Mathematics 1263: introduction to numerical analysis, Michaelmas 2014

Colm Ó Dúnlaing

March 31, 2015

1 GCD

(1.1) Definition *The greatest common divisor (gcd), also called the highest common factor (hcf), of two integers m and n, is the largest positive integer which divides both m and n. It is undefined if $m = n = 0$.*

One writes $\gcd(m, n)$ for the gcd of m and n.

(1.2) Proposition ('division algorithm'). *Let d be a* positive *integer*[1] *For* every *integer n there exist integers q and r such that*

- $n = qd + r$
- $0 \leq r \leq d - 1$

Also, q and r are unique.

We call q and r the *quotient* and *remainder* on dividing n by d, or the remainder of n modulo d. We write

$$n = qd + r$$
$$q = n \div d$$
$$r = (n \mod d)$$

For example, $100 = 7 \times 13 + 9$: with $d = 13$, $n = 100$, the quotient $100 \div 13$ is 7 and the remainder is 9.

(1.3) Lemma (i) $\gcd(m, n) = \gcd(n, m)$
(ii) *If $m > 0$ and $n = 0$, then* $\gcd(m, n) = m$.
(iii) *If $m \geq n > 0$, then* $\gcd(m, n) = \gcd(n, m \mod n)$

[1] The definition can be extended to negative integers d, but it is tedious and seldom used.

For example,

$$\gcd(100, 13) = \gcd(13, 9) = \gcd(9, 4) = \gcd(4, 1) = \gcd(1, 0) = 1.$$

This gives an efficient way to calculate the gcd of two numbers. Assume $m > n > 0$.

$$x_0 = m; x_1 = n; x_2 = x_0 \mod x_1; x_3 = x_1 \mod x_2; \ldots$$

Continue until $x_{k+1} = 0$; then x_k is the gcd.

1.1 A digression

We can get an interesting (not for the purposes of this course) modification of the GCD algorithm.

The general step is

$$x_{k+1} = x_{k-1} \mod x_k$$

If we break this down into two steps

$$q_k = x_{k-1} \div x_k \qquad x_{k+1} = x_{k-1} - q_k x_k$$

we can apply the same combination

$$r_{k-1} - q_k r_k$$

to other sequences of numbers. In particular if we build the sequences $r_0, r_1, \ldots$ and $s_0, s_1, \ldots$ by setting

$$r_0 = 1, r_1 = 0, s_0 = 0, s_1 = 1$$

we get an interesting effect.

i	x_i	r_i	s_i	q_i
0	100	1	0	–
1	13	0	1	7
2	9	1	-7	1
3	4	-1	8	2
4	1	3	-23	4
5	0	–	–	–

You can check that the following 'invariant condition' holds throughout the table:

$$x_i = m \times r_i + n \times s_i$$

In particular,

$$\gcd(100, 13) = 3 \times 100 - 23 \times 13.$$

In view of this, one can make an alternative definition of the gcd:

(1.4) Proposition *Given two integers m and n, with (for simplicity) $m \geq n > 0$,* $\gcd(m, n)$ *is the* smallest positive integer *of the form $rm + sn$ where r and s can be any integers.*

2 Sturm's Theorem

(2.1) Definition *A* **Polynomial in** x *with integer/rational/real/complex coefficients is an expression of the form*

$$a_0 + a_1 x + \ldots + a_n x^n,$$

where $n \geq 0$, $a_n \neq 0$ *except when* $n = 0$ *and* $a_0 = 0$, *and the* **coefficients** a_j *are integer/rational/real/complex numbers.*

If $n = 0$ *so the polynomial is just* a_0, *we identify it with the constant* a_0. *The* zero polynomial *corresponds to the constant* 0.

The **degree** *of the zero polynomial is* $-\infty$. *Otherwise the degree is* n.

We write $\deg(p)$ *for the degree of the polynomial* p.

(2.2) Proposition (Division algorithm.) *If* p *and* d *are polyomials and* $d \neq 0$, *with coefficients in a* field[2] *then there exist unique polynomials* q *and* r, *such that*

- $p = qd + r$ *and*
- $\deg(r) < \deg(d)$.

We are only interested in polynomials with real coefficients.

We call q the **quotient** and r the **remainder** on dividing p by d. If the remainder is zero, we say that d divides p exactly.

Scaling by a constant $x + 1$ divides $x^2 - 1$; so does $2x + 2$. For any nonzero polynomial p, and any nonzero real number c, cp divides p and p divides cp. In this way the gcd is not fully defined. We can standardise it by requiring that it be *monic*: the highest degree term is x^n for some n.

If $f(x)$ is a function and α a real or complex number such that $f(\alpha) = 0$, we call α a *zero* of the function f, or a *root* of the equation $f(x) = 0$, or more loosely, a *root* of f.

GCD computations. Wherever you have a division algorithm, you have a GCD algorithm, more or less. Euclid's algorithm produced an integer GCD: the same method produces polynomial GCDs as well. For example,

$$a = x^4 + 2x^3 - x^2 - 3x - 1 \qquad \text{and} \quad b = x^3 + x^2 - 3x - 3,$$

$$\begin{aligned} x^4 + 2x^3 - 1x^2 - 3x - 1 &= (x+1)(x^3 + x^2 - 3x - 3) + (x^2 + 3x + 2) \\ x^3 + x^2 - 3x - 3 &= (x-2)(x^2 + 3x + 2) + (x+1) \\ x^2 + 3x + 2 &= (x+2)(x+1) + (0) \end{aligned}$$

The gcd is $x + 1$. Below the various polynomials are tabulated.

f	g	h	q
$x^4 + 2x^3 - 1x^2 - 3x - 1$	$x^3 + x^2 - 3x - 3$	$x^2 + 3x + 2$	$x + 1$
$x^3 + x^2 - 3x - 3$	$x^2 + 3x + 2$	$x + 1$	$x - 2$
$x^2 + 3x + 2$	$x + 1$	0	$x + 2$
$x + 1$	0	$-$	$-$

[2] For example, it won't work with integer coefficients

(2.3) Definition *Let $p(x)$ be a polynomial, α a root, so $(x-\alpha)$ divides $p(x)$. If no higher power of $(x-\alpha)$ divides p, then α is a* simple *root of p.*

(2.4) Lemma *Let p be a nonzero polynomial and $g = \gcd(p, p')$. Then the roots of p and p/g are the same, but the roots of p/g are simple.*

In particular, all roots ofp are simple if and only if $\gcd(p, p') = 1$. (No proof) ∎

Sturm sequences. If p is a nonzero simple real polynomial, of degree $n > 0$, the *Sturm Sequence* is the sequence

$$p_0, p_1, \ldots p_k$$

where $p_0 = p$, $p_1 = p'$, and for $j = 1, \ldots, k-1$,

$$p_{j-1} = p_j q_j - p_{j+1},$$

where q_j is the quotient, p_{j+1} is the remainder negatived, and p_k is constant.

This is a GCD computation, **except for the signs of the remainders.**

A Sturm sequence is studied for sign changes (as described below), and

> any member of the sequence can be scaled by a positive constant without changing the pattern of sign changes.

For example, if $p(x) = x^5 - 5x + 1$ then $p'(x) = 5(x^4 - 1)$ and we get the Sturm sequence (note $p_1 = p'/5$ for convenience)

$$x^5 - 5x + 1, x^4 - 1, 4x - 1, 1.$$

Actually the last term is $1 - 1/256$, but it is positive.

(2.5) Lemma *Suppose that $p_i(x),\ 0 \le i \le k$, is a Sturm sequence. Then no two consecutive polynomials in the sequence can vanish at the same point.*

In other words, if α is a real number such that $p_i(\alpha) = 0$ for some i, then $i < k$ and $p_{i+1}(\alpha) \neq 0$.

Proof. Suppose otherwise, so $p_i(\alpha) = p_{i+1}(\alpha) = 0$. The Sturm sequence imitates the sequence of polynomials developed in calculating $\gcd(p, p')$ (the polynomials in each sequence are proportional). Since $\gcd(p, p') = 1$, $i < k$. Since $p_i(\alpha) = p_{i+1}(\alpha) = 0$, $x - \alpha$ divides $p_i(x)$ and $p_{i+1}(x)$; then it divides $p_j(x)$ for every $j \ge i$, and $p_k(x)$ is not constant, a contradiction. ∎

(2.6) Theorem (Sturm's Theorem.) *Suppose that p is a non-constant polynomial and gcd$(p, p') = 1$. Let $a < b$ be two real numbers which are not roots of p. Then the number of real roots between a and b equals the number of sign changes* lost *between a and b.*

In the above example, as $a \to -\infty$ and $b \to \infty$ the sign patterns are

$$- + - +$$

and $+ + + +$, so there are exactly three real roots.

Proof of Sturm's Theorem. Let $p_0, p_1, \ldots, p_k$ be the Sturm sequence.

Imagine a variable z moving from a to b. We are interested in what happens at a point z where $p_i(z) = 0$ for some i. By Lemma 2.5, $i < k$, $p_{i+1}(z) \neq 0$, and, if $i > 0$, $p_{i-1}(z) \neq 0$.

If $i > 0$, then $p_{i-1}(z)$ and $p_{i+1}(z)$ are nonzero with opposite signs, and $p_i(x)$ changes sign at z. If ϵ is sufficiently small, p_i changes sign only once between $z - \epsilon$ and $z + \epsilon$, and the other two don't change sign in that interval.

There are four possible patterns for sign changes:

$p_{i-1}(z-\epsilon)$	$p_i(z-\epsilon)$	$p_{i+1}(z-\epsilon)$	$p_{i-1}(z)$	$p_i(z)$	$p_{i+1}(z)$	$p_{i-1}(z+\epsilon)$	$p_i(z+\epsilon)$	$p_{i+1}(z+\epsilon)$
$+$	$+$	$-$	$+$	0	$-$	$+$	$-$	$-$
$+$	$-$	$-$	$+$	0	$-$	$+$	$+$	$-$
$-$	$+$	$+$	$-$	0	$+$	$-$	$-$	$+$
$-$	$-$	$+$	$-$	0	$+$	$-$	$+$	$+$

There is one sign change before, and there is one after. The number of sign changes does *not* change.

On the other hand, if z is a root of $p(x)$, then either $p(x)$ crosses the x-axis from below, in which case $p'(z) > 0$, or it crosses the x-axis from above, in which case $p'(z) < 0$. So the possibilities are:

$p_0(z-\epsilon)$	$p_1(z-\epsilon)$	$p_0(z)$	$p_1(z)$	$p_0(z+\epsilon)$	$p_1(z+\epsilon)$
$-$	$+$	0	$+$	$+$	$+$
$+$	$-$	0	$-$	$-$	$-$

Thus in this case alone, the number of sign changes *decreases by one.* ∎

Example. This example is hard to work through by hand, but here are the results.

Polynomial and its derivative:

$$2x^5 + 3x^4 - 2.6x^3 + 4x^2 - 4.6x + 1$$
$$10x^4 + 12x^3 - 7.8x^2 + 8x - 4.6$$

Sturm sequence (adjusted so highest-degree coefficient is ± 1)

$$x^5 + 1.5x^4 - 1.3x^3 + 2x^2 - 2.3x + 0.5$$
$$x^4 + 1.2x^3 - 0.78x^2 + 0.8x - 0.46$$
$$x^3 - 1.62955x^2 + 2.36364x - 0.725$$
$$-x^2 + 3.51887x - 1.08464$$
$$-x + 0.34996$$
$$-1$$

```
4 sign changes at -infty: -+-+--
4 sign changes at -5.000000: -+--+-
4 sign changes at -4.000000: -+--+-
4 sign changes at -3.000000: -+--+-
3 sign changes at -2.000000: ++--+-
3 sign changes at -1.000000: +---+-
3 sign changes at 0.000000: +---+-
1 sign changes at 1.000000: ++++--
1 sign changes at 2.000000: ++++--
1 sign changes at 3.000000: ++++--
1 sign changes at 4.000000: +++---
1 sign changes at 5.000000: +++---
1 sign changes at +infty: +++---
```

Conclusion: there are three real roots, one between -3 and -2, and two between 0 and 1.

3 Newton-Raphson

The Newton-Raphson method calculates, very efficiently, approximations to simple roots (simple zeroes) of a differentiable function f.

The idea is simple: if a is close to a root, then the tangent line to the graph at a is a good approximation to f near a, and, almost certainly, it cuts the x-axis at a point closer to the root.

The tangent line to the graph at a — more correctly, at $(a, f(a))$ — has the equation

$$y = f(a) + (x - a)f'(a)$$

Obviously, if $f'(a) = 0$ we're stuck. Otherwise it crosses the x-axis at

$$f(a) + (x - a)f'(a) = 0$$
$$\frac{f(a)}{f'(a)} = a - x$$
$$x = a - \frac{f(a)}{f'(a)}$$

So

The Newton-Raphson method develops a sequence $a_0, a_1, a_2, \ldots$ where a_0 is suitably chosen, and for $j = 1, 2, \ldots$

$$a_{j+1} = a_j - \frac{f(a_j)}{f'(a_j)}$$

Example $f(x) = x^2 - 2$.

```
a = 1
a=a/2+1/a; a
1.50000000000000000000
a = a/2+1/a; a
1.41666666666666666666
a=a/2+1/a; a
1.41421568627450980392
a=a/2+1/a; a
1.41421356237468991062
a=a/2+1/a; a
1.41421356237309504880
a=a/2+1/a; a
1.41421356237309504880
```

3.1 Rate of convergence of the Newton-Raphson method.

Suppose that $|a_j - r| = \epsilon$, where ϵ is small and r is a nearby (simple) root.

The following estimate is Taylor's Theorem with $n = 1$.

$$f(x) = f(a) + (x - a)f'(a) + \frac{(x - a)^2}{2!}f''(X)$$

where X is somewhere between a and x. Now, if x is the root we're approximating, $f(r) = 0$; and if a is one of the approximations a_j, we divide by $f'(a_j)$ and get

$$\frac{f(a_j)}{f'(a_j)} + r - a_j + (r - a_j)^2 \frac{f''(X)}{2f'(a_j)} = 0$$

and substituting a_{j+1} where it fits, we get

$$r - a_{j+1} = M(r - a_j)^2$$

where M is related to $f'(x)$ and $f''(x)$. Usually we can fix an upper bound on M (in absolute value). The point is,

Roughly speaking, the error in a_{j+1} is the square of the error in a_j.

— generally speaking, the Newton-Raphson method produces accurate results very quickly.

4 Floating-point format

Decimal points work this way:

$$123.456 = 1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 + 4 \times 10^{-1} + 5 \times 10^{-2} + 6 \times 10^{-3}.$$

Such numbers are called *fixed-point decimal* to distinguish them from floating point or scientific notation.

We can allow 'binary points' with a similar aim. As *binary numbers*:

$$101.011 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} + 1 \times 2^{-3}.$$

Or (in decimal) $4 + 1 + 1/4 + 1/8 = 5\,3/8 = 5.375$.

In science and engineering, the accuracy of measurement is taken as a proportion. For example, to measure a the radius of a golfball correct to the nearest centimetre is not impressive, but to measure the radius of the earth to the nearest centimetre, is.

In science and engineering, numbers are often given in *scientific notation.* For example, the speed of light is about 186,000 miles per second. In scientific notation, this would be represented as

$$1.86 \times 10^5$$

(or something like $1.86E5$). The *significand* is 1.86 and the *exponent* is 5.

There are *sign, significand* (at least 1 and less than 10), and exponent.

FAQ: You say that numbers should be represented this way. What about zero?

Answer. Zero is an exception. Zero is the only number which cannot be represented by sign, significand, and exponent.

On computers, this idea is adapted to the binary system. Thus

$$101.011 = 1.01011 \times 2^3.$$

On the left, we have fixed-point binary, on the right we have floating-point binary.

- **Sign**
- **Significand** at least 1 and less than 2.
- **Exponent.**
- **Zero** is an exception.

These are called *floating-point numbers.*

Computing a binary significand by hand. The significand has the form (zero an exception) $1.a_1a_2a_3\ldots$ where the figures a_j are *binary digits* (called 'bits' for short).

The trick is to 'pull' the number to the left by repeated doubling. For example, let us try $4/3$.

$$\frac{4}{3} = 1 + 1/3 \ldots 1. \quad \text{Drop the 1 and double}$$
$$2/3 = 0 + 2/3$$
$$4/3 = 1 + 1/3$$
$$2/3 = 0 + 2/3$$
$$1.010101\ldots$$

How can we check this? Summing a geometric series

$$1.010101\ldots = 1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64}\ldots = \frac{1}{1-1/4} = \frac{4}{3}.$$

We're reckoning fractions as fixed-point binary. The answer is always a finite or recurring binary pattern.

Here's a messier example. To compute $20/11$ as binary fixed-point.

```
20 18 14    12          16    20

 9  7  3  6  1  2  4  8  5 10  9
```

Idea. Keep doubling the number. If the result is ≥ 11, it goes on the top line, and one subtracts 11 to get the figure below it. If the result is < 11, it goes directly on the bottom line.

Note that the pattern recurs (it recurs at the original number, 20, but the recurring pattern could start later in the expansions).

In the binary expansion, where there's something on the top line, there's a 1 in the binary expansion; where the top entry is blank, there's a 0.

```
 1. 1  1  0  1  0  0  0  1  0  1  1  1  0  1  0  0  0  ....

20 18 14    12          16    20

 9  7  3  6  1  2  4  8  5 10  9
```

As a recurring pattern:

$$1.(1101000101)^*$$

As a geometric series

$$1 + 837 \times \left(\frac{1}{1024} + \frac{1}{2^{20}} \ldots \right) = 1 + \frac{837}{1023} = \frac{20}{11}.$$

4.1 Computing binary significands, last time.

Given a number x (which might even be irrational) with $1 \leq x < 2$, in binary it is

$$a_0.a_1a_2a_3\ldots$$

and of course $a_0 = 1$.

The digits a_j are calculated as follows.

$$\begin{gathered} x_0 = x \\ \text{for } j = 1, 2, \ldots \\ a_j = [x_j] \quad \text{(integer part of } x_j\text{)} \\ x_{j+1} = 2 \times (x_j - a_j) \end{gathered}$$

4.2 Floating point numbers on computer.

There are *single precision*, 32 bits long, and *double precision*, 64 bits long. The data is split up differently for the different precisions.

(4.1) Definition *Suppose a nonzero floating-point number has significand* $1.a_1a_2\ldots a_k$. *The bit-string* $a_1\ldots a_k$ *is called the* mantissa.

Since the first bit in the significand of a nonzero number is always 1, it is omitted from the mantissa.

The breakdown is as follows.

Single precision: $1 + 8 + 23$.

Double precision: $1 + 11 + 52$.

This means: 1 bit for sign 8 (resp., 11) for exponent, and 23 (resp., 52) for mantissa. This gives precision of about 6 (respectively, 15) decimal places.

4.3 Sign bit

Simply: 0 for positive, 1 for negative.

4.4 Exponent

8-bit binary numbers range from 0 to 255, and 11-bit binary numbers range from 0 to 2047, 'face-value.' The range is adjusted by subtracting the 'bias,' 127 or 1023.

size	subnormal/zero	negative	zero	positive	infinite/NaN
8-bit biased	0	$1 \dots 126$	127	$128 \dots 254$	255
less bias	-127	$-126 \dots -1$	0	$1 \dots 127$	128
11-bit biased	0	$1 \dots 1022$	1023	$1024 \dots 2046$	2047
less bias	-1023	$-1022 \dots -1$	0	$1 \dots 1023$	1024

4.5 Rounding

Suppose x is a nonzero real number with significand

$$1.a_1a_2 \dots a_{23}a_{24} \dots$$

probably with infinitely many digits. There are several schemes for rounding when calculating the mantissa. The simplest is 'round towards zero':

mantissa $a_1a_2 \dots a_{23}$.

Slightly better, and this is the scheme ordinarily used, is to 'round.' It is equivalent to:

If $a_{24} = 0$, round towards zero.

If $a_{24} = 1$, add 2^{-23}. This is equivalent to adding 1 to the mantissa, if the mantissa is interpreted at face value.

Example. Calculate the single-precision floating point representation of $-5/22$.

The sign bit is 1.

The absolute value is less than 1. Multiply it by a power of 2 to get it into the range $1 \dots 2$.

$5/22 \times 2 \times 2 \times 2 = 20/11$.

$5/22 = 2^{-3} \times 20/11$.

$20/11$ will give the significand. It has already been computed.

$$1.(1101000101)^*$$

so the mantissa is

$$1101000101\ 1101000101\ 110\ 100 \dots$$

So $a_{24} = 1$. Add 1 to the first 23 bits.

```
    1101 0001 0111 0100 0101 110
+                              1
--------------------------------
    1101 0001 0111 0100 0101 111
```

The true exponent is -3. The biased exponent is 124. Subtract 3 (binary 11) from 127 ($0111\ 1111$) getting $0111\ 1100$.

Combining all of these together,

```
1 0111 1100 1101 0001 0111 0100 0101 111
```

Put them in groups of four.

```
1011 1110 0110 1000 1011 1010 0010 1111
```

4.6 Hexadecimal numbers

Hexadecimal numbers are to base 16, and are used for a more compact representation of binary numbers.

The 'hex digits' are $0 \ldots 9$, and $a \ldots f$ for $10 \ldots 15$.

0000	0	0100	4	1000	8	1100	c
0001	1	0101	5	1001	9	1101	d
0010	2	0110	6	1010	a	1110	e
0011	3	0111	7	1011	b	1111	f

This gives

```
1011 1110 0110 1000 1011 1010 0010 1111
   b    e    6    8    b    a    2    f
```

4.7 Little endian

One more detail — which is not important — is that on Intel computers the numbers are stored 'little endian'. Computer memory is a collection of *bytes,* each byte being 8 bits or 2 hex digits. In 'little endian' form, the *bytes* are stored in reverse order (not the hex digits, though), giving

$$2f\ ba\ 68\ be$$

Summarising

```
-5/22 = 2^{-3} x 20/11
negative: sign 1
exponent -3   biased 0111 1100
mantissa
    1101 0001 0111 0100 0101 110 1 ... round up
result 23 ba 68 be hex, little endian.
```

Converting the same number to double precision is not much harder.

```
sign 1
exponent -3 biased 011 1111 1100
mantissa
1101 0001 0111 0100 0101 1101 0001 0111 0100 0101 1101 0001 0111... 0100 0101
round down
1101 0001 0111 0100 0101 1101 0001 0111 0100 0101 1101 0001 0111
combined
1011 1111 1100 1101 0001 0111 0100 0101 1101 0001 0111 0100 0101 1101 0001 0111
   b    f    c    d    1    7    4    5    d    1    7    4    5    d    1    7
little endian
17 5d 74 d1 45 17 cd bf
```

5 IEEE standard

The IEEE standard for binary floating-point arithmetic was produced by group p754 in 1985. It was revised in the more general p854 in 1987.

Recall that the single-precision format is

sign 1 bit, exponent 8 bits, mantissa 23 bits

and double-precision

sign 1 bit, exponent 11 bits, mantissa 52 bits

About 6.9 and 15.6 decimal places. Single-precision is seldom used, but we'll focus on it.

The range of biased exponents is 0..255, so the true exponent range is

$$-127 \text{ to } 128.$$

Exponents -127 and 128 have special meanings

-127: ± 0 if the mantissa is zero, otherwise *subnormal numbers* which will not concern us.

128: $\pm\infty$ if the mantissa is zero, otherwise NaN ('not a number.')

(5.1) Definition *A* floating-point *number (single-precision) is any number of the form*

$$\pm b_0.b_1 \ldots b_{23} \times 2^e$$

where the b_i are bits (binary digits) and $-126 \le e \le 127$, and if nonzero, $b_0 = 1$ (the hidden bit).

We say that single-precision floating-point numbers have 24 bits of precision, allowing for the hidden bit. Single-precision machine accuracy, machine-epsilon ϵ_{mach}, is defined as follows

$$1 + \epsilon_{\text{mach}}$$

is the smallest floating-point number > 1. I.e.,

$$1.0000\ 0000\ 0000\ 0000\ 0000\ 001$$

in binary, i.e., $\epsilon_{\text{mach}} = 2^{-23}$

Also, N_{min} is the smallest positive floating-point number and N_{max} the largest:

$$N_{\text{min}} = 1.7 \times 10^{-38}$$
$$N_{\text{max}} = 1.7 \times 10^{38}$$

approximately

A nonzero real number x is in normalised floating-point range *if*

$$N_{\min} \leq |x| \leq N_{\max}.$$

5.1 Rounding

Given a real number x, not a floating-point number, let x_- and x_+ be the nearest floating-point numbers to x, so

$$x_- < x < x_+.$$

These nearest numbers could be zero or infinite, but we're most concerned with numbers in normalised floating-point range. With little loss of generality, x is positive in normalised floating-point range. In this case, write x as

$$2^e \times 1.b_1b_2\ldots b_{23} \mid b_{24}b_{25}\ldots$$

Obviously,

$$x_- = 2^e \times 1.b_1b_2\ldots b_{23}$$

and

$$x_+ = 2^e \times (1.b_1b_2\ldots b_{23} + 2^{-23})$$

There are various rounding rules: round up, round down, round to zero, round to nearest. The last is the default and will concern us.

Round to nearest. If x is a floating-point number then $\text{round}(x) = x$. Ignoring the possibilities of rounding to infinity, if x is positive,

$$\text{round}(x) = \begin{cases} x_- & \text{if } x_- < x < (x_- + x_+)/2 \\ x_+ & \text{if } (x_- + x_+)/2 < x < x_+ \\ x_- \text{ or } x_+ & \text{if } x = (x_- + x_+)/2. \end{cases}$$

The last is ambiguous, and there is a **tie-breaking rule:**

When $x = (x_- + x_+)/2$, one of x_-, x_+ has 0 in the low-order position of the mantissa (least significant bit), and the other has 1; choose the one with 0 in the low-order position of the mantissa.

If x is negative, apply the rounding rule to $|x|$.

(5.2) Definition *If $x \neq 0$ then its* relative rounding error *is*

$$\left| \frac{x - \text{round}(x)}{x} \right|.$$

If $x \neq 0$ is in normalised range, so

$$x = \pm 2^e 1.b_1 \dots b_{24} b_{25} \dots$$

Suppose that $|x|$ rounds down: round$(|x|) < |x|$. Then the relative rounding error is

$$2^{-24} \frac{0.b_{24} b_{25} \dots}{1.b_1 b_2 \dots} \leq 2^{-24}$$

If $|x|$ rounds up, the relative rounding error is

$$2^{-24} \frac{1 - 0.b_{24} b_{25} \dots}{1.b_1 b_2 \dots} \leq 2^{-24}$$

Thus:

(5.3) Proposition round to nearest *rounds to relative error at most* $\epsilon_{\text{mach}}/2$.

5.2 The IEEE requirement.

The sum of two single-precision numbers need not be a single-precision number. For example, $1 + 2^{-24}$ is not.

```
 1.0000 0000 0000 0000 0000 000     x 2^0
+1.0000 0000 0000 0000 0000 000     x 2^{-24}
--------------------------------
 1.0000 0000 0000 0000 0000 000 1   x 2^0
```

But the machine is obliged to produce a floating-point answer. Write $\oplus, \ominus, \otimes, \oslash$ for the machine's result.

The IEEE standard requires that whenever x and y are floating point numbers (so round$(x) = x$ and round$(y) = y$),

$$\boxed{x \odot y = \text{round}(x \cdot y)}$$

where $\odot$ is one of the four arithmetic operations[3]

5.3 Adding and subtracting single-precision numbers (nonzero)

Exact addition of positive – or negative – floating-point numbers can be accomplished if we make the mantissas long enough. Correctly rounded addition can be accomplished with a few extra bits (beyond the usual 24). First suppose we are given two positive floating point numbers x and y.

$$x = 2^e \times 1.a_1 a_2 \dots a_{23} \mid \ 0\,0\,0 \dots$$
$$y = 2^f \times 1.b_1 b_2 \dots b_{23} \mid \ 0\,0\,0 \dots$$

[3] Modular arithmetic and square roots are also discussed.

First, one must line up the binary points. Without loss of generality, $e \geq f$.

$$x = 2^e \times 1.a_1 a_2 \ldots a_{23} \mid \ 0\,0\,0 \ldots$$
$$y = 2^e \times 0.0\,0\,0 \ldots \ 0\,1\,b_1 b_2 \ldots b_{23}\,0\,0\,0 \ldots$$

If $e - f > 24$, then

$$x = 2^e \times 1.a_1 a_2 \ldots a_{23} \mid \ 0\,0\,0 \ldots$$
$$y = 2^e \times 0.0\,0\,0 \ldots \ 0 \mid \ 0 \ldots 1\,b_1 b_2 \ldots b_{23}\,0\,0\,0 \ldots$$

That is, $x \oplus y = x$. If the exponents differ by more than 24, the smaller number is effectively zero.

More generally, to compute $x \oplus y$, where the exponent difference is ≤ 24, one shifts one to the right and makes the exponents equal. Without loss of generality, y has smaller exponent. Since the shift is ≤ 24, $x \oplus y$ can be calculated exactly in 48 bits.

But we can do with less than that.

$$x = 2^e \times 1.a_1 a_2 \ldots a_{23} \mid \ 0\,0\,0 \ldots$$
$$y = 2^e \times 0.0\,0\,0\,b_0 \ldots \ b_{i-1} \mid b_i b_{i+1} \ldots b_{23}\,0\,0\,0 \ldots$$

Possibly $i = 0$ and the first 24 bits in y are zero; $b_0 = 1$.

- If $b_i = 0$ then the sum is rounded down and bits $b_{i+1} \ldots b_{23}$ may be ignored. We call the bit b_i the *guard bit*.
- If the guard bit b_i is 1, and any bit beyond b_i is nonzero, then the sum is rounded up. Otherwise the rounding can be up or down. A *sticky bit* is needed to indicate whether there are any other nonzero bits. The sticky bit is 1 if *any* of $b_{i+1}, \ldots, b_{23}$ is nonzero.
- This is where x and y are both positive, and obviously where they both have the same sign. Where they have opposite sign, or subtracting like signs, it is necessary to retain b_{i+1} as well. This is called the *round bit.*

These notes are based on 'Numerical computing with IEEE floating point arithmetic,' by Michael L. Overton, which is on reserve at the Hamilton Library counter.

6 Meeting the IEEE standard

The three extra bits mentioned, guard, round, sticky, are together called GRS.

- The four arithmetic operations $+, -, \times, /$ need to be implemented in hardware as $\oplus, \ominus, \otimes, \oslash$.
- We suppose given two positive single-precision floating-point numbers

$$x = 2^e \times a_0.a_1 a_2 \ldots a_{23}$$
$$y = 2^f \times b_0.b_1 b_2 \ldots b_{23}$$

- Of course, $a_0 = 1$ and $b_0 = 1$: x and y are *normalised* floating-point numbers.
- Since the sign bits are easily changed, there is no loss of generality in assuming x and y are positive. In that case,
$$N_{\text{min}} \leq x, y \leq N_{\text{max}}.$$
- We only consider addition and subtraction.
- Again because of sign bits, we can assume that $x \geq y$.

6.1 Addition

If x and y have the same exponent, $e = f$, then we can add the significands directly.

$$\begin{aligned} & 1.a_1a_2 \ldots a_{23} \\ + \; & 1.b_1b_2 \ldots b_{23} \end{aligned}$$

There is no problem here: the sum is
$$1d_0.d_1 \ldots d_{23}$$
which needs to be normalised (shifted and rounded), increasing the exponent. Further steps need be taken if $e = f = 127$, making the sum infinite.

Ignore such possibilities.

Conclusion: *if x and y are positive with the same exponent then $x \oplus y$ is easily computed.*

Otherwise, since $x > y$, $e > f$ and the significands should be aligned.

$$\begin{aligned} & 1.a_1a_2 \ldots a_{23} \mid 0\,0\,0 \ldots \\ & 0.0\,0\;\ldots\;1\;b_1b_2 \ldots b_{23} \end{aligned}$$

If $e - f > 24$, then all the nonzero bits in the shifted significand are too far right, the sum is rounded down, and the result is x.

Otherwise, $e - f \leq 24$.

$$\begin{aligned} & 1.a_1a_2 \ldots a_{23} \mid 0\,0\,0 \ldots \\ & \quad 0.0\,0\;\ldots \mid b_i \ldots b_{23}\,0\,0\,0 \ldots \end{aligned}$$

Here
$$i = 24 + f - e$$
Possibly $e - f = 24$, so $i = 0$; $b_0 = 1$.

In any case,

- G: The guard bit is b_i
- R: The round bit is b_{i+1}

- S: The sticky bit is 0 if $b_{i+2}, \ldots, b_{23}$ are all zero, else it is 1.

Suppose the significands are added fully. The result will be

$$d_{-1}d_0.d_1 \ldots d_{23} \mid b_i b_{i+1} \ldots b_{23}$$

where $d_{-1} = 1$ if the sum is at least 2. In the latter case the sum is shifted and the exponents adjusted (maybe $\infty \ldots$)

$$d_{-1}.d_0 d_1 \ldots \mid d_{23} b_i b_{i+1} \ldots b_{23}$$

Clearly, the GSR bits together give enough information to produce $x \oplus y$.

(6.1) Corollary *Given single-precision floating-point numbers x, y of like sign, $x \oplus y$ can be calculated (correctly rounded) using 24 bits plus the GSR bits.* ∎

(Actually, the round bit has no independent significance for adding FPNs of like sign: it could be absorbed into the sticky bit.)

6.2 Subtraction

Given positive FPNs $x > y > 0$ with exponents e and f as before, how is $x \ominus y$ computed? As before, the significand is adjusted, and exponents, by shifting the bits in y to the right.

The following example shows the need for a round bit, i.e., $b_2 4$ in this case.

$$\begin{aligned} &1.0000\ 0000\ 0000\ 0000\ 000 \mid 0000 \times 2^0 \\ -\ &1.0000\ 0000\ 0000\ 0000\ 001 \mid 0000 \times 2^{-2} \end{aligned}$$

Shift:

$$\begin{aligned} &1.0000\ 0000\ 0000\ 000 \mid 0000 \times 2^0 \\ -\ &0.10000\ 0000\ 0000\ 000 \mid 01000 \times 2^0 \end{aligned}$$

Subtract significands:

```
  1.0000 0000 0000 0000 000 000
- 0.0100 0000 0000 0000 000 010
------------------------------
  0.1011 1111 1111 1111 111 11
Shift to normalise
  1.011 1111 1111 1111 1111 1
```

The left shift has moved the guard bit into the mantissa and the round bit adopts the rôle of guard bit.

This happens when a left shift is needed to normalise. A longer left shift could be needed, and it would seem to require further bits to be held for rounding.

Where, after shifting, the difference in significands is small:

$$0.00\ldots00d_0d_1\ldots$$

But the significand in x is at least $1.000\ldots$. Write $c_0.c_1c_2\ldots$ for the shifted significand of y, and make x as small as possible. Also, suppose that y has been shifted right two or more places.

$$\begin{array}{l} 1.0\ 0\ 0\ \ldots \\ 0.0\ c_2\ldots \end{array}$$

The difference is at least $1/2$:

$$0.1d_2d_3\ldots$$

and the left shift, to normalise the difference, is exactly 1 — in other words, if the difference is small then only a small shift was applied to y. We have not exactly proved the following result, but it should be fairly plausible by now.

(6.2) Corollary *Given single-precision floating-point numbers x and y, $x \oplus y$ and $x \ominus y$ can be calculated* correctly rounded *using a guard bit, a round bit, and a sticky bit. All three GRS bits are needed.*

6.3 Multiplication and division

There are convoluted ways of speeding up these operations on a chip, — which led to the Pentium bug — but nothing simple. It appears that one must work with 48 bits.

7 The numbers γ_n

Given $n \in \mathbb{N}$ (nonnegative integers), where $n\epsilon_{\text{mach}} < 1$,

$$\gamma_n = \frac{n\epsilon_{\text{mach}}}{1 - n\epsilon_{\text{mach}}}.$$

(7.1) Lemma *If $0 \le m \le n < 1/\epsilon_{\text{mach}}$, then*

$$\gamma_m \le \gamma_n.$$

Proof. Let $x = m\epsilon_{\text{mach}}$ and $y = n\epsilon_{\text{mach}}$, so $0 \le x \le y < 1$.

$$\gamma_m = \frac{x}{1-x}$$
$$1 + \gamma_m = \frac{1}{1-x}$$
$$1 + \gamma_n = \frac{1}{1-y}.$$

Since $0 < 1 - y < 1 - x \le 1$, $1/(1-x) \le 1/(1-y)$, so $1 + \gamma_m \le 1 + \gamma_n$ and $\gamma_m \le \gamma_n$. ∎

(7.2) Definition Product notation *Given numbers* $n_1, \ldots, n_k$*,*

$$\prod_1^k n_i$$

is an abbreviation for the product

$$n_1 \times n_2 \times \ldots \times n_k$$

(Possibly $k = 0$*, in which case the product takes the default value* 1*).*

The following result is very important.

(7.3) Theorem *Given* n *real numbers* δ_j*, where*

$$0 \leq |\delta_j| \leq \epsilon_{\text{mach}}, \quad (1 \leq j \leq n),$$

and $n\epsilon_{\text{mach}} < 1$*,*

$$\prod_{j=1}^{n} (1 + \delta_j)^{\pm 1} = 1 + \theta,$$

where $|\theta| \leq \gamma_n$*.*

We shall prove it in stages.

(7.4) Lemma *Suppose* $0 \leq \delta \leq \epsilon_{\text{mach}}$*. Then*

$$1 \leq \frac{1}{1-\delta} \leq 1 + \gamma_1.$$

Proof.

$$1 \leq \frac{1}{1-\delta} \leq \frac{1}{1-\epsilon_{\text{mach}}}$$
$$\frac{1}{1-\epsilon_{\text{mach}}} = 1 + \frac{\epsilon_{\text{mach}}}{1-\epsilon_{\text{mach}}} = 1 + \gamma_1 \qquad \blacksquare$$

(7.5) Lemma *Suppose* $0 \leq \delta \leq \epsilon_{\text{mach}}$*. Then*

$$1 - \gamma_1 \leq \frac{1}{1+\delta} \leq 1.$$

Proof.

$$1 + \epsilon_{\text{mach}} \geq 1 + \delta \geq 1 + \delta - 2\epsilon_{\text{mach}}\delta$$
$$1 - \epsilon_{\text{mach}} \geq 1 + \delta - 2\epsilon_{\text{mach}}\delta - 2\epsilon_{\text{mach}}$$
$$= (1 + \delta)(1 - 2\epsilon_{\text{mach}})$$
$$\frac{1}{1+\delta} \geq \frac{1 - 2\epsilon_{\text{mach}}}{1 - \epsilon_{\text{mach}}} = 1 - \frac{\epsilon_{\text{mach}}}{1 - \epsilon_{\text{mach}}} = 1 - \gamma_1. \qquad \blacksquare$$

(7.6) Corollary *If* $|\delta| \le \epsilon_{\text{mach}}$ *then*

$$1 - \gamma_1 \le \frac{1}{1+\delta} \le 1 + \gamma_1. \quad \blacksquare$$

(7.7) Lemma *If* $0 \le r, s \in \mathbb{N}$ *and* $(r+s)\epsilon_{\text{mach}} < 1$*, then*

$$1 \le (1+\gamma_r)(1+\gamma_s) \le 1 + \gamma_{r+s}.$$

Proof.

$$1 + \gamma_r = 1 + \frac{r\epsilon_{\text{mach}}}{1 - r\epsilon_{\text{mach}}} = \frac{1}{1 - r\epsilon_{\text{mach}}}$$

and similarly for the other terms; so

$$(1+\gamma_r)(1+\gamma_s) = \frac{1}{1 - (r+s)\epsilon_{\text{mach}} + rs\epsilon_{\text{mach}}^2} \le \frac{1}{1-(r+s)\epsilon_{\text{mach}}} = 1 + \gamma_{r+s}. \quad \blacksquare$$

(7.8) Corollary *If* $n \in \mathbb{N}$ *and* $0 \le n\epsilon_{\text{mach}} < 1$ *then*

$$(1+\gamma_1)^n \le 1 + \gamma_n.$$

Proof. Induction on n. When $n = 0$ this is equivalent to: $1 = 1$. For induction,

$$(1+\gamma_1)^{n+1} \le (1+\gamma_n)(1+\gamma_1) \le 1 + \gamma_{n+1}. \quad \blacksquare$$

(7.9) Lemma *If* $0 \le r, s \in \mathbb{N}$ *and* $(r+s)\epsilon_{\text{mach}} < 1$*, then*

$$1 - \gamma_{r+s} \le (1-\gamma_r)(1-\gamma_s) \le 1$$

Proof. The inequality is

$$\frac{1 - 2(r+s)\epsilon_{\text{mach}}}{1-(r+s)\epsilon_{\text{mach}}} \le \left(\frac{1-2r\epsilon_{\text{mach}}}{1-r\epsilon_{\text{mach}}}\right)\left(\frac{1-2s\epsilon_{\text{mach}}}{1-s\epsilon_{\text{mach}}}\right)$$

There seems to be no shortcut here — multiply out. The calculations (if correct) make the inequality equivalent to the true inequality

$$3rs\epsilon^2 + 6rs(r+s)\epsilon^3 \ge 0. \quad \blacksquare$$

(7.10) Corollary *If* $n\epsilon_{\text{mach}} < 1$ *then*

$$(1-\gamma_1)^n \ge 1 - \gamma_n.$$

Proof by induction, like Corollary 7.8. $\blacksquare$

Proof of Theorem 7.3. For each of the terms

$$(1+\delta_j)^{\pm 1},$$
$$1 - \gamma_1 \le (1+\delta_j)^{\pm 1} \le 1 + \gamma_1.$$

Hence

$$(1-\gamma_1)^n \le \prod_j (1+\delta_j)^{\pm 1} \le (1+\gamma_1)^n.$$

By Corollaries 7.8 and 7.10,

$$1 - \gamma_n \le \prod_j (1+\delta_j)^{\pm 1} \le 1 + \gamma_n. \quad \blacksquare$$

8 Accuracy of summation

We sum a series

$$S = \sum_1^n x_j$$

in the obvious way

$$x_1 \oplus x_2$$
$$(x_1 \oplus x_2) \oplus x_3 \ldots$$

Let us call the rounded sum $\hat{S}$.

The rounding errors accumulate. We want upper bounds on the total error. According to IEEE p754,[4]

$$x_1 \oplus x_2 = (x_1 + x_2)(1 + \delta_1),$$

where $0 \leq |\delta_1| \leq \epsilon_{\text{mach}}$. Next

$$(x_1 \oplus x_2) \oplus x_3 = ((x_1 \oplus x_2) + x_3)(1 + \delta_2),$$

where $0 \leq |\delta_2| \leq \epsilon_{\text{mach}}$. And so on. Expanding, the computed sum is

$$\hat{S} = (x_1 + x_2)(1 + \delta_2) \cdots (1 + \delta_n) + x_3(1 + \delta_3) \cdots (1 + \delta_n) + \ldots + x_n(1 + \delta_n)$$

$$\hat{S} - S = x_1((1 + \delta_2) \cdots (1 + \delta_n) - 1) + x_2((1 + \delta_3) \cdots (1 + \delta_n) - 1) + \ldots$$

Referring to the γ_n estimates, unless $n \geq 1/\epsilon_{\text{mach}}$ — ridiculously large —

$$|\hat{S} - S| \leq |x_1|\gamma_{n-1} + |x_2|\gamma_{n-2} + \ldots + |x_n|\gamma_1$$

But $\gamma_{n-1} \geq \gamma_{n-2} \ldots$ (Lemma 7.1.

Therefore

$$\boxed{|\hat{S} - S| \leq \gamma_{n-1} \sum_j |x_j|.}$$

It is interesting to find cases where the results are inaccurate despite these guarantees. Here is a simple example: variance (and standard deviation).

There are two ways to compute variance — '1 pass' and '2 pass.' The disadvantage of the 2-pass approach is that the numbers need to be read twice. The definition of *sample mean and variance* of a list $x_1, \ldots, x_n$ of numbers is as follows:

[4]We have forgotten the factor of $1/2$ under round to nearest. Also, it makes little sense to use single-precision arithmetic here, so ϵ_{mach} is usually 2^{-52}. Single precision floating-point numbers are used where memory is scarce, such as on satellites or graphics chips.

$$x = x_1, \ldots, x_n$$
$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$
$$\text{var}(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x - \overline{x})^2.$$

Algebraically,

$$\text{var}(x) = \frac{1}{n-1}\left((\sum_{i=1}^{n} x_i^2) - n(\overline{x})^2\right).$$

The expression on the right gives the '1-pass' method, since you can calculate $\sum x_i^2$ and $\sum x_i$ at the same time. Numerically, it is much less reliable.

One can try this with, say, 100 numbers, all close to 1,000,000. Since the numbers are positive, the sample mean is computed with a relative error of at most

$$\gamma_{100}.$$

which is small both for single- and double-precision numbers.

If to make life easy we ignore the small error in computing the sample mean, the 2-pass sample variance, which is simply a sum followed by a division, is also computed to an accuracy of γ_{100}.

But the 1-pass sample variance computes $\sum x_i^2$ with a small *relative* error, again γ_{100} — **but** — the sum is about 100,000,000,000,000 or 10^{14}. This is really stretching double-precision accuracy (15 decimal digits) and makes nonsense of single-precision accuracy (6).

Here are the results.

```
double precision
n 100 mean 1000000.546875
1 pass variance 0.080956 standard deviation 0.284528
2 pass variance 0.082258 standard deviation 0.286807

single precision
n 100 mean 1000000.187500
1 pass variance 302885.781250 standard deviation 550.350586
2 pass variance 0.212713 standard deviation 0.461208
```

9 Linear equations and matrices

There are procedures for describing the full set of solutions to m linear equations in n unknowns, but we only consider n equations in n unknowns which admit a unique solution. For example

$$x + 2y = 3$$
$$4x + 5y = 6$$

Using matrix notation, this would be written

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}.$$

An $n \times n$ (real) matrix is an $n \times n$ array of real numbers enclosed in brackets for legibility. The array has m rows and n columns. Put another way, m is its *width* and n is its *height*.

Matrices of the same shape can be added or subtracted, but that is not important. **Compatible** matrices can be multiplied.

(9.1) Definition *A $k \times \ell$ matrix A and an $m \times n$ matrix B are* compatible *if $\ell = m$. In other words, the* width *of A and the* height *of B coincide. In this case the* matrix product *is defined. It is a $k \times n$ matrix.*

One can write $A = [a_{rs}]_{k \times m}$ to indicate that a_{rs} is the entry in the r-th row and s-th column of A. Given $B = [b_{st}]_{m \times n}$, the product matrix

$$AB = [c_{rt}]_{k \times n} :$$
$$c_{rt} = \sum_{s=1}^{m} a_{rs} b_{st}.$$

Rather than saying 'A and B are compatible,' we usually say 'AB is defined.'

Clearly AB could be defined and not BA; in fact, they are both defined if and only if they are square matrices of the same size. Even when they are, AB and BA could be different. Matrix multiplication is not commutative.

Matrix multiplication is associative, though: $A(BC) = (AB)C$ when either product is defined — in this case both products are defined.

(9.2) Definition *$\mathbb{R}^{m \times n}$ is the set of all $m \times n$ real matrices.*

When $n = 1$ we have column vectors of height n, *and when $m = 1$ we have* row vectors of width m.

So let us relate this to the matrix equation.

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}.$$

Multiply the left-hand side according to matrix multiplication rules

$$\begin{bmatrix} x + 2y \\ 4x + 5y \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \end{bmatrix}.$$

which is just right.

Actually, matrices are often used to define *linear maps*. For example the matrix

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

can be interpreted as '45° rotation about the origin in $\mathbb{R}^2$ — the Euclidean plane.' The matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

encodes the *identity map*, and is given the name '3×3 identity matrix';

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$$

is reflection in the xy-plane;

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

is vertical projection onto the xy-plane within $\mathbb{R}^2$; and

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

is a projection of $\mathbb{R}^3$ onto $\mathbb{R}^2$. This is our only example involving non-square matrices.

9.1 EROs and Gauss-Jordan elimination

A system of linear equations

$$A \begin{bmatrix} x_1 \\ x_2 \\ \bullet \\ \bullet \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \bullet \\ \bullet \\ b_n \end{bmatrix}$$

(where A is an $n \times n$ matrix) can be solved by forming the *augmented matrix* of the system

$$\begin{array}{cccc} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \dots & & & & \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{array}$$

(This is an $n \times (n+1)$ matrix, but as it is 'stand-alone,' there is no need to surround it with brackets,) and to use *elementary row operations* (EROs) to bring the augmented matix to what is called *reduced row-echelon form*. The operations are

- **Scale** a row by a nonzero constant
- **Swap** two different rows
- **Subtract** from one row a multiple of another row (not the same row).

The reduced row-echelon form we want is where the first n columns form the identity matrix. This is not always possible, but it is when the set of equations has exactly one solution.

Let us take our example of 2 equations in 2 unknowns.

```
Clear the first column, then the second
   1   2   3  call it row 1      1   2   3     subtract 2 x row 2
   4   5   6    - 4 x row 1      0  -3  -6     / (-3) call result row 2
Both columns cleared
   1   0  -1  RREF
   0   1   2
```

EROs preserve equations, and the RREF says

$$x = -1$$
$$y = 2$$

The procedure could grind to a halt, when there are no solutions or more than one.

```
  1  2  3  1     R1    1  2   3   1  -2 R2         1  0 -1 -1
  4  5  6  1  -4 R1    0 -3  -6  -3  /(-3)=R2      0  1  2  1
  7  8  9  1  -7 R1    0 -6 -12  -6  +6 R2         0  0  0  0
```

At this stage, we are stuck, because we cannot use a_{33} to clear the third column. As it happens, this system has infinitely many solutions, but there could just as easily be no solution.

10 Gaussian elimination

Here is a 3×4 augmented matrix.

```
1 -1  4  9
3 -1 10 25
1  0  2  6
```

Gauss-Jordan elimination:

```
1 -1  4  9    =R1
3 -1 10 25  -3 R1  (mistake in earlier draft)
1  0  2  6   - R1

1 -1  4  9   + R2
0  2 -2 -2  /2=R2
0  1 -2 -3   - R2

1 0  3   8   - 3 R3
0 1 -1  -1   +   R3
0 0 -1  -2   *-1=R3
```

```
1 0 0 2   rref: x = 2, y = 1, z = 2.
0 1 0 1
0 0 1 2
```

Gaussian elimination aims to reduce the first n rows and columns to an *upper triangular matrix.* It does *not* scale the diagonal elements. It does *not* clear the column above the diagonal elements. It *does* use 'pivoting,' unlike the example below.

```
1 -1  4  9    =R1
3 -1 10 25  -3 R1
1  0  2  6   - R1

1 -1  4  9
0  2 -2 -2       =R2
0  1 -2 -3  - 1/2 R2

1 -1  4  9          (correcting a mistake in an earlier draft).
0  2 -2 -2
0  0 -1 -2  finished
```

To solve for x, y, z, use back substitution.

$$-z = -2; \quad z = 2$$
$$2y - 2z = -2; \quad 2y - 4 = -2; 2y = 2; \quad y = 1$$
$$x - y + 4z = 9; \quad x - 1 + 8 = 9; x = 2$$

This calculation did not use pivoting, so it was not exactly Gaussian elimination. 'Pivoting' means always making the diagonal element as large as possible by swapping the 'current' row with a lower row, if necessary. (This is 'partial pivoting.' There's something else called 'full pivoting,' but it is not used much.)

```
1 -1  4  9 swap
3 -1 10 25 swap
1  0  2  6

3  -1  10  25       =R1
1  -1   4   9   -1/3 R1
1   0   2   6   -1/3 R1

3   -1    10    25
0 -2/3   2/3   2/3  = R2
0  1/3  -4/3  -7/3  + 1/2 R2

3    -1    10   25
0  -2/3   2/3  2/3
0     0    -1   -2 finished
```

Back substitution

$$\begin{aligned} -z &= -2; \quad z = 2 \\ -2y/3 + 4/3 &= 2/3; \quad y = 1 \\ 3x - 1 + 20 &= 25; \quad x = 2 \end{aligned}$$

Pivoting is a *strategy* to reduce rounding error. The strategy is to try and make the diagonal elements large. The reason is simply that when solving, one divides by the diagonal elements; if you divide by a small number, you can create a relatively large error.

'Large' means *large in absolute value:* for example, -10^{10} is large and 10^{-10} is small.

For example, suppose ε is a small positive number. Given the following augmented matrix

$$\begin{array}{ccc} \varepsilon & 1 & 1 \\ -1 & 1 & 0 \end{array}$$

Without pivoting, and with exact arithmetic, we get

$$\begin{array}{ccc} \varepsilon & 1 & 1 \\ 0 & 1 + \frac{1}{\varepsilon} & \frac{1}{\varepsilon} \end{array}$$

Back substituting

$$\begin{aligned} \frac{1+\varepsilon}{\varepsilon} y = \frac{1}{\varepsilon} \quad y &= \frac{1}{1+\varepsilon} \\ \varepsilon x = 1 - y = \frac{1+\varepsilon-1}{1+\varepsilon} \quad x &= \frac{1}{1+\varepsilon} \end{aligned}$$

All very well, but if ε is small then rounding error causes problems. For example, suppose that the calculations are single-precision floating-point and $\varepsilon = 2^{-25}$. Then the rounded calculations become

$$\begin{array}{ccc} \varepsilon & 1 & 1 \\ 0 & \frac{1}{\varepsilon} & \frac{1}{\varepsilon} \end{array}$$

The difference is that $1 \oplus 1/\varepsilon = 1/\varepsilon$.

Then

$$\begin{aligned} y &= 1 \\ \varepsilon x + 1 &= 1; \quad x = 0 \end{aligned}$$

Although y is correctly rounded, x couldn't be more wrong. If pivoting is used:

$$\begin{array}{ccc} -1 & 1 & 0 \\ \varepsilon & 1 & 1 \end{array}$$

$$\begin{array}{ccc} -1 & 1 & 0 \\ 0 & 1+\varepsilon & 1 \end{array}$$

and $1 \oplus \varepsilon = 1$:

$$\begin{array}{ccc} -1 & 1 & 0 \\ 0 & 1 & 1 \end{array}$$

whence $x = y = 1$. This time x and y are correctly rounded.

11 LU factorisation

Gaussian elimination aims to reduce the first n rows and columns to an *upper triangular matrix.* It does *not* scale the diagonal elements. It does *not* clear the column above the diagonal elements. It *does* use 'pivoting.'

Repeat the non-pivoting Gaussian elimination example; this time, apply the same EROs to the identity matrix

```
1 -1  4  9    =R1      1  0  0
3 -1 10 25  -3 R1      0  1  0
1  0  2  6   - R1      0  0  1

1 -1  4  9              1  0  0
0  2 -2 -2       =R2   -3  1  0
0  1 -2 -3  - 1/2 R2   -1  0  1

1 -4  4  9               1    0   0
0  2 -2 -2              -3    1   0
0  0 -1 -2  finished   1/2 -1/2   1
```

Convention. The equations are understood to take the form $Ax = b$ when matrices are used. The augmented matrix consist of A with b appended as the last column. Also, U will mean the corresponding upper-triangular matrix.

Using the usual arguments about EROs and elementary matrices, we know that

$$U = MA$$

where M is the lower-triangular matrix on the right. We write $A = LU$, where $L = M^{-1}$. This is an *LU-factorisation of A*. U is an upper-triangular matrix; M is a lower-triangular matrix with 1s on the main diagonal. So is L (the properties hold for the inverse). We can calculate L quite easily, by solving for a, b, c below.

$$LM = I:$$

$$\begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ -3 & 1 & 0 \\ 1/2 & -1/2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$a - 3 = 0; \quad a = 3$$

$$c - 1/2 = 0; \quad c = 1/2$$

$$b - 3/2 + 1/2 = 0; \quad b = 1$$

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}$$

What happens if there is pivoting?

```
1 -1  4  9 swap  1  0  0
3 -1 10 25 swap  0  1  0
1  0  2  6       0  0  1

3  -1  10  25       =R1   0   1   0
1  -1   4   9   -1/3 R1   1   0   0
1   0   2   6   -1/3 R1   0   0   1

3   -1    10    25              0    1  0
0 -2/3   2/3   2/3  = R2        0 -1/3  0
0  1/3  -4/3  -7/3  + 1/2 R2  1/2 -1/3  1

3    -1    10   25              0     1   0
0  -2/3   2/3  2/3              0  -1/3   0
0     0    -1   -2 finished    1/2 -1/2   1
```

Because rows were swapped, the matrix M is not upper triangular. We'll come back to that later.

Now it is possible to calculate the LU factorisation directly, and where there is no pivoting it is quite easy. Just follow this order:

- First row of U
- First column of L
- Second row of U
- Second column of L
- etcetera

For example,

$$\begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix} \begin{bmatrix} d & e & f \\ 0 & g & h \\ 0 & 0 & i \end{bmatrix}$$

$$d = 1; \quad e = -1; \quad f = 4$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ b & c & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 0 & g & h \\ 0 & 0 & i \end{bmatrix}$$

$$a = 3; \quad b = 1.$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & c & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 0 & g & h \\ 0 & 0 & i \end{bmatrix}$$
$$-3 + g = -1; \quad g = 2$$
$$12 + h = 10; \quad h = -2$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & c & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 2 & -2 \\ 0 & 0 & i \end{bmatrix}$$
$$-1 + 2c = 0; \quad c = 1/2$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 2 & -2 \\ 0 & 0 & i \end{bmatrix}$$
$$4 - 1 + i = 2; \quad i = -1$$

$$\begin{bmatrix} 1 & -1 & 4 \\ 3 & -1 & 10 \\ 1 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 1 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 4 \\ 0 & 2 & -2 \\ 0 & 0 & -1 \end{bmatrix}$$

11.1 Analysis of LU factorisation

(11.1) Definition *If $A = [a_{ij}]_{m\times n}$ is a (real) matrix, then its* absolute value matrix *is the matrix*

$$[|a_{ij}|]_{m\times n}$$

One writes the absolute value matrix of A as

$$|A|$$

(This notation could be confusing; sometimes $|A|$ is used for the determinant of A.)

(11.2) Proposition *Suppose A is an $n \times n$ (invertible) matrix. Let $A = LU$ be the LU factorisation of A (without pivoting). In the presence of rounding errors, approximations $\widehat{L}$ and $\widehat{U}$ are computed. Then the rounding errors satisfy*

$$|LU - \widehat{L}\widehat{U}| \leq \gamma_n |\widehat{L}||\widehat{U}|.$$

Proof for the second-easiest case, $n = 2$**,**

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ e & 1 \end{bmatrix} \begin{bmatrix} f & g \\ 0 & h \end{bmatrix}$$

We compute the first row of U, the first column of L, the second row of U, the second column of L (nothing to do there)

The first row of U equals that of A:

$$f = a; \quad g = b$$

For the first column of L,

$$c = ef; \quad e = c/f = c/a$$

And the second row of U:

$$eg + h = d; \quad h = d - eg = d - (c/a)b$$

Write $\hat{e}$ for the rounded form of e. Remember that $x \oplus y$ is the correctly rounded floating-point sum of floating-point numbers x and y. Similarly $\ominus, \otimes, \oslash$.

$$\hat{f} = a; \quad \hat{g} = b$$
$$\hat{e} = c \oslash \hat{f}$$
$$\hat{h} = d \ominus \hat{e} \otimes \hat{g}$$

We need to prove the following inequality (all the numbers are floating-point).

$$|A - \hat{L}\hat{U}| \leq \gamma_2 |\hat{L}||\hat{U}|.$$

Actually, it seems that for the 2×2 case, we can replace γ_2 by γ_1.[5]

In other words,[6]

$$\begin{bmatrix} 0 & 0 \\ |c - \hat{e}\hat{f}| & |d - \hat{e}\hat{g} - \hat{h}| \end{bmatrix} \leq \quad \gamma_1 \begin{bmatrix} |a| & |b| \\ |\hat{e}||\hat{f}| & |\hat{e}||\hat{g}| + |\hat{h}| \end{bmatrix}$$

in other words

$$|c - \hat{e}\hat{f}| \leq \gamma_1 |\hat{e}||\hat{f}|, \quad \text{and} \quad |d - \hat{e}\hat{g} - \hat{h}| \leq \gamma_1 (|\hat{e}||\hat{g}| + |\hat{h}|).$$

Recall the important theorem: if $|\delta_i| \leq \epsilon_{\text{mach}}$ for $1 \leq i \leq n$, and $n\epsilon_{\text{mach}} < 1$, then

$$\prod_{i=1}^{n} (1 + \delta_i)^{\pm 1} = 1 + \theta,$$
$$\text{where } |\theta| \leq \gamma_n.$$

[5]There could be a mistake in the calculation. If so, it will be corrected in due course.

[6]We're sticking with γ_1.

The first inequality is easy. $\hat{f} = f = a$ (exactly) and $\hat{e} = c \oslash f$:

$$\hat{e} = \frac{c}{f}(1 + \delta_1)$$

where $|\delta_1| \leq \epsilon_{\text{mach}}$. Then (again note $\hat{f} = f$)

$$\begin{aligned} \hat{e}\hat{f} &= c(1 + \delta_1) \\ c &= \frac{\hat{e}\hat{f}}{1 + \delta_1} \\ c - \hat{e}\hat{f} &= \hat{e}\hat{f}\left(\frac{1}{1 + \delta_1} - 1\right) \\ |c - \hat{e}\hat{f}| &\leq \gamma_1 |\hat{e}||\hat{f}|, \end{aligned}$$

as required.

The other inequality is harder.

$$\begin{aligned} h &= d - eg \\ \hat{h} = d \ominus \hat{e} \otimes \hat{g} \quad &= (d - \hat{e}\hat{g}(1 + \delta_2))(1 + \delta_3) \\ \frac{\hat{h}}{1 + \delta_3} &= d - \hat{e}\hat{g}(1 + \delta_2) \\ \frac{\hat{h}}{1 + \delta_3} + \hat{e}\hat{g}(1 + \delta_2) &= d \\ d - \hat{h} - \hat{e}\hat{g} &= \hat{h}\left(\frac{1}{1 + \delta_3} - 1\right) + \hat{e}\hat{g}(1 + \delta_2 - 1) \\ |d - \hat{h} - \hat{e}\hat{g}| &\leq \gamma_1(|\hat{e}||\hat{g}| + |\hat{h}|), \end{aligned}$$

as required. ∎

11.2 Inaccuracy

Applied without pivoting to the array

$$\begin{bmatrix} \varepsilon & 1 \\ 1 & 1 \end{bmatrix}$$

we get

$$\begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \begin{bmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{bmatrix}$$

Using single-precision, with $\varepsilon = 2^{-25}$, $1 - 1/\varepsilon = -1/\varepsilon$:

$$\hat{L}\hat{U} = \begin{bmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{bmatrix} \begin{bmatrix} \varepsilon & 1 \\ 0 & -\frac{1}{\varepsilon} \end{bmatrix} = \begin{bmatrix} \varepsilon & 1 \\ 1 & 0 \end{bmatrix}$$

So $LU - \hat{L}\hat{U} =$

$$\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix},$$

A non-negligible error. This can be reconciled with the upper bound, because $|\hat{L}||\hat{U}|$ contains a large entry:

$$|\hat{L}||\hat{U}| = \begin{bmatrix} \varepsilon & 1 \\ 1 & \frac{2}{\varepsilon} \end{bmatrix}.$$

Pivoting would have reduced the error enormously.

12 LU, permutation matrices, and pivoting

A *permutation* of $1, 2, \ldots, n$ is a bijective map from this set to itself. For example,

$$1 \mapsto 1, \quad 2 \mapsto 4, \quad 3 \mapsto 2, \quad 4 \mapsto 3$$

or, in more compact form,

$$1 \mapsto 1, \quad 2 \mapsto 4 \mapsto 3 \mapsto 2$$

is an example.

In Gaussian elimination with pivoting, EROs 'swap' and 'subtract' can be used (no scaling):

- Form the $n \times (n+1)$ augmented matrix, call it M_0.
- Swap rows 1 and $1'$ where $1' \geq 1$ (possibly $1' = 1$).
- Apply 'subtract' operations to reduce the first column.
- Swap rows 2 and $2'$.
- Apply 'subtract' operations to reduce the second column.
- Etcetera... until the $n-1$-st column is reduced (then we have upper triangular form) U.

The same effect could be obtained differently, if with hindsight you knew the sequence $1', 2', 3' \ldots (n-1)'$.

- Swap rows 1 and $1'$ in M_0, producing a matrix M_1. Then swap rows 2 and $2'$ in M_1, producing a matrix M_2. And so on, until a matrix M_{n-1} is produced. M_{n-1} is a *row-permuted* version of the augmented matrix M_0.
- Apply subtract operations, possibly different from the previous ones because of the rows being swapped, to produce the same upper triangular form U.

(12.1) Corollary *Gaussian elimination with partial*[7] *pivoting on an augmented matrix M is equivalent to Gaussian elimination* without pivoting *on some row-permuted copy of M.*

LU-factorisation of a square matrix A with pivoting (whatever that means) is equivalent to LU-factorisation without pivoting of a row-permuted version of A. ∎

[7] The qualifier 'partial' had been mistakenly omitted previously. With partial pivoting, the rows can be permuted. One can also have 'full pivoting' where the columns can be permuted as well. In practice it doesn't seem to be used much.

Repeat LU factorisation without pivoting on the matrix

$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{matrix}$$

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{bmatrix} \begin{bmatrix} ? & ? & ? \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

The first row of U equals that of A.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

The first column of L can be completed.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & ? & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

From the second row of L the second row of U can be computed.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & ? & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & ? \end{bmatrix}$$

From the second column of U the third row, second column of L can be computed.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & ? \end{bmatrix}$$

From the third row of L the third row of U can be computed.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4 & 1 & 0 \\ 7 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{bmatrix}.$$

Let us apply Gaussian elimination *with pivoting* to an augmented matrix with A in the first 3 columns.

```
1    2    3    3   swap     7   8   10   16          = R1
4    5    6    9            4   5    6    9     - 4/7 R1
7    8   10   16   swap     1   2    3    3     - 1/7 R1

7    8   10    16           7     8    10    16
0  3/7  2/7  -1/7  swap      0   6/7  11/7   5/7     = R2
0  6/7 11/7   5/7  swap      0   3/7   2/7  -1/7 - 1/2 R2
```

```
  7     8     10      16    z = 1; y = -1; x = 2.
  0   6/7   11/7     5/7
  0     0  -7/14   -7/14
```

Apply the same eros to the $3x3$ identity matrix. Without pivoting, this would be a lower-triangular matrix, L^{-1}. What is it?

```
  1    0    0     swap     0    0    1        = R1
  0    1    0              0    1    0    - 4/7 R1
  0    0    1     swap     1    0    0    - 1/7 R1

  0    0      1            0    0      1
  0    1   -4/7   swap     1    0   -1/7     = R2
  1    0   -1/7   swap     0    1   -4/7  -1/2 R2

   0   0      1
   1   0   -1/7
-1/2   1  -7/14
```

This is easy enough to invert...

$$\begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & -1/7 \\ -1/2 & 1 & -1/2 \end{bmatrix}^{-1} = \begin{bmatrix} 1/7 & 1 & 0 \\ 4/7 & 1/2 & 1 \\ 1 & 0 & 0 \end{bmatrix}.$$

Pivoting can also be applied to LU factorisation. What it means is that a *row-permuted* version of A can be factorised as LU.

If we apply just the *swap* operations above, to the 3×3 identity matrix, in the given order, we obtain a *permutation matrix* P.

```
  1    0    0     swap     0    0    1
  0    1    0              0    1    0 swap
  0    0    1     swap     1    0    0 swap

  0    0    1
P: 1    0    0
  0    1    0
```

If C is any matrix of height 3, then PC is the corresponding row-permuted version of C. Also, P is an orthogonal matrix: $P^{-1} = P^T$.

Partial pivoting amounts to swapping rows of A, and *parts of* rows of L. Take the same example again.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} = LU = \begin{bmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{bmatrix} \begin{bmatrix} ? & ? & ? \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

Swap rows 1 and 3 of L, to bring up the pivot element. Then the first row of A matches that of U.

$$\begin{bmatrix} 7 & 8 & 10 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ ? & 1 & 0 \\ ? & ? & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

The first column of L can be completed.

$$\begin{bmatrix} 7 & 8 & 10 \\ 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 4/7 & 1 & 0 \\ 1/7 & ? & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & ? & ? \\ 0 & 0 & ? \end{bmatrix}$$

Now we consider pivoting on the second row. That is, perhaps the second and third rows should be swapped to increase the absolute value of u_{22}.

If we swap, we must swap within the first column of L. Without swapping,

$$(4/7)8 + u_{22} = 5, \quad u_{22} = 3/7$$

and with swapping

$$(1/7)8 + u_{22} = 2, \quad u_{22} = 6/7$$

which is larger (in absolute value): we swap.

$$\begin{bmatrix} 7 & 8 & 10 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/7 & 1 & 0 \\ 4/7 & ? & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & 6/7 & ? \\ 0 & 0 & ? \end{bmatrix}$$

Then $10/7 + u_{23} = 3$ or $u_{23} = 11/7$.

$$\begin{bmatrix} 7 & 8 & 10 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/7 & 1 & 0 \\ 4/7 & ? & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & 6/7 & 11/7 \\ 0 & 0 & ? \end{bmatrix}$$

There is no question of pivoting again. $(4/7)(8) + (6/7)\ell_{32} = 5$, so $\ell_{32} = 1/2$.

$$\begin{bmatrix} 7 & 8 & 10 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/7 & 1 & 0 \\ 4/7 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & 6/7 & 11/7 \\ 0 & 0 & ? \end{bmatrix}$$

Then $(4/7)10 + (1/2)(11/7) + u_{33} = 6$, $u_{33} = -1/2$.

$$\begin{bmatrix} 7 & 8 & 10 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1/7 & 1 & 0 \\ 4/7 & 1/2 & 1 \end{bmatrix} \begin{bmatrix} 7 & 8 & 10 \\ 0 & 6/7 & 11/7 \\ 0 & 0 & -1/2 \end{bmatrix}$$

Summarising: LU factorisation with pivoting of a matrix A results in

$$LU = PA$$

where P is a permutation matrix. P can be computed by applying the swap operations in the correct order on the identity matrix.

13 Accuracy of linear equation solutions

Errors in calculation are of two types (three if one includes data measurement errors).

- Truncation errors, such as is inevitable when the Newton-Raphson method has to be stopped after finitely many steps. In Simpson's Rule, truncation error depends on the step-size. Similarly, truncation errors are inevitable in the $QR - RQ$ method.
- Rounding errors, inevitable with floating-point calculation, whether single- or double-precision.

Gaussian elimination is more-or-less equivalent to

- To solve $Ax = b$
- Form the LU factorisation of A (or, with partial pivoting, on a row-permuted version of A).
- Solve $Ly = b$.
- Solve $Ux = y$.

(13.1) Proposition *Solving $Ux = y$ produces a rounded answer $\tilde{x}$ such that*

$$(U + \Delta U)\tilde{x} = y$$

where $|\Delta U| \leq \gamma_n |U|$. Similarly: solving $Ly = b$. (No proof.) ∎

(13.2) Corollary *Solving $Ax = b$ produces a rounded solution $\tilde{x}$ such that*

$$(A + \Delta A)\tilde{x} = b$$

where $|\Delta A| \leq \gamma_{3n} |A|$.

Proof. Write Δ_1 for the matrix $\tilde{L}\tilde{U} - A$. We know

$$|\Delta_1| \leq \gamma_n |\tilde{L}||\tilde{U}|.$$

Now, solving $Ly = b$ — we don't solve $Ly = b$, we solve $\tilde{L}y = b$, an approximation to the correct equation, and the solution $\tilde{y}$ is an approximation to the solution to $\tilde{L}y = b$. Referring to the above proposition,

$$(\tilde{L} + \Delta\tilde{L})\tilde{y} = b$$

where $|\Delta\tilde{L}| \leq \gamma_n |\tilde{L}|$. Next, solving

$$\tilde{U}x = \tilde{y}$$

produces $\tilde{x}$, where

$$(\tilde{U} + \Delta\tilde{U})\tilde{x} = \tilde{y}$$

Composing these results

$$(\tilde{L} + \Delta\tilde{L})(\tilde{U} + \Delta\tilde{U})\tilde{x} = b$$

Multiply out and replace $\tilde{L}\tilde{U}$ by $A + \Delta_1$:

$$(A + \Delta_1 + \Delta\tilde{L}\tilde{U} + \tilde{L}\Delta\tilde{U} + \Delta\tilde{L}\Delta\tilde{U})\tilde{x} = b$$
$$(A + \Delta A)\tilde{x} = b, \quad \text{where}$$
$$\Delta A = \Delta_1 + \Delta\tilde{L}\tilde{U} + \tilde{L}\Delta\tilde{U} + \Delta\tilde{L}\Delta\tilde{U}$$

Applying the estimates for Δ_1, $\Delta\tilde{L}$, and $\Delta\tilde{U}$, we get

$$|\Delta A| \le (3\gamma_n + \gamma_n^2)|\tilde{L}||\tilde{U}|.$$

The coefficient is

$$\frac{3\epsilon_{\text{mach}}}{1 - \epsilon_{\text{mach}}} + \frac{n^2\epsilon_{\text{mach}}^2}{(1 - n\epsilon_{\text{mach}})^2} =$$
$$\frac{3n(1 - n\epsilon_{\text{mach}})\epsilon_{\text{mach}} + n^2\epsilon_{\text{mach}}^2}{(1 - n\epsilon_{\text{mach}})^2} =$$
$$\frac{3n\epsilon_{\text{mach}} - 2n^2\epsilon_{\text{mach}}^2}{(1 - n\epsilon_{\text{mach}})^2} \le$$
$$\frac{3n\epsilon_{\text{mach}}}{(1 - n\epsilon_{\text{mach}})^2} \le \frac{3n\epsilon_{\text{mach}}}{1 - 2n\epsilon_{\text{mach}}} \le \gamma_{3n}. \quad \blacksquare$$

Backward error analysis. Experts in the field say that it is often easier to relate the approximate solution to a problem to the exact solution to a nearby problem.

For example, suppose $Ux = b$ were solved with approximate solution $\tilde{x}$. We may write Δx for $x - \tilde{x}$. The *forward error* would be Δx. On the other hand, if we produce a related matrix ΔU such that $(U + \Delta U)\tilde{x} = b$ *exactly*, analysis of ΔU would be called backward error analysis.

(13.3) Definition *If M is an invertible matrix, then its* condition matrix *is*

$$|M^{-1}||M|$$

(remember $|M|$ is a matrix with nonnegative entries).

(13.4) Lemma *For any compatible matrices M, N,*

$$|MN| \le |M||N|$$

— for example,

$$\begin{bmatrix} 1 & -2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & -6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} -9 & -22 \\ 43 & 14 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

and

$$\begin{bmatrix} 9 & 22 \\ 43 & 14 \end{bmatrix} \le \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}.$$

$$U\Delta x = (\Delta U)\tilde{x}$$
$$\Delta x = U^{-1}\Delta U\tilde{x}$$
$$|\Delta x| \leq \gamma_n |U^{-1}||U||\tilde{x}|.$$

If the condition matrix has large entries, then computations can be inaccurate. Here is a triangular matrix with large condition matrix:

$$\begin{bmatrix} \epsilon & 1 \\ 0 & \epsilon \end{bmatrix}$$

Its inverse is

$$\begin{bmatrix} (1/\epsilon) & -1/\epsilon^2 \\ 0 & 1/\epsilon \end{bmatrix}$$

Its condition matrix is

$$\begin{bmatrix} 1 & 2/\epsilon \\ 0 & 1 \end{bmatrix}$$

14 Numerical integration

This means approximate integration, based on the fact that

$$\int_a^b f(x)dx$$

is the signed area between the graph of the curve and the x-axis, or more precisely

$$\lim_{N\to\infty} \sum_{i=0}^{N-1} \frac{f(a + i(b-a)/N)}{N}.$$

At least, this is so if f is continuous. (Integrating discontinuous functions has fascinated analysts for a century.)

The **Trapezoidal method** approximates f by a *piecewise-linear* function, passing through the points $(x_i, f(x_i))$. The area between x_i and x_{i+1} is the area of a trapezoid, hence the name. It is

$$((y_i + y_{i+1})/2) \times \Delta x_i.$$

Adding we get

$$\frac{(b-a)}{2N}(y_0 + 2y_1 + \ldots + 2y_{N-1} + y_N).$$

For example, with $f(x) = 4/(1+x^2)$, $a = 0$, $b = 1$, and $N = 4$, the Trapezoidal method yields

$$\frac{1}{8}(4 + 8\frac{16}{17} + 8\frac{4}{5} + 8\frac{16}{25} + 2) = 3.1311,$$

a rather bad approximation to π.

The Trapezoidal Method is based on approximating a function by a piecewise-linear function. Accuracy can be improved, *using the same data-points*, by approximating a function by a piecewise-quadratic function, a piecewise-cubic function, and so on.

Given $n+1$ points (x_i, y_i), where the x_i are all distinct, there is a unique polynomial of degree $\leq n$ whose graph passes through these points. (If curious to know more, look up Lagrangian Interpolation.)

We want to integrate such a polynomial with $n = 2$ and $x_i = a + ih$ where h, the steplength, $= \Delta x_i = (b-a)/2$. In other words, y_0, y_1, y_2 are given and x_0, x_1, x_2 are $a, a+h, a+2h$ respectively. ($x_2 = b$).

Suppose $c = x_1 = (a+b)/2$ and write the interpolating polynomial as a polynomial in $x - c$:

$$A(x-c)^2 + B(x-c) + C.$$

Then

$$\int_a^b (A(x-c)^2 + B(x-c) + C)dx = \int_{-h}^{h} (Ax^2 + Bx + C)dx = \frac{2Ah^3}{3} + 2Ch.$$

Also,

$$A(-h)^2 + B(-h) + C = y_0, \quad A(0)^2 + B(0) + C = y_1, \quad \text{and} \quad A(h)^2 + B(h) + C = y_2.$$

$C = y_1$, B is irrelevant, and $2Ah^2 = y_0 - 2C + y_2$, so $A = (y_0 - 2y_1 + y_2)/2h^2$. The approximate integral is

$$\frac{h(y_0 - 2y_1 + y_2)}{3} + 2y_1 h = \frac{h}{3}\left(y_0 + 4y_1 + y_2\right).$$

If we are given an *even* number N of intervals, and we apply this formula to consecutive pairs of intervals, the sum is

$$\frac{b-a}{3N}\left((y_0 + y_N) + 4(y_1 + y_3 + \ldots) + 2(y_2 + y_4 \ldots)\right)$$

which is **Simpson's Rule.**

Applied to the previous data, the Simpson's Rule approximation for π is

$$\frac{1}{12}(4 + 16\frac{16}{17} + 8\frac{4}{5} + 16\frac{16}{25} + 2) = 37.698823/12 = 3.1415685,$$

which is far more accurate than the Trapezoidal Rule provides.

Truncation error. The trapezoidal method uses a linear interpolant

$$a_0 + a_1(x-a).$$

It resembles a degree-1 Taylor polynomial, but $a_1 \neq f'(a)$. The following result is similar to Taylor's Theorem (up to linear terms).

Even for the Trapezoidal method, it is hard to estimate the error. Here is an **'order of magnitude'** estimate.

(14.1) Notation *$g(x)$ is $O((x-a)^k)$ means (roughly)*

$$\lim_{x\to a} \frac{g(x)}{(x-a)^k} < \infty$$

(14.2) Lemma *If $g(x)$ is $O((x-a)^k)$, then $\int_a^b g(x)dx$ is $O((b-a)^{k+1}$.*

(14.3) Lemma *(i) The truncation error in a single step of the Trapezoidal method is $O((b-a)^3)$.*
(ii) The truncation error with steplength $h = (b-a)/N$ is $O(h^2)$.

Proof uses Taylor's Theorem. (i) The trapezoidal method computes the integral of the polynomial

$$a_0 + a_1(x-a) = f(a) + (x-a)\frac{f(b)-f(a)}{b-a}$$

and, from the mean value theorem, $a_1 = f'(c)$ for some c between a and b.

From Taylor's Theorem

$$f(x) = a_0 + (x-a)f'(a) + O((x-a)^2).$$

$$\int_a^b (f(x) - a_0 - a_1(x-a)) = (f'(a) - f'(c))\frac{(b-a)^2}{2} + O((b-a)^3).$$

On the other hand, $f'(c) - f'(a) = (c-a)f''(d)$ for some d between a and c, so it is $O(b-a)$, and $(f'(a) - f'(c))(b-a)^2/2$ is also $O((b-a)^3)$. This proves (i).

(ii): Total error over N steps is $O(Nh^3)$ from (i), which is $O(h^2)$. **Q.E.D.**

Here is a more useful estimate, from the textbooks:

(14.4) Corollary *Given an upper bound M on $|f''(\xi)|$ over $[a,b]$, the absolute-value error in the Trapezoidal method is bounded by*

$$\frac{M(b-a)}{12}h^2.$$

For example, a laborious calculation (unreliable) with $f(x) = 4/(1+x^2)$, $a = 0$, $b = 1$, shows $M = 8$. The error bound for the Trapezoidal-method estimate of π ($N = 4$) is

$$\frac{8}{12},$$

which is consistent with the data.

Simpson's Rule can be treated in the same way. The details are even more laborious than with the Trapezoidal method.

These results can be found in *Analysis of Numerical Methods,* by Isaacson and Keller. It's not an easy book to read, being full of cross-references. It would take about 3 pages of these notes to repeat the analysis, and we don't. But here is the error estimate:

(14.5) Lemma *The error in Simpson's Rule is bounded by*

$$\frac{M(b-a)}{90}h^4,$$

where M is the maximum value of $|f^{(4)}(x)|$ over $[a,b]$. ∎

In particular, if f is a cubic polynomial, then Simpson's rule is exact because $f^{(4)}(x) \equiv 0$. This is easily checked.

It would be instructive to apply this to the calculation of π, but we would need to calculate the fifth derivative of $1/(1+x^2)$, which would take some time. Maple, or no doubt Mathematica, would easily furnish the answer.

Using Maple, it appears that the fourth derivative absolute value maximum in is 24, giving an error estimate of

$$\frac{1}{15 \times 2^{15}} = .0005249$$

15 ODEs

We consider only the following *initial value problem*

$$\frac{dy}{dx} = f(x,y); \quad y(a) = c.$$

We want an approximate solution over the interval $[a,b]$.

For example

$$\frac{dy}{dx} = y; \quad y(0) = 1$$

has the solution $y = e^x$.

The crudest and obvious numerical solution is to choose a stepsize $h = (b-a)/N$, again

$$x_i = a + ih, \quad 0 \le i \le N,$$

and calculate approximations

$$y_i \quad \text{to} \quad y(x_i),$$

beginning of course with $y_0 = c$,

— using $\Delta y_i = (y_{i+1} - y_i)/h$ as a substitute for $y'(x_i)$. Then for $1 \le m \le N$,

$$\frac{y_m - y_{m-1}}{h} = f(x_{m-1}, y_{m-1}),$$

or

$$y_m = y_{m-1} + hf(x_{m-1}, y_{m-1}). \tag{15.1}$$

This is the **Euler method.** It is meant, of course, to give approximate answers where a closed-form solution is unknown, or where we want values for the closed-form solution. Let us apply it to

the familiar $dy/dx = y, y(0) = 1$ equation, with $b = 1$ and $h = 1/N$. Here it is easy to work out exactly what gets calculated (ignoring rounding errors, of course).

$$y_m = y_{m-1} + hy_{m-1}.$$

or

$$y_m = (1 + h)y_{m-1},$$

so clearly

$$y_m = (1 + h)^m.$$

Recalling that $h = 1/N$, we get

$$y_N = \left(1 + \frac{1}{N}\right)^N.$$

It can be shown using a McLaurin Series and the Binomial Theorem that the right-hand side converges to e as $N \to \infty$, so y_m is an approximation to $e^{m/N} = e^{x_m}$, as it should be.

Here is the data with $N = 10$.

```
Euler method
x_m = 0.000000, y_m = 1.000000, e^x - y = 0.000000
x_m = 0.100000, y_m = 1.100000, e^x - y = 0.005171
x_m = 0.200000, y_m = 1.210000, e^x - y = 0.011403
x_m = 0.300000, y_m = 1.331000, e^x - y = 0.018859
x_m = 0.400000, y_m = 1.464100, e^x - y = 0.027725
x_m = 0.500000, y_m = 1.610510, e^x - y = 0.038211
x_m = 0.600000, y_m = 1.771561, e^x - y = 0.050558
x_m = 0.700000, y_m = 1.948717, e^x - y = 0.065036
x_m = 0.800000, y_m = 2.143589, e^x - y = 0.081952
x_m = 0.900000, y_m = 2.357948, e^x - y = 0.101655
x_m = 1.000000, y_m = 2.593742, e^x - y = 0.124539
```

Accuracy of this method. If $y_m = y(x_m)$ then

$$y(x_{m+1}) - y_{m+1} = O(h^2)$$

by Taylor's Theorem. Suppose $x_0 = a$ and $x_N = x_0 + Nh = b$. The accumulated error at b is roughly

$$O(\sum h^2) = O(Nh^2) = O(h).$$

The accuracy is *linear* in h.

15.1 Explicit trapezoidal method

Euler's method is analogous to approximating an integral using a bar-chart, whereas the trapezoidal method uses a piecewise-linear approximation and Simpson's rule uses a piecewise-quadratic approximation.

If we knew y_{m+1} exactly, which is ridiculous, a better estimate of the derivative would be the average at y_m and y_{m+1}. We can't do that, but we can, so to speak, plug in the Euler version of y_{m+1} and use it to get a better-balanced estimate of the derivative.

This leads to the *explicit trapezoid method, aka improved Euler method or Heun method*: Let $a = f(x_m, y_m)$ and $b = f(x_m + h, y_m + hf(x_m, y_m))$.

$$y_{m+1} = y_m + \frac{h}{2}(a + b)$$

It can be shown that the per-step error in the improved Euler method is $O(h^3)$. Applying it to solving $dy/dx = y, y(0) = 1$:

```
 x_0 = 0.000000,  y_0 = 1.000000, e^x - y = 0.000000
 x_1 = 0.100000,  y_1 = 1.105000, e^x - y = 0.000171
 x_2 = 0.200000,  y_2 = 1.221025, e^x - y = 0.000378
 x_3 = 0.300000,  y_3 = 1.349233, e^x - y = 0.000626
 x_4 = 0.400000,  y_4 = 1.490902, e^x - y = 0.000923
 x_5 = 0.500000,  y_5 = 1.647447, e^x - y = 0.001275
 x_6 = 0.600000,  y_6 = 1.820429, e^x - y = 0.001690
 x_7 = 0.700000,  y_7 = 2.011574, e^x - y = 0.002179
 x_8 = 0.800000,  y_8 = 2.222789, e^x - y = 0.002752
 x_9 = 0.900000,  y_9 = 2.456182, e^x - y = 0.003421
x_10 = 1.000000, y_10 = 2.714081, e^x - y = 0.004201
```

Much better than the ordinary Euler method.

16 Taylor's Theorem

(16.1) Proposition (Taylor's Theorem) *Making certain assumptions about the existence and continuity of the derivatives of a function $f(x)$ near $x = a$,*

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \frac{(x-a)^2}{3!}f^{(3)}(a) + \ldots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \frac{(x-a)^{n+1}}{(n+1)!}f^{(n+1)}(X)$$

for some X between a and x.

For example, if we take $f(x) = (1+x)^3$ and $a = 0$,

$$\begin{aligned} f(0) &= 1 \\ f'(0) &= 3 \\ f''(0) &= 6 \\ f'''(0) &= 6 \\ f''''(0) &= 0 \end{aligned}$$

so

$$(1+x)^3 = 1 + 3x + 3x^2 + x^3 + 0$$

the remainder term being zero.

If we take $f(x) = \sin(x)$, $a = 0$,

$$f(0) = 0 f'(0) = 1 f''(0) = 0 f'''(0) = -1$$

and the pattern repeats in blocks of 4.

$$\sin(x) = x - \frac{x^3}{3!}\frac{x^5}{5!} + \ldots + \frac{x^{n+1}}{(n+1)!}\frac{f^{(n+1)}(X)}{(n+1)!}$$

The remainder term is bounded by $1/(n+1)!$ in absolute value — the series converges quickly if x is reasonably small.

While Taylor's Theorem is a little tricky to prove, there is a similar result (Cauchy form of the remainder) which one can prove using integraton by parts.

We shall show how to prove a slightly different version of Taylor's Theorem, based on the method of *integration by parts*

$$\begin{aligned} \frac{d}{dt}(uv) &= u\frac{dv}{dt} + v\frac{du}{dt} \\ \int u\frac{dv}{dt}dt &= uv - \int v\frac{du}{dt}dt \end{aligned}$$

Now

$$f(x) = f(a) + \int_a^x f'(t)dt$$

Using (this may look a little odd)

$$\frac{d}{dt}(t-x)f'(t) = f'(t) + (t-x)f''(t)$$

we get

$$\begin{aligned} [(t-x)f'(t)]_a^x &= \int_a^x f'(t)dt + \int_a^x (t-x)f''(t)dt \\ (x-a)f'(a) &= \int_a^x f'(t)dt + \int_a^x (t-x)f''(t)dt \\ f(x) &= f(a) + (x-a)f'(a) - \int_a^x (t-x)f''(t)dt \end{aligned}$$

Using

$$\frac{d}{dt}\frac{(t-x)^2}{2!}f''(t) = (t-x)f''(t) + \frac{(t-x)^2}{2!}f'''(t)$$

we get

$$\begin{aligned}
\left[\frac{(t-x)^2}{2!}f''(t)\right]_a^x &= \int_a^x (t-x)f''(t)dt + \int_a^x \frac{(t-x)^2}{2!}f'''(t)dt \\
-\frac{(x-a)^2}{2!} &= \int_a^x (t-x)f''(t)dt + \int_a^x \frac{(t-x)^2}{2!}f'''(t)dt \\
-\int_a^x (t-x)f''(t)dt &= \frac{(x-a)^2}{2!}f''(a) + \int_a^x \frac{(t-x)^2}{2!}f'''(t)dt \\
f(x) &= f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \int_a^x \frac{(t-x)^2}{2!}f'''(t)
\end{aligned}$$

This is easily generalised by induction to

$$f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \ldots + \frac{(x-a)^n}{n!}f^{(n)}(a) + \int_a^x \frac{(x-a)^n}{n!}f^{(n+1)}(t)dt$$

This is the *Cauchy form of the remainder.* The other, more commonly used, version can be deduced using arguments based on continuity.

17 Partial derivatives

If $f(x, y)$ is a bivariate function then, under certain continuity assumptions, one can define *partial derivatives*

$$\begin{aligned}
\frac{\partial f}{\partial x} &= \lim_{h\to 0} \frac{f(x+h, y) - f(x, y)}{h} \\
\frac{\partial f}{\partial y} &= \lim_{k\to 0} \frac{f(x, y+k) - f(x, y)}{k}
\end{aligned}$$

They are no harder to compute than ordinary derivatives: when computing $\partial f/\partial x$, treat y as a constant; for $\partial f/\partial y$, treat x as a constant.

For example,

$$\begin{aligned}
f(x, y) &= xy + \cos(x/y) \\
\frac{\partial f}{\partial x} &= y - \frac{1}{y}\sin(x/y) \\
\frac{\partial f}{\partial y} &= x + \frac{x}{y^2}\sin(x/y)
\end{aligned}$$

There are forms of Taylor's Theorem for two variables and more. In particular, if f is continuously differentiable,

$$f(x+h, y+k) = f(x, y) + h\frac{\partial f}{\partial x} + k\frac{\partial f}{\partial y} + O(h^2 + k^2)$$

18 Runge-Kutta methods

We begin with one of the simpler Runge-Kutta schemes.

$$\begin{aligned} y_0 &= c \\ g_1 &= f(x_{m-1}, y_{m-1}) \\ g_2 &= f(x_{m-1} + \alpha h, y_m + \beta h g_1) \\ y_m &= y_{m-1} + h(A_1 g_1 + A_2 g_2), \end{aligned}$$

where $h = (b-a)/N$ is the step-size and α, β, A_1, A_2 are constants.

There is some freedom in choosing these constants. We shall derive some relations involving them which will ensure that the method has $O(h^3)$ error in each step.

The choice of these constants is based on Taylor's Series.

$$y(x_m) = y(x_{m-1}) + hy'(x_{m-1}) + \frac{h^2}{2} y''(x_{m-1}) + O(h^3).$$

Now, $y'(x) = f(x, y)$ and

$$y''(x) = \frac{d}{dx} f(x,y) = \frac{\partial}{\partial x} f(x,y) + y'(x) \frac{\partial}{\partial y} f(x,y) = \frac{\partial}{\partial x} f(x,y) + f(x,y) \frac{\partial}{\partial y} f(x,y).$$

Therefore

$$\begin{aligned} y(x_m) = \\ y(x_{m-1}) \\ + hf(x_{m-1}, y(x_{m-1})) \\ + \frac{h^2}{2} \left(\frac{\partial}{\partial x} f(x_{m-1}, y(x_{m-1})) + f(x_{m-1}, y(x_{m-1})) \frac{\partial}{\partial y} f(x_{m-1}, y(x_{m-1})) \right) \\ + O(h^3). \end{aligned}$$

Given $g_1 = f(x_{m-1}, y_{m-1})$ and

$$g_2 = f(x_{m-1} + \alpha h, y_{m-1} + \beta g_1 h),$$

we can expand g_2 as a Taylor's series:

$$g_2 = f(x_{m-1}, y_{m-1}) + \alpha h \frac{\partial f}{\partial x}(x_{m-1}, y_{m-1}) + \beta h f(x_{m-1}, y_{m-1}) \frac{\partial f}{\partial y}(x_{m-1}, y_{m-1}) + O(h^2).$$

Then if $y_m = y_{m-1} + h(A_1 g_1 + A_2 g_2)$,

$$\begin{aligned} y_m = \\ y_{m-1} \\ + hA_1 f(x_{m-1}, y_{m-1}) \\ + hA_2 \left(f(x_{m-1}, y_{m-1}) + \alpha h \frac{\partial f}{\partial x}(x_{m-1}, y_{m-1}) + \beta h f(x_{m-1}, y_{m-1}) \frac{\partial}{\partial y} f(x_{m-1}, y(x_{m-1})) \right) \\ + O(h^3). \end{aligned}$$

Assuming y_{m-1} is correct, i.e., $y_{m-1} = y(x_{m-1})$, we can compare terms and conclude

$$\begin{cases} A_1 + A_2 = 1 \\ A_2\alpha = \frac{1}{2} \\ A_2\beta = \frac{1}{2}. \end{cases}$$

There is some freedom of choice of these constants.

When $A_1 = 0$, we get the *modified Euler method*:

$$g_1 = f(x_m, y_m), \quad g_2 = f(x_m + h/2, y_m + hg_1/2)$$
$$y_{m+1} = y_m + hg_2$$

Heun's method $A_1 = A_2 = 1/2$, $\alpha = \beta = 1$.

$$g_1 = f(x_m, y_m), \quad g_2 = f(x_m + h, y_m + hg_1) y_{m+1} = y_m + h(g_1 + g_2)/2$$

The modified Euler method has already been demonstrated.

Accuracy of this Runge-Kutta method. The error term at each step is $O(h^3)$ (Taylor's Theorem), hence the cumulative error is $O(h^2)$.

18.1 A very good Runge-Kutta method

$$s_1 = f(x_i, y_i)$$
$$s_2 = f(x_i + h/2, y_i + hs_1/2)$$
$$s_3 = f(x_i + h/2, y_i + hs_2/2)$$
$$s_4 = f(x_i + h, y_i + hs_3)$$
$$y_{i+1} = y_i + \frac{h}{6}(s_1 + 2s_2 + 2s_3 + s_4)$$

The stepwise error is $O(h^4)$, cumulative $O(h^3)$ (much too hard to analyse). Applying it again to the initial value problem $dy/dx = y, y(0) = 1$:

```
 x_0 = 0.000000,  y_0 = 1.000000, e^x - y = 0.000000
 x_1 = 0.100000,  y_1 = 1.105171, e^x - y = 0.000000
 x_2 = 0.200000,  y_2 = 1.221403, e^x - y = 0.000000
 x_3 = 0.300000,  y_3 = 1.349858, e^x - y = 0.000000
 x_4 = 0.400000,  y_4 = 1.491824, e^x - y = 0.000000
 x_5 = 0.500000,  y_5 = 1.648721, e^x - y = 0.000001
 x_6 = 0.600000,  y_6 = 1.822118, e^x - y = 0.000001
 x_7 = 0.700000,  y_7 = 2.013752, e^x - y = 0.000001
 x_8 = 0.800000,  y_8 = 2.225540, e^x - y = 0.000001
 x_9 = 0.900000,  y_9 = 2.459601, e^x - y = 0.000002
x_10 = 1.000000, y_10 = 2.718280, e^x - y = 0.000002
```

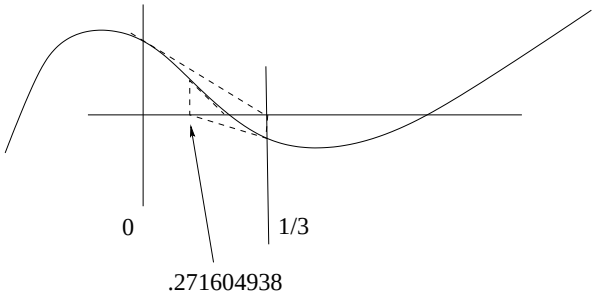

Figure 1: bracketing the root: not to scale.

19 More on Newton-Raphson

It is usually possible to measure the accuracy of one's Newton-Raphson approximation to a root of $f(x)$, using the Mean Value Theorem.

If r is a root and a is an approximation to t, we know that $f(r) = 0$, and also

$$f(r) = f(a) + (r - a)f'(X)$$

for some X between r and a. Therefore

$$r = a - \frac{f(a)}{f'(X)}$$

exactly, except that we usually don't know what X is. But if we

- know some reasonably small interval $[x_0, x_1]$ containing r, and
- the *sign* of $f'(x)$ is unchanged over $[x_0, x_1]$,
- no, better still, that $f'(x)$ is strictly incrasing or strictly decreasing on the interval, and doesn't change sign, then
- r lies between $a - f(a)/f(x_0)$ and $a - f(a)/f(x_1)$.

We shall apply this to the problems on the first quiz.

The function $p(x) = 10x^3 - 5x^2 - 3x + 1$.

Sequence

$$0,\ \ 1/3,\ \ 0.271604938,\ \ 0.276372283,\ \ 0.276393202$$

and $p(a) = 0$ to calculator accuracy where a is last on list.

One thing which should have been checked at the start is that there is one sign change at $1/3$, so the root we want is in the interval $(0, 1/3)$.

The sign of p is positive at 0 and negative at $1/3$, so over this interval the sign of p is positive to the left of r and negative to its right.

A calculation shows that $f'(x)$ changes sign only once in $[0, 1]$, at $.524126$. Also, $f''(x)$ changes sign only at $1/6$ (inflection). See Figure 1.

To be completed. It will turn out that (working with computer bc calculator to 10 places decimal) the root is between

$$.2763932017 + .0000000014/f$$

where $3 < f < 3.5$. This means adding 4 or 5 to the last place;

$$.2763932021$$

or

$$.2763932022$$

(rounded up).

Explanation. Let $a = .2763932017$; $p(a) = .0000000014$. Between $.1666666667$ and $.524126$, $p'(x)$ is negative and ($f''(x) = 60x - 10$) increasing.

Since there is one root r in the interval of interest, and p changes from $+$ to $-$ at r, in the interval, x is left of r if and only if $p(x) > 0$.

Thus $a < r$, and since p' is increasing to -3 at $1/3$, $p'(a)$ is a lower bound for $p'(X)$ between a and $1/3$. Since $p'(a) = -3.472135958650$,

$$a + .0000000014/3.472135958650 \leq r \leq a + .0000000014/3$$
$$a + .000000000403 \leq r \leq a + .000000000466$$
$$.276393202103 \leq r \leq .276393202166$$

(to 12 decimal places).

20 Draft syllabus for 2015 exam

- The overall mark will take 20% coursework and 80% in the May exam.

□

  The May exam will have 4 questions of which you will be asked to answer 3.

- Euclid's gcd algorithm
- Sturm sequences, Sturm's theorem (possible proof asked)
- Newton-Raphson (see extra notes)
- Converting fractions to floating-point format, with answer in hex.
- IEEE standard, with emphasis on the guard, round, sticky bits applied to toy examples. Also, $N_{\max}$ etcetera for toy examples.
- The numbers γ_n and theorem 7.3, which might be asked.
- Accuracy of summation (proof might be asked).
- Gaussian elimination and partial pivoting.
- LU factorisation: be able to to this on 2x2 and 3x3 matrices, even with pivoting. Know Proposition 11.2 and how to apply it to $2x2$ examples, but the proof will not be asked.

- Numerical integration. Accuracy of the Trapezoidal method, using Taylor's Theorem. Know Simpson's Rule and its accuracy.
- Proof of Taylor's Theorem will not be asked.
- ODEs. Euler's Method. You may be asked to prove the $O(h^2)$ per-step accuracy, using Taylor's Theorem.
- Runge-Kutta methods. You may be asked to prove the $O(h^3)$ per-step accuracy of the given kinds of Runge-Kutta, using various flavours of Taylor's Theorem.
- The best method: be able to apply it and the other three methods covered in the module, for comparison.