated power series is an expression of the form

$\sum_{n=0}^{k} a_n$

where k is some natural number. One truncated power series

$\sum_{n=0}^{k} a_n x^n$

extends another power series $\sum_{n=0}^{j} b_n x^n$

if $j < k$ and $a_n = b_n$ for all $n \leq j$.

Of course we do not literally have to write down all infinitely many terms of a power series, as long as we have a rule for generating the coefficients. It would seem that we could construct a truncation of the power series, which we could use as an approximation, and keep some process ready to extend the truncation if needed.

Definition. 27 *We can represent an infinite power series as (T, P) where T is a truncated power series, and P is a process which produces another infinite power series representation $(T*, P*)$ where $T*$ extends T.*

The idea here is that we can see any finite part, as much as we like, of the power series. If we want more of it there is a process in the background which will get it. These are sometimes called lazy series.

7.11 Polynomial GCD

7.12 Gauss lemma and GCDs in $Z[x]$

The relationship between factorisation (and GCDs) in $Z[x]$ and $Q[x]$ was studied by Gauss. Gauss defined the content of a polynomial to be the GCD of its coefficients. A polynomial is said to be primitive if its content is 1. If A is a polynomial, $pp(A) = A/content(A)$. $pp(A)$ is called the primitive part of A. (Note that these definitions generalise to $K[x]$ where K is any ring in which there exist GCDs.)

Gauss Lemma For A, B in $Z[x]$, if A and B is primitive, so is AB.

Proof. By contradiction. Suppose A and B are primitive and AB is not primitive. So $content(AB) = c$, where c is not a unit. Let k be a prime so that $k \mid c$. Let $f_k : Z[x] \to Z_k[x]$ be the ring morphism which reduces coefficients mod k. That is

$f_k(a_d x^d + ... + A_0) = (a_d mod k)x^d + ...a_0 mod k$

Since k does not divide $content(A)$ and does not divide $content(B)$, we have

$f_k(A) \neq 0, f_k(B) \neq 0$.

But $Z_k[x]$ is an integral domain, $f_k(AB) = 0$ since k does divide $content(AB)$; and also $f_k(AB) = f_k(A) * f_k(B)$ since f_k is a ring morphism. So we have a contradiction.

□

Suppose A and B are in $Z[x]$.

Corollary 1 $pp(AB) = pp(A)pp(B)$

Corollary 2 $content(AB) = content(A)content(B)$

Corollary 3 $A \mid B$ *iff* $(content(A) \mid content(B)$ *and* $pp(A) \mid pp(B))$

Corollary 4 $GCD(A, B) = GCD(content(A), content(B)) * GCD(pp(A), pp(B))$

Corollary 5 *if A has a non trivial factorisation in $Q[x]$, then A also has a non trivial factorisation in $Z[x]$.*

7.13 Gauss lemma and GCDs in $Z[x_1, ..., x_n]$

We say that an integral domain has unique factorisation if, as in Z, every non zero non unit object can be expressed as a product of irreducible factors; and if such factorisation is unique, except for change of order of factors and multiplication by units.

We will use the following theorem without proof.

Theorem 11 *$Z[x_1, ..., x_n]$ has unique factorisation.*

The Gauss lemma can be generalised (with only small change to the proof) from $Z[x]$ to $K[x]$, where K is an integral domain with unique factorisation. In particular, consider $K[x_n]$, where K is $Z[x_1, ..., x_{n-1}]$.

Gauss Lemma 2 For A, B in $Z[x_1, ..., x_{n-1}][x_n]$, if A and B are primitive, so is AB.

All the corollaries also generalise.

7.14 Problems

1. Prove that if k is a prime integer then Z_k is an integral domain.

2. Prove that if I is an integral domain, so is $I[x]$.

3. Write $(r^2+s)(r-s+1)$ in $Z[r][s]$.

4. Represent $(r^2+s)(r-s+1)$ in sparse recursive form in $Z[r][s]$.

5. Prove that if a, b, c are in $Z[x]$ and $ab = c$, then $con(a) * con(b) = con(c)$, and $pp(a) * pp(b) = pp(c)$. Use the Gauss lemma, and show where it is used.

6. Show that for a, b in $Z[x]$,

   $GCD(a, b) = GCD_Z(con(a), con(b)) * GCD_{Z[x]}(pp(a), pp(b))$

   Use previous problem.

7.15 GCDs in $Z[x_1, ..., x_n]$

We will look at a number of methods for finding these GCDs. This is a basic problem since it appears to be needed for addition and multiplication in $Z(x_1, ..., x_n)$.

All of these methods follow the same recursive pattern, using the GCD corollary of the previous section. Suppose we have a way to find GCDs in $Z[x_1, ..., x_{n-1}]$. Let $K = Z[x_1, ..., x_{n-1}]$. Suppose given A and B in $K[x_n]$. Find $content(A), content(B), pp(A), pp(B)$, using our existing method for GCDs in K. Then

$GCD(A, B) = GCD(content(A), content(B)) * GCD(pp(A), pp(B))$.

We can find $GCD(content(A), content(B))$ by our existing method. We are left with finding GCD for $pp(A)$ and $pp(B)$. So we need a way to get GCDs for primitive polynomials. There are various ways of doing this.

wolog, in all the following, we assume A and B are primitive.

Assume in the following that $K[x]$ is an integral domain.

7.15.1 Method 1

Given A and B primitive in $K[x_n]$, extend K to field of fractions $Fr(K)$. Find the GCD of A and B in $FR(K)[x_n]$ using the Euclidean algorithm. Suppose this GCD is G. The coefficients of G may be fractions of polynomials in K. Multiply by the least common multiple of the denominators, to clear all denominators, obtaining a polynomials g in $K[x]$. Then take $pp(g)$. This is $GCD(A, B)$ in $K[x_n]$.

Example 4 $A(x) = x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5$
$B(x) = 3x^6 + 5x^4 - 4x^2 - 9x + 21$

A problem is that there may be many recursive GCD computations, and the coefficients may become huge. So we are led to consider the possibility of working directly in K, and not using fractions at all.

7.15.2 Pseudodivision technique

Finding remainders depends on division. So we define something which is like a remainder, but which will not need division which goes outside K.

Theorem 12 *Suppose $p(x)$ and $q(x)$ are in $\mathbf{K}[x]$, and $p(x)$ has degree m and $q(x)$ has degree n. There exist $c(x)$ and $r(x)$ in $\mathbf{K}[x]$ with the degree of $r(x)$ less than the degree of $q(x)$, or $r(x) = 0$, so that*
$lc(q)^{m-n+1} * p(x) = c(x) * q(x) + r(x)$

Definition. 28 *The polynomial $r(x)$ asserted to exist above will be denoted $prem(p, q)$, $prem(p, q)$ will be called the pseudo remainder of p with respect to q.*

Method 2 for calculating gcds in $\mathbf{K}[x]$ is to use pseudoremainders instead of remainders in Euclid's algorithm.

————————-

Definition. 29 *The Pseudo Euclidean Algorithm for GCDs Assume A and B are primitive in $K[x]$.*
$GCD(A, B)$ is defined by:

$while(B \neq 0)(R := prem(A, B), A := B, B := R)$ *Then* $GCD = pp(A)$.

————————

A problem with this is that the contents get huge. Try it on the previous example in $Z[x]$. Especially annoying is the fact that we do not want the contents. We are trying to get the GCD of primitive polynomials, and so we will end up with a primitive GCD.

Method 3 is to remove the contents at each stage. This has lots of recursive GCD computations.

7.15.3 Polynomial remainder sequences

We need some notation. We assume that $A(x)$ and $B(x)$ are in $K[x]$ and that the degree of $A(x)$ is at least as large as the degree of $B(x)$. Define a *polynomial remainder sequence* for A and B to be a sequence of polynomials $R_0, R_1, R_2, ..., R_j$ in $K[x]$ satisfying:

1. $R_0 = A, R_1 = B$
2. For some α_i, Q_i, β_i in $K[x]$,
   $\alpha_i R_{i-1} = Q_i R_i + \beta_i R_{i+1}$
3. $prem(R_{j-1}, R_j) = 0$

For the Euclidean polynomial remainder sequence using pseudo-remainders, $\alpha_i = r_i^{\delta_i+1}, r_i = lc(R_i), \delta_i = deg(R_{i-1}) - deg(R_i), \beta_i = 1$.

For the primitive polynomial remainder sequence, $\alpha_i = r_i^{\delta_i+1}, \beta_i = content(prem(R_{i-1}, R_i))$.

A good deal of work has been done to try to predict part of the content of the polynomials in the sequence, without actually having to calculate all the recursive GCDs. If you try some experiments, you will see that the multipliers (the r_i) often turn up later on in the contents. In fact, it can be shown that in the Euclidean sequence, we can always find a factor of r_{i-1} in R_{i+1}. **Method 4** is to divide this out, giving us a reduced polynomial remainder sequence:

$\alpha_i = r_i^{\delta_i+1}, \beta_1 = 1, \beta_i = \alpha_{i-1}$ for $2 \leq i \leq j$.

Quite often more than one factor of this kind appears. **Method 5**, the trial divisor method, reduces the content by as large a power of r_{i-1} as divides evenly.

The ultimate thing of this kind is the **subresultant polynomial remainder sequence** which looks like this:

$\alpha_i = r_i^{\delta_i+1}, \beta_1 = (-1)^{\delta_i+1}, \beta_i = -r_{i-1}\Psi_i^{\delta_i}$ for $2 \leq i \leq j$,

with Ψ_i defined by:

$\Psi_1 = -1$

$\Psi_i = (-r_{i-1})^{\delta_{i-1}} \Psi_{i-1}^{1-\delta_{i-1}}$ for $2 \leq i \leq j$.

8 Numerical Solving by Newton's Method

9 Newton's Method

In this section $F = (f_1, ..., f_n) : \mathbf{C}^n \to \mathbf{C}^n$, all the partial derivatives of all orders are assumed to exist, $x = (x_1, ..., x_n)$, and $J(x)$ is defined to be the n by n matrix of partial derivatives of $F(x)$, i.e.

$J(x) = (\partial f_i / \partial x_j)$.

$J(x)$ is called the Jacobian of $F(x)$.

The Jacobian, $J(x)$, is an n dimensional derivative. It specifies the "slope" of the best linear approximation to $F(x)$ at a point.

$F(x) = F(x_0) + J(x_0)(x - x_0) + O(| x - x_0 |^2)$.

The linear function $F(x_0) + J(x_0)(x - x_0)$ is called the linearisation of the function $F(x)$ at x_0.

A solution point of n equations in n unknowns, $F(x) = 0$, is said to be non singular if the Jacobian is non singular at that point.

Suppose we want to solve $F(x) = 0$ and we have an initial approximation $x = a$ to a solution. Newton's method constructs a sequence of approximations to a solution by linearising the function at each point, and solving the linearised equation. The recurrence obtained is: $x_0 = a$, x_{i+1} satisfies

$$F(x_i) + J(x_i)(x_{i+1} - x_i) = 0$$

If $J(x_i)$ is non singular, then:

$$x_{i+1} = x_i - J^{-1}(x_i)(F(x_i))$$

This works nicely if our original guess x_0 is sufficiently near a non singular solution.

Here is a famous theorem.

Theorem 13 *Inverse Function Theorem. If $F(x) = y$ and the Jacobian is non singular at x, then F has an inverse in some neighbourhood of y.*

And here is a famous conjecture.

Conjecture 2 *The Jacobian Conjecture *****. In either R^n or C^n, if F is a polynomial map and the Jacobian of F is never singular, then F is 1-1 and onto.*

and another famous theorem:

Theorem 14 *If $F(a) = 0$ and the Jacobian is continuous and non singular at $x = a$, there is a neighbourhood of a in which Newton's method converges to the solution a no matter what initial point is chosen in the neighbourhood.*

9.1 Interval Arithmetic

We can define an arithmetic of intervals as follows.

Definition. 30 $[a, b] + [c, d] = [a + c, b + d]$

$[a, b] * [c, d] = [m, M]$, *where m and M are the min and max respectively of (ac, ad, bc, bd).*

We can represent numbers, r, as $[r, r]$.

So $[a, b] - [c, d]$ should be $[a, b] + [-1, -1] * [c, d]$.

So far we have:

Lemma 4 $\{xopy : x \in [a, b], y \in [c, d]\} \subseteq [a, b]op[c, d]$

with op being any of $+, -, *$.

The same inclusion property holds for polynomials. We would like to make these intervals smaller if possible, while keeping the inclusion. So, for example, it is better to define $[a, b]^3$ as $[a^3, b^3]$, rather than as $[a, b] * [a, b] * [a, b]$ as above. It seems that the right way to define $[a, b]^2$ depends on the signs of a and b...

Definition. 31 *A Q-box in R^n is a closed coordinate aligned box whose vertices have coordinates which are either $-\infty$ or ∞ or rational numbers, expressed as bounded precision floats.*

9.2 Validated numerical methods

Interval arithmetic can be used in solving systems of equations. If f is a function of n variables with values in R^m, and *box* is a box in R^n we will write $f(box)$ to be the box in R^m obtained by the interval technique discussed above. So we are guaranteed that the image of *box* under f is contained in $f(box)$. This already gives us a powerful method to exclude the possibility that $f(x) = 0$ has a solution in *box*.

0 not in $f(box) \to (\neg(\exists x \in box)f(x) = 0$

This simple idea can be used with recursive subdivision of the box to make an elegant solver. Suppose we have a condition $small(box)$ so that if this condition holds and $0 \in f(box)$, then we would accept *box* as an approximate solution.

The following algorithm either returns one such *box* or it returns the empty set, in which case there is no solution in the box.

Algorithm $solve(f = 0, box)$

1. if 0 is not in $f(box)$, then return $\emptyset$
2. if 0 is in $f(box)$ and $small(box)$, then return box
3. Otherwise, subdivide box into $box1$ and $box2$, splitting at the middle of a side of maximal length. Return $solve(f = 0, box1)$ if this is non empty. Otherwise, return $solve(f = 0, box2)$.

Suppose we measure the length of a vector, v, of complex numbers by taking the square root of the dot product of v and $conjugate(v)$.

If A is a matrix of complex numbers, let $\|A\|$ be the maximum of the lengths of the rows of A.

We have

$$| Aw | \leq \|A\| \mid w \mid n$$

,

where A is square n by n and w is a vector of length n, with complex entries.

We can also use interval arithmetic to verify convergence of Newton's method.

Theorem 15 *Suppose box is a box in C^n. Let $newbox$ be $(x - J^{-1}(x)F(x))(box)$.*
Suppose:

1. *$newbox \subset box$*
2. *For x and y in box, $\|I - J(x)J^{-1}(y)\| < 1/2n$*

Then the Newton sequence started at any $x_0 \in box$ converges to a solution in box of $F(x) = 0$.

Proof.
The iteration
$x_{n+1} = x_n - J^{-1}(x_n)F(x_n)$
stays inside of box, since $newbox \subset box$. Consider one step.
Draw a straight line, L, from x_n to x_{n+1}, and consider what happens to $F(x)$ on this line.
$F(x_{n+1}) = F(x_n) + \int_L J(x)dx$
Parameterise $x(t) = x_n(1 - t) + x_{n+1}t$, for $0 \leq t \leq 1$.
$dx = -J^{-1}(x_n)F(x_n)dt$
$F(x_{n+1}) = F(x_n) + \int_0^1 (J(x_n) + (J(x(t)) - J(x_n))(-J^{-1}(x_n)F(x_n))dt$
$\leq Max \mid I - J(x)J^{-1}(x_n) \mid\mid F(x_n) \mid$, for x on the line L.
Thus, by the second condition of the premise of the theorem,
$\mid F(x_{n+1}) \mid < 1/2 \mid F(x_n) \mid$.
This implies that the norm of $F(x_n)$ tends to zero as the iteration proceeds, and thus the difference between x_{n+1} and x_n tends to zero. Thus the Newton sequence converges.
$\square$

We can prove convergence of the Newton sequence by verifying the two conditions using interval arithmetic.

There are a number of other tests to verify Newton convergence. In conclusion, we have a powerful technique for solving n equations in n unknowns, provided that we begin with a good approximation to a non singular solution.

9.3 Coursework 2: Due Friday May 11

Hand in to Mathematics Office, before 4 pm. Write and test a procedure for Newton's method in complex n space. The user should give

1. A vector α_0 in C^n or R^n
2. A vector $F(x) = (f_1(x), \ldots, f_n(x))$, with $f_i(x) = f_i(x_1, \ldots, x_n) : C^n \to C$, and $F(x) : C^n \to C^n$.
3. A precision k. (Any number ϵ, with $\mid \epsilon \mid < 10^{-k}$ will be regarded as insignificantly small.)
4. A maximum number of iterations

The procedure should return either an $\alpha \in C^n$ so that $\mid F(\alpha) \mid < 10^{-k}$, or a helpful error message of some kind.

Marking Scheme:

1. Design of procedure (20)
2. Simplicity and correctness of code (50)
3. Testing (30)

Here are some examples which can be used for testing, with various starting points.

- $x^3 - 1 : C \to C$. Try randomly chosen initial points in unit circle around 0 in C.
- $(x_1^3 - 1, x_2^3 - x_1, ..., x_n^3 - x_{n-1}) : C^n \to C^n$
- $(e^{xz} - 2 * y * w, e^{xw} - 3, e^{yz} - 7, e^{yw} - 11) : C^4 \to C^4$
- $x^4 + x^3 - x - 1 : C \to C$.

10 Factorisation in$Z$ and in $Z[x]$

10.1 The Tau conjecture

A computation of a number m in Z is a sequence

$x_0, \ldots, x_l$

with $x_0 = 1$ and $x_l = m$, and, for each i, x_i is obtained from two previous elements in the sequence by either addition, subtraction or multiplication. Define $\tau(m)$ to be the length of the shortest computation which ends in m.

Conjecture 3 *The τ conjecture. There is no constant C so that*
$\tau(k!) \leq (\log k)^C$ *for $k \in N$*

If the τ conjecture is false, i.e. if there is a quick way to compute $k!$, then there is also a quick way to factorise integers.

If $n = pq$ with $p < k < q$, compute $k!$ by quick computation

$x_0, ..., x_l$

and then get a computation of $k!\ modn$ by

$x_0\ modn, ..., x_l\ modn = r$

and then take $GCD(r, n)$ to get a factor of n.

See *Complexity and Real computation*, Blum, Shub, Smale, Cucker, Springer, (510.095.9).

10.2 Polynomial factorisation

Height of polynomial. Landau Mignotte bound on height of factors. To factor $p(x)$ in $Z[x]$, find a root α, approximate with Newton's method. Then use PSLQ to find a minimum polynomial $q(x)$ defining α. Then $q(x)$ is irreducible and $q(x)$ divides $p(x)$.