Numerical Methods

No. 1

Date 1 . 5 . 19

Errors

- Computations involve errors because of noisy data, floating repr, etc.
- Absolute error: $\varepsilon_x = |x^* - x| \Rightarrow x^* = x \pm \varepsilon_x$ ← x^* is our approx.
- Relative error: $\eta_x = \Delta x / |x| \Rightarrow x^* = x(1+\eta_x)$
- For $x^* \pm y^*$, $\varepsilon_{x+y} = \varepsilon_x + \varepsilon_y$. No quadrature because these are not distributions.
  - ↳ be careful if subtraction results in near-zero because $\eta \uparrow$ a lot
- The error in $f(x^*)$ is $\varepsilon_{f(x)} \approx |f'(x^*)| \varepsilon_x$.
- If $f(x^*)$ is a truncated Taylor series, we have the Lagrange error bound:
  $\varepsilon_{f(x)} \leq \frac{M(\varepsilon_x)^{n+1}}{(n+1)!}$ where M is the upper bound on $|f^{(n+1)}(k)|$, for $k \in (x, x^*)$ or $k \in (x^*, x)$.
- Forward error analysis propagates perturbations in the input
  - ↳ sometimes overestimates because we propagate worst case but really there will be some cancellation.
- Backward error analysis instead finds the input error associated with a given output data and verifies it is within the uncertainty of the data
  e.g. if $f(x) = x^2$ then $f^*(x) = (x^2)^* = x^2(1+\eta_{x^2})$.
  We can then find some $\tilde{\eta}$ such that $(1+\tilde{\eta})^2 = 1+\eta$
  $\Rightarrow f^*(x) = x^2(1+\tilde{\eta})^2 = f(x(1+\tilde{\eta}))$. $\tilde{\eta}$ can be compared with the unc. of the data.

Numerical differentiation

- For analytic functions, $f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n$ ← Taylor series
- This allows the first approximation that $f'(x_0) = \frac{f(x) - f(x_0)}{x - x_0}$
- Alternatively, we can utilise the discretisation parameter h; the forward approx is then $D_{f,1}^{+}(x) = \frac{f(x+h) - f(x)}{h}$
  - ↳ by expanding $f(x+h)$ and rearranging for $f'(x)$, we can show that the truncation error is $f'(x) - D_{f,1}^{+}(x) = -hf''(x)/2 + O(h^2)$

- The symmetric approx is $D_{fi}^{0}(x) = \frac{f(x+h) - f(x-h)}{2h}$
  - ↳ truncation error is $h^2 f'''(x)/3! + O(h^3)$
  - ↳ quadratic in h i.e second-order method
- Generally, an nth order method has truncation error $O(h^n)$
  - ↳ N.B: not the same as order of convergence

Numerical integration

- A Riemann sum approximates $\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} f(t_i)(x_{i+1} - x_i)$
  - ↳ $t_i \in [x_i, x_{i+1}]$ is the point within the interval at which height is evaluated.
- The trapezoidal rule approximates $f(x)$ as piecewise linear

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \left(f(x_{i+1}) + f(x_i)\right) \frac{b-a}{2n}, \qquad x_i = a + (b-a)\,i/n$$

- Simpson's rule treats $f(x)$ as piecewise quadratic (fits $\cap$ to 3 points)

$$\int_a^b f(x)dx \approx \frac{b-a}{3n}\left(f(x_0) + 4f(x_1) + \underbrace{2f(x_2) + 4f(x_3)}_{\text{alternates}} + \dots + 4f(x_{n-1}) + f(x_n)\right)$$

  - ↳ n must be even
  - ↳ fourth-order method
- Higher order polynomial interpolation is not always better.

- Monte Carlo integration can be useful for high dimension integrals
  - randomly sample a domain of known area
  - count number of points that fall within a target region
  - area of target is approx'd by the fraction of points that are within target.

Iterative methods

- Iterative methods all have an iterative step and some termination criterion, e.g error small enough, time, change between iterations.
- Error can be analysed by substituting our guess back into the eq or finding $|x_n - x_{n-1}|$, though the latter is unreliable.
- Convergence may not be guaranteed, or can be slow.
- kth order convergence $\Rightarrow$ $\varepsilon_n = M\varepsilon_{n-1}^K$ for $M>0$ as $n\to\infty$.
  - ↳ linear convergence is slow
  - ↳ 2nd order (quadratic) convergence means that num sig. digits doubles at each iteration (i.e much faster).
- Most common application of iterative methods is root finding.

Bisection

1. Choose a, b such that $\text{sign}(f(a)) \neq \text{sign}(f(b))$ (this is actually quite difficult to do quickly)
2. Find midpoint $c = (a+b)/2$
3. If $|f(c)| <$ desired_accuracy stop.
4. If $\text{sign}(f(c)) = \text{sign}(f(a))$ then $a=c$ else $b=c$
5. GOTO 2

- Similar to a binary search for the root
- First order convergence

Fixed point iteration

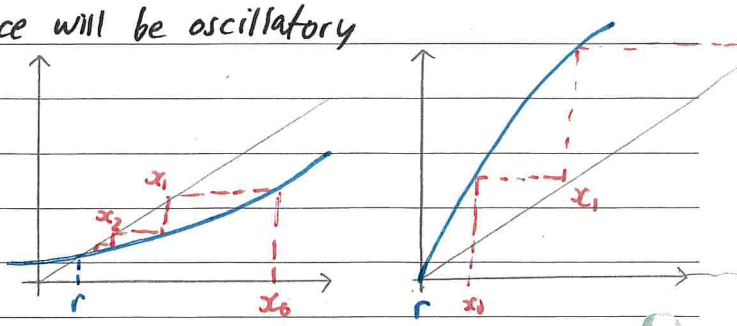
- Rewrite $f(x)=0$ as $g(x)=x$ for some g. Iteratively compute $x_{n+1} = g(x_n)$. If this converges to r, r is a fixed point of g.
- Sufficient condition for convergence
  - $g : I \to I$ for some interval I
  - $|g'(x)| < 1$ for $\forall x \in I$.

- This is because the error depends on the gradient:

$$x_n = r + \varepsilon_n \implies r + \varepsilon_{n+1} = g(r+\varepsilon_n) \approx g(r) + \varepsilon_n g'(r)$$
$$\implies \varepsilon_{n+1} \approx \varepsilon_n g'(r)$$

  - if $0 < g'(x) < 1$, convergence will be monotonic
  - if $-1 < g'(x) < 0$, convergence will be oscillatory

- Convergence/divergence can be analysed graphically by bounding with $y = x$



- Sufficient condition for divergence:
  if $|g'(r)| > 1$ for all fixed points r, then they are repelling fixed points and all (x_n) will diverge (unless $x_0 = r$).
- Thus we must carefully choose $g(x)$ and x_0 to fall under the convergence criterion.
- Fixed point theory can be used to prove that there exist solutions.

Newton-Raphson

- NR is a special case of fixed point iteration: $x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
  ↳ error can be analysed by considering the 1st order Taylor expansion of $f(r+x_n)$, with $\varepsilon_n = r - x_n$

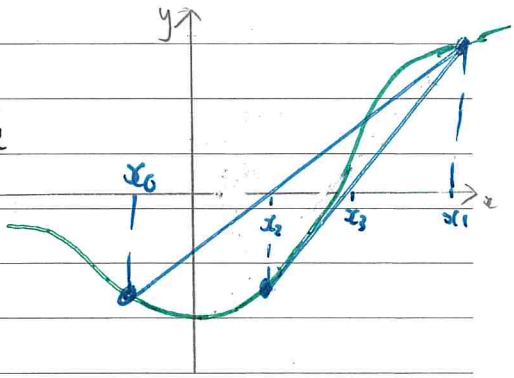
$$|f(r) - f^*(r)| = |f(r) - f(x_n) - \varepsilon_n f'(x_n)| \leq \frac{M\varepsilon_n^2}{2}$$

Using $\frac{f(x_n)}{f'(x_n)} = \varepsilon_{n+1} - \varepsilon_n$ from NR:

$$|\varepsilon_{n+1}| \leq \frac{M\varepsilon_n^2}{2|f'(x_n)|}$$

- In practice, x_0 may be found as the root of a crude linear approx to $f(x)$.
- If the $f'(x)$ is badly behaved near $x = r$, e.g. $f(x) = |x|^{0.5}$, then NR may overshoot and thus diverge/oscillate.
- If the root multiplicity of roots > 1, we must use modified NR to preserve quadratic convergence: $x_{n+1} = x_n - m\frac{f(x_n)}{f'(x_n)}$ ← m is multiplicity.

Secant method



- NR requires $f'(x_n)$. We might approximate this by using a secant between any two points.

$$\therefore x_{n+1} = x_n - f(x_n)\frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

- The convergence is superlinear:

$$\varepsilon_{n+1} = \varepsilon_n - f(x_n)\frac{\varepsilon_n - \varepsilon_{n-1}}{f(x_n) - f(x_{n-1})} \approx \varepsilon_n \varepsilon_{n-1}$$

$$\therefore \varepsilon_{n-1}^{p^2-p-1} = M. \quad \varepsilon_{n-1} \to 0 \text{ for convergence}$$

$$\therefore p^2 - p - 1 = 0$$

- Thus the order of convergence is ϕ.
- Faster than bisection, but doesn't guarantee convergence unless we ensure that we only construct secants between points of opposite sign (can be done with caching).

Gradient descent

- To find the minimum, we look in the direction of fastest descent

$$\underset{\sim}{x}_{n+1} = \underset{\sim}{x}_n - \gamma_n \nabla f(\underset{\sim}{x}_n)$$

↑ learning rate

- We then repeat this until $\|\nabla f(\underset{\sim}{x}_n)\| < \varepsilon$
- This is a fixed point iteration.
- γ_n can be chosen by an adaptive scheme: choose γ_n that minimises the projection of f onto the plane given by $\nabla f(\underset{\sim}{x}_n)$

  i.e $\gamma = \arg\min_\gamma f(\underset{\sim}{x}_n - \gamma \nabla f(\underset{\sim}{x}_n))$
- Gradient descent only finds local minima (unless f is convex)
- Slow convergence when going from side to side down a valley.

Simulated annealing

- A Monte Carlo method for finding the global minimum.
  ↳ in each iteration, a random candidate is jumped to with some probability.

Start with x_0 and some temperature T_0

1. Randomly select $c_{n+1} = C(x_n, T_n)$ e.g $C(x,T) = x + \text{uniform}(-1,1)e^T$
2. $x_{n+1} = c_{n+1}$ with probability $P(f(x_n), f(c_{n+1}), T)$

   e.g $P(y, y', T) = \frac{1}{1+e^{\frac{y'-y}{T}}}$
3. Decrease temp, e.g $T_{n+1} = A(T_n)$ with $A(T) = 0.999T$

- As $T \to 0$, $P(y, y', T) \to 0$ if $y' > y$ i.e over time we are less likely to jump to a higher value.

Linear Systems

- All about solving matrix equations of the form $Ax = b$
- The simplest way is to find A^{-1}, but ~~dot~~ this is unstable for large matrices.

Gaussian elimination

- Uses the principle that multiples of rows can be added to achieve an upper-triangular form matrix

```
for each row a_i:
    for j in range(0, i):
        set a_{i,j} to zero by  a_i = a_i - (a_{i,j} / a_{j,j}) a_j
```

- Then back-substitute to find unknowns, which is $O(n^2)$
- $O(n^3)$ in total

- If the pivot element is too small, the coefficients may become huge
  ↳ rows can be interchanged, so we should choose the one with the largest leading value.
  ↳ columns can be permuted but then rows in $\underline{x}$ need to be as well.

LU factorisation

- If we can write A as the product of lower and upper triangular forms, we can easily solve for x:
  - solve $L\underline{w} = \underline{b}$ using forward sub
  - then solve $U\underline{x} = \underline{w}$ with back sub.
- Dolittle algorithm: do a normal Gaussian elimination on A to get U, but keep track of the steps in a new matrix
  - start with $L = \begin{pmatrix} 1 & 0 & 0 \\ \heartsuit & 1 & 0 \\ \heartsuit & \heartsuit & 1 \end{pmatrix}$.
  - e.g. if $R2 \to R2 - 3R1$ then $L = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$.
  - $O(n^3)$, small overhead on Gauss
- Unlike Gaussian elimination, $\underline{b}$ is not part of the decomposition. Thus we can factorise $A = LU$ then apply it to a $\underline{b}$ in $O(n^2)$.
- If $A_{1,1}$ is 0, the algorithm will fail even though a factorisation exists
  ↳ first permute the rows, corresponding to $PA = LU$.
  ↳ any square matrix has an LUP factorisation.

- LU can be used to determine if A has an inverse: A^{-1} exists iff all L/U diagonals are non zero
  ↳ $A^{-1} = U^{-1}L^{-1}$, with inverse of $\triangle$ matrix easy to compute.

Cholesky factorisation

- The matrix A is positive definite if $v^T A v > 0$ for all v:
  - eigenvalues are positive
  - restrictions how far an input vector v can be rotated
  - alternatively, positive semi-definite (≥ 0), negative definite (< 0) ...
- If A is symmetric and positive definite, LU fact. can be optimised by setting $U = L^T$

$$LL^T = \begin{pmatrix} L_{00}^2 & \text{SYMMETRIC} & \\ L_{10}L_{00} & L_{10}^2 + L_{11}^2 & \\ L_{20}L_{00} & L_{20}L_{10} + L_{21}L_{11} & L_{20}^2 + L_{21}^2 + L_{22}^2 \end{pmatrix} = A$$

- We can then forward substitute to find L_{ij}
- The Cholesky decomp can be used to test for positive definiteness, by seeing if the decomp exists.
- We can find A^{-1} using $A^{-1} = A^T(AA^T)^{-1}$, where AA^T is symmetric

Ill-conditionedness

- It is important to know how a small input perturbation affects the output.
- The condition number is the maximum ratio of the relative change in output to a relative change in input
  ↳ problem is ill-conditioned if it has a high condition number
- Conditionedness is a property of the problem, not algorithm

$$K(x) = \lim_{\varepsilon \to 0} \left| \frac{f(x+\varepsilon) - f(x)}{f(x)} \Big/ \frac{\varepsilon}{x} \right| = \left| \frac{x f'(x)}{f(x)} \right|$$

- For linear systems, we want to know how a vector perturbation of $\underline{b}$ affects the value of $\underline{x}$.

$$K(A) = \max_{x, \varepsilon \neq 0} \frac{\|A^{-1}\varepsilon\|}{\|x\|} \Big/ \frac{\|\varepsilon\|}{\|Ax\|} = \max_{\varepsilon \neq 0} \frac{\|A^{-1}\varepsilon\|}{\|\varepsilon\|} \times \max_{x \neq 0} \frac{\|Ax\|}{\|x\|}$$

- When using the Cholesky decomposition, we may end up with negatives due to rounding errors → problem in sqrt. A solution is to add a diagonal matrix: $A = LDL^T = \begin{pmatrix} 1 & 0 & 0 \\ l & 1 & 0 \\ l & l & 1 \end{pmatrix} \begin{pmatrix} d & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & d \end{pmatrix} \begin{pmatrix} 1 & l & l \\ 0 & 1 & l \\ 0 & 0 & 1 \end{pmatrix}$

  ↳ LDL^T decomp has the same $O(n^3)$ complexity as LL^T.

QR factorisation

- If we write $A = QR$, with Q being orthogonal and R being upper triangular form, we can easily solve. $y = Q^T b$ then $Rx = y$.
  - this works because $Q^{-1} = Q^T$
  - also works when A and R are tall
- To build a set of orthogonal vectors starting from vectors a_k in A, we use the Gram-Schmidt algorithm.
- Treat A, Q, R as their column vectors. Gram-Schmidt is like Gaussian elimination but we are subtracting to make orthogonal.

1. Initialise $q_0 = \frac{a_0}{\|a_0\|}$, $r_0 = \begin{pmatrix} \|a_0\| \\ 0 \\ 0 \end{pmatrix}$

2. Using a_1, construct a vector orthogonal to q_0, called w_1. Then normalise to get q_1

$$w_1 = a_1 - (q_0 \cdot a_1) q_0$$

then $q_1 = \frac{w_1}{\|w_1\|}$ $\quad r_1 = \begin{pmatrix} q_0 \cdot a_1 \\ \|w_1\| \\ 0 \\ \vdots \end{pmatrix}$

3. $$w_k = a_k - \sum_{i=0}^{k-1} (q_i \cdot a_k) q_i$$

$$q_k = \frac{w_k}{\|w_k\|}$$

Then we have $Q = (q_1 | q_2 | \dots | q_m)$

$$R = \begin{pmatrix} a_1 q_1 & a_2 q_1 & \dots & a_n q_1 \\ 0 & a_2 q_2 & & a_n q_2 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & 0 & a_n q_m \end{pmatrix}$$

- Small errors in the dot product can accumulate and hinder orthogonality.

Least Squares

- A linear model has the form $y = \beta_0 + \beta_1 x$
- The measurements are then $y_i = \beta_0 + \beta_1 x + \varepsilon_i$, where ε_i are statistical errors (i.e. diff between measured and true value).
- The residual is the difference between the measurement and our model's prediction: $r_i = y_i - (\beta_0 + \beta_1 x_i)$
- We want to find $\hat{\beta}_0$ and $\hat{\beta}_1$ to minimise the sum of squared residuals

$$S(\beta_0, \beta_1) = \sum_i r_i^2 = \sum_i (y_i - \beta_0 - \beta_1 x_i)^2$$

$$\frac{\partial S}{\partial \beta_1} = 0 \text{ and } \frac{\partial S}{\partial \beta_0} = 0 \Rightarrow \hat{\beta}_1 = \frac{\sum xy - n\bar{x}\bar{y}}{\sum x^2 - n\bar{x}^2}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

- This is linear least squares because it is linear in β, so the partial derivatives give linear equations (this is why we use squares)
  - ↳ Generally: $y = \sum_{j=1}^{p} \beta_j \phi_j(x)$ e.g. $y = \beta_0 + \beta_1 x^2 + \beta_2 x^4 + \dots + \beta_p x^p$.
  - ↳ We don't care what ϕ_j is. We can even use different independent variables $x_1, x_2, \dots x_p$
- Can reformulate with matrices:

observations ↓

$$\begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} x_{11} & \cdots & x_{p1} \\ \vdots & \ddots & \vdots \\ x_{1n} & \cdots & x_{pn} \end{pmatrix} \begin{pmatrix} \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \vdots \\ \varepsilon_n \end{pmatrix}$$

$$\Rightarrow y = X\beta + \varepsilon$$

$$S(\beta) = \|r\|^2 = (y - X\beta)^T (y - X\beta)$$

grad w.r.t β →

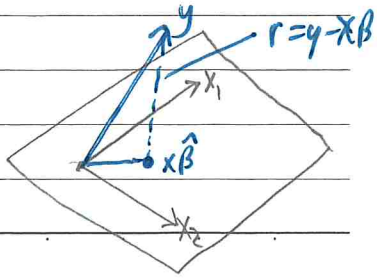
$$\therefore \nabla S(\beta) = \nabla (y^T y - 2\beta^T X^T y + \beta^T X^T X \beta)$$

think of $\beta^T \beta$ as β^2 ∴ $2X^T X\beta$

$$= 0 - 2X^T y + 2X^T X\beta = 0$$

$$\therefore \hat{\beta} = (X^T X)^{-1} X^T y$$

- This has a geometric interpretation:
  - the vectors of X span some subspace.
  - we want to get as close to y as possible, so we project
  - $r = y - X\beta$ is orthogonal to the subspace
    ∴ $X^T(y - X\beta) = 0$



R^2

- X^TX isn't invertible $\iff$ columns linearly dependent
  - ↳ use Cholesky to check. If it succeeds we can easily invert LL^T as $(L^T)^{-1}L^{-1}$.
  - ↳ if singular, some of our variables are independent.
- Rather than explicitly computing $(X^TX)^{-1}$ to solve, we can just solve $(X^TX)\beta = X^Ty$ using any of our techniques.

Goodness of fit

- The coefficient of determination (R^2) is defined by

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y}_m)^2}$$

← "unexplained" variance of residuals
← total variance

  - ↳ hence R^2 is the explained variance
- Adding a new independent variable cannot decrease R^2, in the worst case its β will be zero and R^2 unchanged.
  - ↳ we can thus use the adjusted R^2: $R^2_{adj} = 1 - (1-R^2)\frac{n-1}{n-p-1}$

Eigenvalues and Eigenvectors

- Matrices are linear transformations: $A(\alpha x + \beta y) = \alpha Ax + \beta Ay$
- Av will only span the space of v if A is nonsingular.
- Eigenvectors are only scaled (or rotated) when A is applied, $Av = \lambda v$.
- Solving the characteristic polynomial is very ill-conditioned, thus we want better numerical algorithms.
- Power iteration can be used to find the dominant eigenvalue for a diagonalisable matrix.
  - ↳ starting with any nonzero vector, we repeatedly apply the matrix A: this converges to the eigenvector $x_{k+1} \approx x^{(k+1)} = \frac{Ax^{(k)}}{\|Ax^{(k)}\|}$
  - ↳ the corresponding eigenvalue can be found using the Rayleigh quotient: $\lambda = \frac{x^TAx}{x^Tx}$

- If A is diagonalisable, it can be written as $Q \Lambda Q^{-1}$, with Q being a matrix of eigenvectors and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots \lambda_n)$
  - $A^k = (Q \Lambda Q^{-1})(Q \Lambda Q^{-1}) \dots (Q \Lambda Q^{-1}) = Q \Lambda^k Q^{-1}$
  - our initial random vector may be written as Qx for some x.

$$\therefore A^k(Qx) = Q\Lambda^k x = \sum q_j \lambda_j^k x_j$$

$$\therefore A^k(Qx) = \lambda_1^k \left(\sum q_j \left(\frac{\lambda_j}{\lambda_1} \right)^k x_j \right)$$

  - because λ_1 (in our case) is the dominant value, $\lim_{k \to \infty} \left(\frac{\lambda_j}{\lambda_1} \right)^k = 0$

$$\therefore A^k(Qx) \to \lambda_1^k q_1 x_1$$ hence converges to eigenvector.

- Because power iteration only has linear convergence, it may not be suitable for huge matrices (e.g PageRank)
- A far fast iterative method uses QR factorisation instead.

$$A_k = Q_k R_k$$
$$A_{k+1} = Q_k^T A_k Q_k$$

↳ A_k converges to upper triangular, hence we can easily find all eigenvalues as the diagonal entries.

- For a real symmetric matrix: there is a set of n eigenvectors that form an orthonormal basis.
  ↳ similarly, if we have n unique eigenvalues, their eigenvectors will be orthogonal.

- Principal component analysis (PCA) is all about representing a dataset in a basis that maximises variance
  - these PCs will be eigenvectors, and because $x^T x$ is real symmetric, they will be orthogonal
  - it may be the case that only a few PCs are needed to show most of the variation.

No. 7

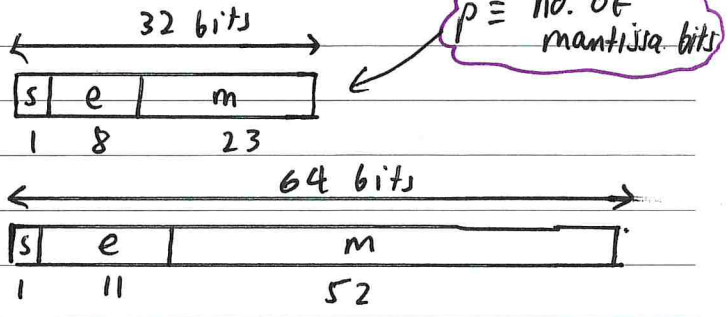
Date 23 · 5 · 19

Floating point representation

- A given 8-bit binary number can be interpreted as signed (Two's complement) or unsigned: $1000\ 1011 \rightarrow 2^7+2^3+2^1+2^0 = 139$, $-2^7+2^3+2^1+2^0 = -117$
- Non-integer values could be represented by a **fixed point**, which reserves some bits for integer / fractional parts
  ↳ choosing position of decimal point is a tradeoff between prec. and range.
- **Floating points** are written as $(-1)^s \times m \times \beta^e$
  - s is the sign
  - m is the **mantissa** (i.e sig. digits)
  - β is the base
  - e is the exponent.
- The **IEEE 754** standard specifies constraints:
  - because all mantissas start with 1, this is a **hidden bit** (not stored).
  - in order for e to be stored unsigned (easier for radix sort), we include a **bias**, i.e $(-1)^s \times 1.m \times \beta^{e-b}$

single precision (32 bits)
↳ bias $= 2^{8-1}-1 = 127$

double precision (64 bits)
↳ bias $= 2^{11-1}-1 = 1023$



- For a single prec number, the exponent can be used to ~~represen~~ represent -126 to $+127$, i.e e is from 1 to 254 inclusive. This is because e=0 and e=255 are reserved for special values:
  - zero needs a special representation because the hidden bit is 1.
  - **denormalised numbers** allow for extra small numbers by allowing m to have leading 0s.
  - NaN allows us to recognise when a result isn't useful without aborting the computation.

	m=0	m≠0
e=0	zero	denormalised
e=MAX	∞	NaN

- Care must be taken because exact fractions in decimal may be recurring in binary, e.g. $0.1_{10} = 0.0001100110011\ldots_2$
- IEEE zeroes and infinities are signed, which can be useful to signify where a value started.

IEEE Arithmetic

- Treat operands as precise, then round the result to the nearest IEEE number. If there is a tie, choose the value with 0 LSB (even).
- This rounding (unbiased) is the default, but we can also choose rounding towards 0, $+\infty$, $-\infty$, using a global flag.
- For other functions, 'perfect accuracy' is considered 0.5 ulp (unit in the last place)
- Using floats, errors will come from quantisation and rounding errors.

Machine epsilon

- ε_m is defined as the difference between 1.0 and the smallest representable number greater than 1. $\varepsilon_m = \beta^{-(p-1)}$
  ↳ upper bound on the relative error caused by being off by 1 ulp.
- In IEEE, the diff between 1.0 and the greatest number smaller than 1.0 is $\varepsilon_m/2$.

Range reduction

- Consider trying to evaluate $\sin x$ for large x using its Maclaurin series
- To avoid loss of significance, we can use modular arithmetic to bring x to $[-\pi, \pi)$. This is range reduction.
- However, these large numbers will have a large absolute error (even though $\eta = \varepsilon_m$), hence it will be difficult to know which cycle x falls into unless π is stored to very high precision.