Computer arithmetic & algorithms

$d_m d_{m-1}...d_0.d_{-1}d_{-2}...d_{-n} = d_m \times 10^m + d_{m-1} \times 10^{m-1} + ...+ d_0 \times 10^0 + d_{-1} \times 10^{-1} + d_{-2} \times 10^{-2} + ...+ d_{-n} \times 10^{-2}$

$b_m b_{m-1}...b_0.b_{-1}...b_{-n} = b_m \times 2^m + b_{m-1} \times 2^{m-1} + ...+ b_0 \times 2^0 + b_{-1} \times 2^{-1} + ... + b_{-n} \times 2^{-n}$

base 10 (decimal), $d_m d_{m-1}...d_0$: integer, $d_{-1}d_{-2}...d_{-n}$: fractional

converting binary to decimal

$d_m d_{m-1}...d_0.d_{-1}d_{-2}...d_{-n} = (b_r b_{r-1}...b_0.b_{-1}b_{-2}...)_2$

$d_m d_{m-1}...d_0 = (b_r b_{r-1}...b_0)_2$ & $d_{-1}d_{-2}...d_{-n} = (b_{-1}b_{-2}...)_2$

To find integer part, - divide $d_m d_{m-1}...d_0$ by 2 (remainder = b_0)

- divide quotient by 2 (remainder = b_1)...repeat process until quotient is 0

For fractional part, - multiply $.d_{-1}d_{-2}...d_{-n}$ by 2 (integer part = b_{-1})

- multiply new fractional part by 2 (integer part = b_{-2})...repeat until fractional part is 0; otherwise fractional part is infinite

converting decimal to binary

integer = integer & fractional = fractional

E.g. 2.4 = 10.011001100110...

2/2 = 1R0. 1/2 = 0R1

0.4 x 2 = 0 + 0.8   0.8 x 2 = 1 + 0.6

0.6 x 2 = 1 + 0.2   0.2 x 2 = 0 + 0.4

$0.01100110... = 2^{-2} + 2^{-3} + 2^{-6} + 2^{-7} + ... = (2^{-2} + 2^{-3})\sum_{i=0}^{\infty} 2^{-4i} = \frac{3}{8}\left(\frac{1}{1-2^{-4}}\right) = 0.4$

Floating-point formats

Binary num in scientific notation normalized form

$(b_r b_{r-1}...b_0.b_{-1}...b_{-n})_2 = \pm(1.b_{r-1}...b_0 b_{-1}...b_{-n})_2 \times 2^r$ $(0.00101_2 = 1.01_2 \times 2^{-3})$

$\pm(1.s_1 s_2...s_N)_2 \times 2^k$

IEEE standard	Number of bits		
precision	sign	exponent	mantissa (N)
single	1	8	23
double	1	11	52
long double	1	15	64

Computer use bit (binary digit 0 or 1) to store info about the digits / mantissa ($s_1, s_2,...s_N$), exponent k (in binary form as well) and sign (0 for +ve and 1 for -ve)

Note that exponent will use 1 bit for its sign and the other bits for its binary form

-Use double precision for this module

Smallest +ve normalized double precision floating-point num: $+(1.00...00 \times 10^{-1111111110})_2 = 1 \times 2^{-1022} \approx 2.22 \times 10^{-308}$

$+(1.00..00 \times 10^{-1111111111})$ reserved for $+\infty$

Largest +ve normalized double precision floating-point num: $+(1.11...11 \times 10^{+1111111111})_2 = (2 - 2^{-52}) \times 2^{1023} \approx 1.8 \times 10^{308}$

machine epsilon = dist btw 1 and smallest floating point num > 1 and for double precision, $\epsilon_{mach} = 2^{-52}$

Rounding rule

IEEE Rounding to Nearest Rule

1. 53rd bit = 0: truncate after 52nd bit
2. 53rd bit = 1:
   a. 54th bit onwards all 0 & 52nd bit = 0: truncate after 52nd bit
   b. else 1 added to 52nd bit

Applies to both normal and subnormal num

$(1.*...*_{52}0_{53})_2 \times 2^k = (1.*...*_{52})_2 \times 2^k$

$(1.*...*0_{52}1_{53})_2 \times 2^k = (1.*...*0_{52})_2 \times 2^k$

$(1.*...*1_{52}1_{53})_2 \times 2^k = (*.*...*0_{52})_2 \times 2^k$

$(1.*...**_{52}1_{53}...*1*...)_2 \times 2^k = (*.*...**_{52})_2 \times 2^k$

Subnormal floating point

For num smaller than $2^{-1022} \approx 2.2 \times 10^{-308}$, subnormal floating point number is used: $\pm(0.s_1 s_2...s_{52})_2 \times 2^{-1022}$

So, smallest +ve double precision num = $(0.00...1)_2 \times 2^{-1022} = (1 \times 2^{-52}) \times 2^{-1022} = 2^{-1074}$

Although num below ϵ_{mach} is machine representable, adding to 1 may have no effect

Computer arithmetic

For a num x, fl(x) = num stored in computer (not exact)

Use special symbol to represent computer arithmetic

$x \oplus y = fl(fl(x) + fl(y))$ $x \ominus y = fl(fl(x) - fl(y))$...

Matrix multiplication

$A = (a_{ij})_{m \times n} \in \mathbb{R}^{m\times n}$, $B = (b_{ij})_{n \times p} \in \mathbb{R}^{n\times p}$, $C = AB = (c_{ij})_{m \times p} \in \mathbb{R}^{m\times p}$

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

$c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$, $\forall$ i = 1,...,m, j = 1,...,p

O(mpn). Exact is O(mpn + mp(n-1)) = O(mp(2n-1))

```
initialise C = (c_ij)_{m x p} to be zero matrix
for i = 1,..., m do
  for j = 1,...,p do
    c_ij ← a_i1 b_1j
    for k = 2,..., n do
      c_ij ← c_ij + a_ik b_kj
Result: C = (c_ij)_{m x p}.
```

3 main qns: Find i, k s.t. $a_{ik} \neq 0$. Find k, j s.t. $b_{kj} \neq 0$. Find i, j s.t. $c_{ij} \neq 0$

Lower triangular multiplication

Suppose A and B are lower triangular n x n matrix, C = AB also lower triangular

$c_{ij} = 0\ \forall\ i < j$, $C = (c_{ij})_{n \times n}$

For i ≥ j: – A is lower triangular: $a_{ik} = 0$, $\forall$ i < k

– B is lower triangular: $b_{kj} = 0$, $\forall$ k < j

– $a_{ik}b_{kj} = 0$ if i < k or k < j. So multiplication only needed for k ≤ i or k ≥ j

$c_{ij} = \sum_{k=j}^{i} a_{ik}b_{kj}$, 1 ≤ j ≤ i ≤ n

```
initialise c_ij = 0 for all 1 ≤ i < j ≤ n
for i = 1,..., n do
  for j = 1,...,i do
    c_ij ← a_ij b_jj
    for k = j+1,..., i do
      c_ij ← c_ij + a_ik b_kj
```

Num of additions: $\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j+1}^{i} 1 = \sum_{i=1}^{n}\sum_{j=1}^{i}(i-j)$

$= \sum_{i=1}^{n}[(i-1)+\ldots+1]$

$= \sum_{i=1}^{n}\frac{i(i-1)}{2} = \frac{1}{2}\left(\frac{n(n+1)(2n+1)}{6} - \frac{n(n+1)}{2}\right) = \frac{(n-1)n(n+1)}{6}$

Num of multiplications: $\sum_{i=1}^{n}\sum_{j=1}^{i}(1 + \sum_{k=j+1}^{i} 1) = \sum_{i=1}^{n}\sum_{j=1}^{i} 1 + \sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=j+1}^{i} 1 = \sum_{i=1}^{n} i$ + num of additions $= \frac{n(n+1)}{2} + \frac{(n-1)n(n+1)}{6} = \frac{n(n+1)(n+2)}{6}$

$\sum_{i=1}^{n} 1 = (n - 1) + 1$ $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$ = num of terms * (1st term + last term) / 2 $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Upper Hessenberg multiplication

$A = (a_{ij})_{n \times n}$ be tridiagonal matrix =

$$\begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 \\ 0 & a_{32} & \ddots & \ddots & 0 \\ 0 & 0 & \ddots & \ddots & a_{n-1,n} \\ 0 & 0 & 0 & a_{n,n-1} & a_{n,n} \end{pmatrix},$$

B be upper triangular and

C = AB be upper Hessenberg matrix = $(c_{ij})_{n \times n}$

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & \cdots & c_{1n} \\ c_{21} & c_{22} & c_{23} & \cdots & \vdots \\ 0 & c_{32} & \ddots & \ddots & \vdots \\ 0 & 0 & \ddots & \ddots & c_{n-1,n} \\ 0 & 0 & 0 & c_{n,n-1} & c_{n,n} \end{pmatrix}$$

$c_{ij} = 0$, ∀j < i - 1, 3 ≤ i ≤ n

For j ≥ i - 1: – $a_{ik} = 0$, ∀k < i - 1, 3 ≤ i ≤ n or k > i + 1, 1 ≤ i ≤ n-2

– $b_{kj} = 0$, ∀k > j

– $a_{ik}b_{kj} = 0$, ∀k < i - 1, 3 ≤ i ≤ n or k > min(i+1, j), 1 ≤ i ≤ n - 2

≠ 0, ∀i - 1 ≤ k ≤ min(i+1,j), ∀i = 1,...,n, ∀j = max(1,i-1),..., n

$c_{ij} = \sum_{k=max(1,i-1)}^{min(i+1,j)} a_{ik}b_{kj}$, ∀i = 1,...,n, ∀j = max(1,i-1),..., n

```
initialise c_ij = 0 for all i,j, where 1 ≤ j < i - 1 < n
for i = 1,..., n do
  for j = max(1,i - 1),...,n do
    k ← max(1, i - 1);   c_ik ← a_ik b_kj
    for k = max(2, i),...., min(i + 1, j) do
      c_ij ← c_ij + a_ik b_kj
```

Num of add: $\sum_{i=1}^{n}\sum_{j=max(1,i-1)}^{n}\sum_{k=max(2,i)}^{min(i+1,j)} 1 =$

$\sum_{i=1}^{n}\sum_{j=max(1,i-1)}^{n}(min(i+1,j) - max(2,i) + 1) =$

$\sum_{j=1}^{n}(min(2,j) - 1) + \sum_{i=2}^{n}\sum_{j=i-1}^{n}(min(i+1,j) - i + 1) =$

$1 + 2 + 2 + ... + 2 - n + \sum_{i=2}^{n}(0 + 1 + 2 + \ldots + 2) =$

$1 + 2(n-1) - n + \sum_{i=2}^{n}(1 + 2(n-(i+1)+1) =$

$n - 1 + \sum_{i=2}^{n}(1 + 2n - 2i) = n - 1 + n - 1 + 2n(n-1) - 2(\frac{n(n+1)}{2} - 1) = n^2 - n$

Num of multiply: $\sum_{i=1}^{n}\sum_{j=max(1,i-1)}^{n}(1 + \sum_{k=max(2,i)}^{min(i+1,j)} 1) =$

$\sum_{i=1}^{n}\sum_{j=max(1,i-1)}^{n} 1$ + num of add =

$\sum_{i=1}^{n}(n - max(1, i-1) + 1)$ + num of add =

n(n+1) - [1 + 1 + 2 + 3 + ... + (n - 1)] + num of add =

$n(n+1) - [1 + \frac{n(n-1)}{2}] + n^2 - n = \frac{1}{2}(3n-2)(n+1)$

Intermediate Value Theorem (IVT): Let f be cts fn on [a,b], where f(a)f(b) < 0. Then ∃ r ∈ (a,b) s.t. f(r) = 0   set a ← a_o, b ← b_0

Bisection Mtd	Bisection mtd: Solve for eqn: f(r) = 0. Keep dividing interval into 2 until length of new interval/2 ≤ TOL. Soln: exact root r = approx root ± TOL (tolerance) After n bisection steps: Approximate root = midpoint of $(a_n, b_n) = (a_n + b_n)/2$ Error of approx soln = $\lvert \text{exact root - approx root} \rvert \le \frac{b_n-a_n}{2} = \frac{1}{2}\left(\frac{b-a}{2^n}\right) = \left(\frac{b-a}{2^{n+1}}\right)$ Num of fn evaluations = n + 2 (f(a), f(b), n times of f(c))	while (b-a)/2 > TOL do c ← (a+b)/2 if f(c) == 0: stop else if f(a)f(c) < 0: b ← c else a ← c Result: appox root = (a+b)/2
	Soln is correct within p d.p if error is less than 0.5 x 10^{-p}. Num of steps for bisection: $\frac{b-a}{2^{n+1}} < 0.5 \times 10^{-p}$ However, due to rounding errors by computers, may terminate early and not have required digits of precision.	
Fixed-Point Iteration (FPI)	Solve eqn: g(x) = x. Keep applying g() to x_i until num converge. r is a fixed pt if g(r) = r Backward error = $\lvert g(x_i+1) - x_{i+1}\rvert = \lvert f(x_a)\rvert$ Suppose f(x) = g(x) - x, and x_a is approximation for r where f(r) = 0 Forward error = $\lvert r-x_a\rvert$ Note that seq x_i may not always converge. Note any eqn f(x) = 0 can be turned into a fixed-pt problem g(x) = x	set initial guess x_0 for i = 0,1,2,... do (have max i in case divergent) $x_{i+1} \leftarrow g(x_i)$ if $\lvert g(x_i+1) - x_{i+1}\rvert$ < TOL: $x_{fp} \leftarrow g(x_{i+1})$, break Result: x_{fp}
FPI convergence rate	Convergence Thrm: Suppose g is continuously differentiable, and g(r) = r and S := $\lvert g'(r)\rvert$ < 1. Then FPI converges linearly with rate S to r for initial guesses sufficiently close to r Let $e_i := \lvert x_i - r\rvert$ denote error at step i. If $\lim_{i\to\infty}\frac{e_{i+1}}{e_i} = S < 1$, mtd obey linear convergence with rate S So S = 0: fastest convergence rate. S < 1: cfm converge. S > 1: won't converge	Fixed-Point Thrm: Let g be cts fn in [a,b] s.t. g(x) ∈ [a,b], ∀ x ∈ [a,b] and g' exists on (a,b) and ∃ constant 0 < k < 1 s.t. $\lvert g'(x)\rvert \le k$, ∀ x ∈ [a,b]. Then for any num x_o in [a,b], seq $x_n = g(x_{n-1})$, n ≥ 1 converges to unique fixed pt r in [a,b]
	Bisection Mtd: Guaranteed to converge linearly. However, need predefine initial interval and convergence rate = 1/2 (might be slower than FPI)	FPI: convergence rate S can be < 1/2, faster than bisection. But might be diff to formulate g from f
Horner's Mtd	Given x, evaluate h(x). Common that h is polynomial. $P_m(x) = a_o + a_1x + ... + a_{m-1}x^{m-1} + a_mx^m = a_0 + x(a_1 + x(a_2 + ... + x(a_{m-1} + xa_m)))$ Horner's Mtd: most optimal mtd for finding value of polynomial. O(m) $p_m = a_m$. $p_{m-1} = a_{m-1} + xp_m$. ... $p_1 = a_1 + xp_2$. $p_0 = a_0 + xp_1$	$P \leftarrow a_m$ for k = m-1,...,0 do $P \leftarrow a_k + xP$ Result: P

Gaussian Elimination $O(n^3)$ Leading term: $\frac{2n^3}{3}$	$A_{n\times n}x_{n\times 1} = b_{n\times 1}$ Note lower triangular part not set to zero as they would not be used in further computation Subtraction: n(n-1)(2n+5)/6 (without labelling rows) Multiplication: n(n-1)(2n+5)/6 (without labelling rows) Division: n(n+1)/2 (without labelling rows) Note zero column = non-invertible Could have cases where $\mathbf{a_{r_i i}} = 0$. So label rows to keep track of swapping rows Due to computer arithmetic errors, num supposed to be 0 becomes a very small num and cause loss of significant digits	for i = 1,...,n: $r_i \leftarrow i$ (labelling rows) for i = 1,...,n-1: { j ← i while j ≤ n and $a_{r_j i} = 0$: j ← j+1 (check which col has pivot entry) if j = n+1: Output: "Error: Singular matrix" (means have zero col) else if j ≠ i: swap r_j and r_i for j = i+1,...,n: $m_{ji} \leftarrow a_{r_j i}/\mathbf{a_{r_i i}}$ (elimination: making matrix into upper triangular) for k = i+1,...,n+1: (augmented matrix include $b_{n\times 1}$ as well) $a_{r_j k} \leftarrow a_{r_j k} - m_{ji}a_{r_i k}$} $x_n \leftarrow a_{r_n,n+1}/a_{r_n n}$ (backward substitution) for i = n-1,...,1: {$x_i \leftarrow a_{r_i,n+1}$ for j = i+1,...,n: $x_i \leftarrow x_i - a_{r_i j}x_j$ $x_i \leftarrow x_i/a_{r_i i}$}
Partial Pivoting	1st mtd to perfrom row swap to minimise errors Select pivot elem s.t. its absolute value is as large as possible in col Num of comparisons is $\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2}$ However, if remaining entries of pivot row also has large magnitude, then would cause loss of significant digits in the other rows.	(labelling rows) for i = 1,..., n-1 {l ← i for j = i+1,...,n {if $\lvert a_{r_j i}\rvert > \lvert a_{r_l i}\rvert$: l ← j} if $a_{r_l i} = 0$: Output: "Error: Singular matrix"(row swap) else if l ≠ i: swap r_j and r_i } (elimination), (backward substitution)
Relative absolute ratio	A way to do this is to compare the relative absolute sizes in each col j = 1,...,n , relative absolute size of row i = a_{ij}/max(row i) By choosing largest relative absolute size for each col as pivot elem, loss of sig digits is minimised Divisions = $\frac{(n+2)(n-1)}{2}$, Comparisons = $\frac{n(n-1)(2n+5)}{6}$	for i = 1,..., n-1 for j = i,..., n do (find max abs val in row) $s_{r_j} \leftarrow \lvert a_{r_j i}\rvert$ for k = i+1,..., n {if $\lvert a_{r_j k}\rvert > s_{r_j}$: $s_{r_j} \leftarrow \lvert a_{r_j k}\rvert$} l ← i, max ← $\lvert a_{r_i i}\rvert/s_{r_i}$ (find max relative abs ratio) for j = i+1,...,n {ratio ← $\lvert a_{r_j i}\rvert/s_{r_j}$ if ratio > max: l ← j, max ← ratio}
Scaled Partial Pivoting (SPP)	2nd mtd to perform row swap to minimise errors To save computational cost, assume max entry of row does not change too much in elimination process, and only find max for each row once at beginning, then use this max from original row to calculate relative absolute ratio: scaled partial pivoting Comparisons = $\frac{3}{2}n(n-1)$	(labelling rows) for i = 1,..., n {$s_i \leftarrow \lvert a_{i1}\rvert$ for j = 2,..., n {if $\lvert a_{ij}\rvert > s_i$: $s_i \leftarrow \lvert a_{ij}\rvert$} if $s_i = 0$: Output: "Error: Singular matrix"} for i = 1,..., n-1 {(find max relative abs ratio), (row swap)} (elimination), (back substitution)
LU factorization $O(n^3)$ + $O(2pn^2)$	Solve for multiple L.S with same coefficient matrix, $Ax_1 = b_1$, $Ax_2 = b_2$..., $Ax_p = b_p$. So can preprocess A to not repeat ops. Find A = LU, where L is lower triangular, U: upper triangular. LUx = b. L = strictly lower part of processed A + diagonal all 1. U = upper triangular of processed A	for i = 1,..., n-1 do (LU factorization) for j = i+1,..., n do $a_{ji} \leftarrow a_{ji}/a_{ii}$ #store L in lower triangular part of A for k = i+1,..., n {$a_{jk} \leftarrow a_{jk} - a_{ji}a_{ik}$} #elimination
	Forward substitution (solve Ly = b for y) Backward substitution (solve Ux = y for x) Time complexity for both substitution = $O(n^2)$ So, applying Gaussian elimination p times = $O(2pn^3)$	$y_1 \leftarrow b_1$ (forward sub) for j = 2,..., n do $y_j \leftarrow b_j$ for i = 1,..., j-1: $y_j \leftarrow y_j - L_{ji}y_i$ $x_n \leftarrow y_n/a_{nn}$ (backward sub) for i = n-1,...,1 {$x_i \leftarrow y_i$ for j = i+1,..., n: $x_i \leftarrow x_i - U_{ij}x_j$ $x_i \leftarrow x_i/U_{ii}$}

PA = LU factorization $O(n^3)$ + $O(pn^2)$	Matrix which require row swap to get REF cannot be LU factorized. (or can check det of top left entry, top left 2 x 2 entries, top left 3 x 3 ... ≠ 0 then can be factorized).
	Hence, need perform row swap at start with SPP, PAx = Pb, where P is a permutation matrix (n x n matrix
		Find max abs values of each row and initialise L, U, r
		for i = 1,..., n-1 do (PA = LU factorization)
		(find max relative abs ratio & row swap)
		$L_{r_i,i} \leftarrow 1$, $U_{ii} \leftarrow a_{r_i,i}$
		for j = i+1,..., n do
		
		(new forward sub)
		$y_1 \leftarrow b_{r_1}$
		for j = 2,..., n do
		$y_j \leftarrow b_{r_j}$
		for i = 1,..., j-1:

consisting all 0, except for a single 1 in every row and col)
Now PA = LU. Solve Ly = Pb, then solve Ux = y
No need to explicitly find P, just output r (stored row index) to replace P

$L_{r_j,i} \leftarrow a_{r_j,i}/a_{r_i,i}$
for k = i+1,..., n: $a_{r_j,k} \leftarrow a_{r_j,k} - L_{r_j,i}a_{r_i,k}$
$U_{ij} \leftarrow a_{r_i,j}$
$L_{r_n,n} \leftarrow 1$, $U_{n,n} \leftarrow a_{r_n,n}$

$y_j \leftarrow y_j - L_{r_j,i}y_i$

backward sub still same as previous

A = LU & PA = LU

If n is large, memory to store L and U might be consuming

Special matrix

Symmetric positive-definite matrix. n x n matrix A is symmetric if $A^T = A$. A is positive-definite if $x^TAx > 0$ ∀ col vector x ≠ 0.
To check if matrix is positive-definite, could expand algebraically with x and then complete the sq to check > 0.
OR If A is symmetric, A is positive-definite iff all eigenvalues > 0
Principal submatrix of sq matrix A is a sq submatrix whose diag entries = diag entries of A. (a_{11}, a_{22}, a_{33}, $\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$, $\begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix}$, $\begin{pmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{pmatrix}$)
Any principal submatrix of a symmetric positive-definite matrix is symmetric positive-definite.

Cholesky factorization $O(n^3) + O(pn^2)$

Cholesky factorization: every symmetric positive-definite matrix A can be factored as $A = R^TR$ (R is upper triangular). Hence would save roughly half the memory compared to A = LU.
Idea: Use row/col ops to reduce A into identity matrix to get $A = R^TR$.
Forward sub: $R^Ty = b$ for y. Backward sub: Rx = y for x. $O(n^2)$

1a. Apply row, col ops to introduce zeros into 1st col & row
1b. Apply row, col scaling so 1st diagonal entry becomes 1
2a. Apply row, col ops to introduce zeros into 2nd col & row
2b. Apply row, col scaling so 2nd diagonal entry becomes 1...
n. Apply row and col scaling so last diagonal entry is 1

Note $\tilde{A} = K_1 - \frac{1}{a_{11}}u_1u_1^T$ also positive definite.
A = $R_1^T A_1 R_1$. $(R_1^T)^{-1}AR_1^{-1} = A_1$. Since $x^T(R_1^T)^{-1}AR_1^{-1}x = (R_1^{-1}x)^TA(R_1^{-1}x)$ and A is symm positive definite, then $(R_1^T)^{-1}AR_1^{-1} = A_1$ also symm positive definite. Since $\tilde{A}$ is principal submatrix of A_1, $\tilde{A}$ also symm positive definite
for k = 1,2,..., n do
  if $A_{kk} < 0$: Stop algo. A not positive definite
  $R_{kk} \leftarrow \sqrt{a_{kk}}$ (diag entries of R)
  $v^T \leftarrow \frac{1}{R_{kk}}A_{k,k+1:n}$; $R_{k,k+1:n} \leftarrow v^T$ (rest of row k)
  $A_{k+1:n,k+1:n} \leftarrow A_{k+1:n,k+1:n} - vv^T$ (principal submatrix)

A = $\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{13} & a_{23} & a_{33} \end{pmatrix} = \begin{pmatrix} a_{11} & u_1^T \\ u_1 & K_1 \end{pmatrix} = \begin{pmatrix} \sqrt{a_{11}} & \frac{u_1^T}{\sqrt{a_{11}}} \\ 0 & I \end{pmatrix}^T \begin{pmatrix} 1 & 0^T \\ 0 & K_1 - \frac{1}{a_{11}}u_1u_1^T \end{pmatrix} \begin{pmatrix} \sqrt{a_{11}} & \frac{u_1^T}{\sqrt{a_{11}}} \\ 0 & I \end{pmatrix} = R_1^T A_1 R_1$ where $u_1 = \begin{pmatrix} a_{12} \\ a_{13} \end{pmatrix}$, $K_1 = \begin{pmatrix} a_{22} & a_{23} \\ a_{23} & a_{33} \end{pmatrix}$. Then repeat finding for $\tilde{A} = K_1 - \frac{1}{a_{11}}u_1u_1^T$ and so on...Finally will get $R^T = \begin{pmatrix} \sqrt{a_{11}} & \frac{a_{12}}{\sqrt{a_{11}}} & \frac{a_{13}}{\sqrt{a_{11}}} \\ 0 & \sqrt{\widetilde{a_{11}}} & \frac{\widetilde{a_{12}}}{\sqrt{\widetilde{a_{11}}}} \\ 0 & 0 & \sqrt{\widehat{a_{11}}} \end{pmatrix}^T$

Jacobi Method $O(n^2)$

Similar to fixed-point iteration. $A_{n \times n}$ = L + D + U
Ax = b. (L + D + U)x = b. Dx = b - (L + U)x. x = D^{-1}(b- (L + U)x)
D^{-1} is just reciprocal of all entries in D. So $x^{(k+1)} = D^{-1}(b - Lx^{(k)} - Ux^{(k)})$
Then solve eqn element-wise, $x_1^{(k+1)} = \frac{1}{a_{11}}\left[b_1 - a_{12}x_2^{(k)} - \ldots - a_{1n}x_n^{(k)}\right]$
$x_2^{(k+1)} = \frac{1}{a_{22}}\left[b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} \ldots - a_{2n}x_n^{(k)}\right]$...
$x_n^{(k+1)} = \frac{1}{a_{nn}}\left[b_n - a_{n1}x_1^{(k)} - \ldots - a_{n,n-1}x_{n-1}^{(k)}\right]$
So calculation can be parallelized

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 0 & 0 & \cdots & 0 \\ a_{21} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{pmatrix} + \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix} + \begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1,n} \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix} = L + D + U$$

set initial vector $x^{(0)}$
d ← diag(A); R ← A - diag(d) (to get L + U)
for k = 0,1,... do
  $x^{(k+1)}$ = (b - $Rx^{(k)}$)⊘d (⊘ = element-wise division)

Terminate algo when max iteration is reached
OR $x^{(k+1)}$ approximately solve L.S, i.e. $\max_{1 \le i \le n}\{|Ax^{(k+1)} - b|_i\} < \varepsilon$
A is stricly diagonally dominant (sdd) if for each 1 ≤ i ≤ n, $|a_{ii}| > \sum_{j \ne i}|a_{ij}|$, i.e. diagonal entry > sum of non-diagonal entries in same row = Jacobi mtd converge
If A sdd, A is nonsingular matrix. If A not sdd, MIGHT still converge

Another mtd to check convergence is spectral radius p(B) = max magnitude of eigenvalues of B. If p(B) < 1, and c is arbitrary, then for any vector x_0, $x_{k+1} = Bx_k + c$ converges. In particular, check $p(D^{-1}(L + U)) < 1$. Use $\det(D^{-1}(L + U) - \lambda I) = 0$ to find eigenvalues.

Gauss-Seidel Method $O(n^2)$

Similar to Jacobi. Now, $x^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)})$
$x_1^{(k+1)} = \frac{1}{a_{11}}\left[b_1 - a_{12}x_2^{(k)} - \ldots - a_{1n}x_n^{(k)}\right]$ (no change)
$x_2^{(k+1)} = \frac{1}{a_{22}}\left[b_2 - a_{21}\boldsymbol{x_1^{(k+1)}} - a_{23}x_3^{(k)} \ldots - a_{2n}x_n^{(k)}\right]$...
$x_n^{(k+1)} = \frac{1}{a_{nn}}\left[b_n - a_{n1}\boldsymbol{x_1^{(k+1)}} - \ldots - a_{n,n-1}\boldsymbol{x_{n-1}^{(k+1)}}\right]$
Now cannot parallelized, as x_2,... x_n dependent on x_1, ..., x_{n-1}
But since updated values are used, will converge faster than Jacobi
If A is sdd, Gauss-Seidel mtd will converge

set initial vector $x^{(0)}$
for k = 0,1,2,... do
  for i = 1,2,...n do
    $x_1^{(k+1)} \leftarrow b_i$
    for j = i+1,..., n: { $x_i^{(k+1)} \leftarrow x_i^{(k+1)} - a_{ij}x_j^{(k)}$}
    for j = 1,..., i-1: { $x_i^{(k+1)} \leftarrow x_i^{(k+1)} - a_{ij}x_j^{(k+1)}$}
    $x_i^{(k+1)} \leftarrow \frac{x_i^{(k+1)}}{a_{ii}}$

Successive Over-Relaxation (SOR) Method $O(n^2)$

Gauss-Seidel iterate: $x_{GS}^{(k+1)} = D^{-1}(b - Lx^{(k+1)} - Ux^{(k)})$
Current iterate: $x^{(k)}$
Let ω be a real num, and define $x^{(k+1)}$ as weighted average of $x_{GS}^{(k+1)}$ and $x^{(k)}$. i.e. $x^{(k+1)} = (1 - \omega)x^{(k)} + \omega x_{GS}^{(k+1)}$
ω is called relaxation parameter and ω>1 = over-relaxation
GS: ω = 1. SOR: ω > 1
Faster convergence than GS
Need to choose ω wisely, usually 1.1 or 1.2

set initial vector $x^{(0)}$
for k = 0,1,2,... do
  for i = 1,2,...n do
    $x_1^{(k+1)} \leftarrow b_i$
    for j = i+1,..., n: { $x_i^{(k+1)} \leftarrow x_i^{(k+1)} - a_{ij}x_j^{(k)}$}
    for j = 1,..., i-1: { $x_i^{(k+1)} \leftarrow x_i^{(k+1)} - a_{ij}x_j^{(k+1)}$}
    $x_i^{(k+1)} \leftarrow (1 - \omega)\, x_i^{(k)} + \omega\, \frac{x_i^{(k+1)}}{a_{ii}}$

When to use which mtd?

Direct mtd: Gaussian elimination, A = LU or PA = LU, Cholesky factorization. $O(n^3)$ for preprocessing, $O(n^2)$ for finding
Iterative mtd: Jacobi, Gauss-Seidel, SOR. $O(n^2)$

Use iterative mtd if 1. requirement of accuracy not high, save computational cost
2. good approximation already known (to be used as initial guess)
3. If A is sparse (many entries = 0). Most expensive op is matrix-vector multiplication which would be cheaper

Interpolation

Given data points $(x_0, f(x_0))$, $(x_1, f(x_1))$, ..., $(x_n, f(x_n))$. Want to find fn g to connect these pts to restore original fn f
Interpolation: restoration of f
Interpolant / Interpolating fn: g
Interpolation nodes: x_0, x_1, ..., x_n

Properties of g: Simplicity (easy to evaluate) and approximability (accuracy)
Weierstrass approximation theorem: Let f be a cts fn on [a, b]. For any $\epsilon > 0$, ∃ a polynomial P(x) s.t. $|f(x) - P(x)| < \epsilon$, ∀ x ∈ [a, b]
If we have n+1 data points, polynomial must be of degree n, with n+1 coeff
From general eqn for polynomial, sub data points in to get n+1 eqns and solve L.S to find coeff

Basis fn

Basis fn: $\{\varphi_0(x), \varphi_1(x), ..., \varphi_n(x)\}$, where $\varphi_k(x_j) = \delta_{jk}$,

g must satisfy: $g(x_0) = f(x_0)$, $g(x_1) = f(x_1)$, ..., $g(x_n) = f(x_n)$

Kronecker delta, $\delta_{jk} = \begin{cases} 0, if\ j \neq k \\ 1, otherwise \end{cases}$, j = 0,1,..., n, k = 0,1, ..., n

Using basis fn, g(x) = f(x_0)φ_0(x) + f(x_1)φ_1(x) + ... + f(x_n)φ_n(x)
E.g. g(x_0) = f(x_0)(1) + f(x_1)(0) + ... + f(x_n)(0) = f(x_0)

Lagrange basis polynomial

Let $x_0, x_1, ..., x_n$ be the n+1 distinct real nums. For k = 0, 1, ..., n, the k^{th} Lagrange basis polynomial $L_k(x)$ is a polynomial of degree n where
$L_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x-x_j}{x_k-x_j} = \frac{(x-x_0)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)}$. So $L_k(x_j) = \delta_{jk}$

Then interpolating fn, P(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + ... + f(x_n)L_n(x)
which satisfies P(x_0) = f(x_0), P(x_1) = f(x_1), ..., P(x_n) = f(x_n)
P(x) is called the Lagrange interpolating polynomial

LS is guaranteed to have a soln if (deg n, num of eqns m)
m=n: By constructing Lagrange interpolating polynomial
m > n: Not guaranteed
m < n: Infinitely many solution

Using LS: Pros – direct mtd | Cons: tedious computations
Lagrange: Pros – easy to analyse. Good if we need to interpolate many fns with same set of interpolating nodes | Cons: not convenient to add more data points

Uniqueness of Lagrange polynomial

If $x_0, x_1, ..., x_n$ are n + 1 distinct nums and f is a fn whose values are given at these nums, then a unique polynomial $P_n(x)$ of degree at most n exists with f(x_k) = $P_n(x_k)$, for k = 0,1,...,n
$P_n(x)$ = f(x_0)L_0(x) + f(x_1)L_1(x) + ... + f(x_n)L_n(x) where L_k is the k^{th} Lagrange basis polynomial

Suppose there is another polynomial $\tilde{P}_n$ having deg ≤ n satisfying $\tilde{P}_n(x_k)$ = f(x_k), for k = 0,1,...,n. Then $\tilde{P}_n(x_k)$ - $P_n(x_k)$ = 0, for k = 0,1,...,n
This means x_k, k = 0,1,...,n are roots of polynomial $\tilde{P}_n(x_k)$ - $P_n(x_k)$ and has at least n+1 roots. But $\tilde{P}_n$ and P_n are deg ≤ n.
By contradiction, Hence $\tilde{P}_n = P_n$

Adding more interpolating nodes

Let P_{n-1} be the Lagrange interpolating polynomial of f(x) with n nodes $x_0, x_1,...,x_{n-1}$. Suppose we get one more data point (x_n, f(x_n)).
$P_{n-1}(x)$ = f(x_0)L_0(x) + ... + f(x_{n-1})L_{n-1}(x) where $L_k(x) = \prod_{j=0, j\neq k}^{n-1} \frac{x-x_j}{x_k-x_j}$
Naive way: Recalculate Lagrange basis polynomial again
$P_n(x)$ = f(x_0)L_0(x) + ... + f(x_n)L_n(x) where $L_k(x) = \prod_{j=0, j\neq k}^{n} \frac{x-x_j}{x_k-x_j}$

Let $Q_n(x) = P_n(x) - P_{n-1}(x)$. $Q_n(x)$ is the unique interpolating polynomial that interpolates (x_0, 0), (x_1, 0), ..., (x_{n-1}, 0), (x_n, f(x_n - $P_{n-1}(x_n)$))
$Q_n(x)$ = f[$x_0, x_1, ..., x_n$](x - x_0)(x - x_1)...(x - x_{n-1}) where (Proof in l13)
n^{th} divided diff of f = f[$x_0, x_1, ..., x_n$] = $\sum_{k=0}^{n} f(x_k) \prod_{j=0, j\neq k}^{n} \frac{1}{x_k-x_j}$
So $P_n(x) = P_{n-1}(x) + Q_n(x) = P_{n-1}(x)$ + f[$x_0, x_1, ..., x_n$](x - x_0)(x - x_1)...(x - x_{n-1})
= ... = $P_0(x)$ + f[x_0, x_1](x - x_0) + ... + f[$x_0, ..., x_n$](x - x_0)(x - x_1)...(x - x_{n-1})

Newton's Polynomial (Easier to add more nodes)

$P_n(x) = \sum_{k=0}^{n} f[x_0,...,x_k](x-x_0)\ (x-x_1)...(x-x_{k-1})$ =
$P_0(x)$ + f[x_0, x_1](x - x_0) + ... + f[$x_0, ..., x_n$](x - x_0)(x - x_1)...(x - x_{n-1})
And n^{th} divided diff of f = f[$x_0, x_1, ..., x_n$] =
$\sum_{k=0}^{n} f(x_k) \prod_{j=0, j\neq k}^{n} \frac{1}{x_k-x_j}$. where f[$x_0$] = $P_0(x)$ = f(x_0).
Note. order of nodes don't matter. i.e. f[x_0, x_1, x_2] = f[x_1, x_2, x_0] (proof in t8)

Let $x_0, x_1, ..., x_n$ be n+1 distinct real nums. Then f[$x_0, x_1, ..., x_n$] = $\frac{f[x_1,x_2,...,x_n]-f[x_0,x_1,...,x_{n-1}]}{x_n-x_0}$. (Proof in l14)
Compute f[x_0], f[x_1]...f[x_n] first, then f[x_0, x_1] = $\frac{f[x_1]-f[x_0]}{x_1-x_0}$...
O(n^2) for computing n^{th} divided diff. Num of entries = n + (n-1) + ... + 1 = $\frac{n(n+1)}{2}$. So total need n(n+1) subtraction and $\frac{n(n+1)}{2}$ divisions

Divided Diff & Computation

for k = 0,1,...,n do {f[x_k] ← f(x_k)}
for j = 1,...,n do {
  for k = 0,...,n-j do {f[$x_k,...,x_{k+j}$] ← (f[$x_{k+1},...,x_{k+j}$] – f[$x_k,...,x_{k+j-1}$])/(x_{k+j} – x_k)}
} Result: all f[$x_j,...,x_k$] for 0 ≤ j ≤ k ≤ n

To compute $P_n(x)$ for some x, use Horner's method. O(n)
P ← f[$x_0, x_1, ..., x_n$]
for k = n-1,...,0 do {P ← f[$x_0, x_1, ..., x_k$] + (x-x_k)P}

Error of interpolation

Runge fn: f(x) = $\frac{1}{1+25x^2}$, x ∈ [-1, 1]
Runge's phenomenon: Wider oscillation at ends near interpolation interval (worse interpolation at ends). ↑ num of nodes only worsen approximation at ends of interval.
Error of interpolation, |f(x) - $P_n(x)$| : error is 0 on all nodes, error has peak btw every pair of adjacent nodes, peaks closer to 1st and last nodes are higher than nodes in the middle
$g_n(x)$ = |(x-x_0)(x-x_1)...(x-x_n)| = $|\prod_{k=0}^{n}(x-x_k)|$ has similar pattern to error |f(x) - $P_n(x)$| w equally-spaced nodes

$P_n^{(n+1)}$ = 0 as P_n only deg ≤ n.
$\frac{d^{n+1}}{dx^{n+1}}[\lambda$(x-$x_0$)(x-$x_1$)...(x-$x_n$)] = λ(n+1)! since any x w deg < n+1, after differentiating n+1 times would become 0

Let $x_0 < x_1 <...< x_n$ be n+1 distinct pts on [a, b]. If f ∈ C^{n+1}([a,b]), i.e. all derivatives f, $f^{(1)}$,..., $f^{(n+1)}$ are cts in [a, b], and P_n is the interpolating polynomial of f w deg ≤ n at $x_0, x_1, ..., x_n$. Then ∀ x ∈ ℝ, there is a ξ ∈ ℝ depends on $x_0, x_1, ..., x_n$, x, and ξ lies btw the min and max of {$x_0, x_1, ..., x_n$, x}, so that f(x) - $P_n(x)$ = $\frac{f^{(n+1)}(\xi)}{(n+1)!}$(x-$x_0$)(x-$x_1$)...(x-$x_n$) = $\frac{f^{(n+1)}(\xi)}{(n+1)!}\frac{1}{2^n}T_{n+1}(x)$
Proof. For x ∈ {$x_0, x_1, ..., x_n$}, LHS = f(x) - $P_n(x)$ = f(x) - f(x) = 0 = RHS
For x ∉ {$x_0, x_1, ..., x_n$}. Let h(x) = f(x) - $P_n(x)$ - λ(x-x_0)(x-x_1)...(x-x_n).
For h(x) = 0, now prove there exist ξ s.t. $\lambda = \frac{f^{(n+1)}(\xi)}{(n+1)!}$
Since h(x_0) = h(x_1) = ... = h(x_n) = h(x) = 0, by IVT, ∃ min(x, x_0) < y_0 < ... < y_n < max(x, x_n), s.t. $h^{(1)}(y_i)$ = 0, ∀ i = 0,1,...,n
Continue applying IVT until $h^{(n+1)}(\xi)$ = 0. Also, $h^{(n+1)}(\xi)$ = $f^{(n+1)}(\xi)$ - $P_n^{(n+1)}(\xi)$ - $\frac{d^{n+1}}{dx^{n+1}}[\lambda$(x-$x_0$)(x-$x_1$)...(x-$x_n$)]$|_{x=\xi}$ = $f^{(n+1)}(\xi)$ - λ(n+1)!. So $\lambda = \frac{f^{(n+1)}(\xi)}{(n+1)!}$

Chebyshev Interpolation

Chebyshev nodes: $x_k = \cos\left(\frac{(k+1/2)\pi}{n+1}\right)$, k = 0,1,..., n in [-1, 1]
Using Chebyshev nodes means max value of $g_n(x)$ is the smallest, with min = $\frac{1}{2^n}$. Min is achieved by $|\prod_{k=0}^{n}(x-x_k)| = \frac{1}{2^n}T_{n+1}(x)$, where $T_{n+1}(x)$ denotes the deg n+1 Chebyshev polynomial = cos((n+1)arccos x) (Proof in L15)
So now $g_n(x)$ = $|\frac{1}{2^n}$cos((n+1)arccos x)| ≤ $\frac{1}{2^n}$. g_n → 0 as n → ∞
For [a,b]: $x_k = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{(k+1/2)\pi}{n+1}\right)$, k = 0,1,..., n
For [a,b]: $|\prod_{k=0}^{n}(x-x_k)| \leq \frac{((b-a)/2)^{n+1}}{2^n}$

Now, all peaks have same height (error is more evenly distributed). Difference btw nodes is smaller near bounderies
By using Chebyshev nodes to interpolate w divided diff, this polynomial = Chebyshev interpolating polynomial
Approximation using Chebyshev nodes is worse ard center, but oscillation near ends much milder. As n increase, Chebyshev interpolating polynomial will converge to f(x)

Linear Least Square Problem

If use interpolation, deg of polynomial is higher and data points might contain errors.
Inconsistent sys: SLE w no solution, typically m > n
So find $\bar{x}$ s.t. A$\bar{x}$ is closest to b, A$\bar{x}$ lies on plane Ax
b - A$\bar{x}$ is perpendicular to plane.
Length = Euclidean norm = $||x||_2 = \sqrt{x_1^2 + x_2^2 + ... + x_n^2}$

Dot product of 2 n-dimensional col vectors = u·v = u^Tv = $u_1v_1 + u_2v_2 + ... + u_nv_n$
If u^Tv = 0, then u and v are perpendicular/orthogonal to each other. u⊥v
Note $x^Tx = ||x||^2$. Normal equations: $A^TA\bar{x} = A^Tb$
So, (b-A$\bar{x}$) ⊥ {Ax|x∈ $\mathbb{R}^n$}. $(Ax)^T$(b-A$\bar{x}$) = 0. x^TA^T(b-A$\bar{x}$) = 0.Since x ≠ 0, A^T(b-A$\bar{x}$) = 0. $A^TA\bar{x} = A^Tb$. $\bar{x}$ is L.S sol to Ax = b, which minimizes Euclidean norm r = b - Ax
If r = 0, then $\bar{x}$ is the sol to Ax = b.

If need to use $A_{m\times n}$ (full col rank = cols all LI) for multiple inconsistent sys, can use Cholesky factorization for $A^TA = R^TR$, where R is an upper triangular matrix

A^TA is symmetric positive-definite. For all x ≠ 0, $x^TA^TAx = (Ax)^T(Ax) = y^TY$ = $||y||^2$ > 0
Forward sub: $R^Ty = A^Tb$. Backward sub: R$\bar{x}$ = y

QR factorization

A^TA not numerically stable as could have rounding errors.
So, use (reduced) QR factorization: A = QR, where Q is an orthogonal matrix ($Q^T = Q^{-1}$), R is upper triangular. Then $A^TAx = A^Tb$, $(QR)^T(QR)x = (QR)^Tb$, $R^TRx = R^TQ^Tb$, Rx = Q^Tb if R^T is invertible.
So given $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_n \in \mathbb{R}^m$ (m ≥ n), need to find

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} = \begin{pmatrix} q_{11} & q_{12} & \cdots & q_{1n} \\ q_{21} & q_{22} & \cdots & q_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{m1} & q_{m2} & \cdots & q_{mn} \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ 0 & r_{22} & \cdots & r_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & r_{nn} \end{pmatrix}$$

	$\mathbf{q}_1, \mathbf{q}_2, ..., \mathbf{q}_n \in \mathbb{R}^m$ s.t. they are unit vectors (i.e. $\|\|\mathbf{q}_i\|\| = 1$) and $\mathbf{q}_i$, $\mathbf{q}_j$ are orthogonal to each other when i ≠ j $r_{11}, r_{12}, r_{22}, ..., r_{nn}$	$(\boldsymbol{a}_1\quad \boldsymbol{a}_2\quad \cdots\quad \boldsymbol{a}_n) = (\boldsymbol{q}_1\quad \boldsymbol{q}_2\quad \cdots\quad \boldsymbol{q}_n)\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix} =$ $(r_{11}\boldsymbol{q}_1\quad r_{12}\boldsymbol{q}_1 + r_{22}\boldsymbol{q}_2\quad \cdots\quad r_{1n}\boldsymbol{q}_1 + \ldots + r_{nn}\boldsymbol{q}_n)$

Gram-Schmidt orthogo-nalization	$\mathbf{y}_1 = \mathbf{a}_1$, $\mathbf{q}_1 = \frac{y_1}{r_{11}}$
	$\mathbf{y}_2 = \mathbf{a}_2 - r_{12}\mathbf{q}_1$, $\mathbf{q}_2 = \frac{y_2}{r_{22}}$
	$\mathbf{y}_j = \mathbf{a}_j - r_{1j}\mathbf{q}_1 - r_{2j}\mathbf{q}_2 - \ldots - r_{j-1,j}\mathbf{q}_{j-1}$, $\mathbf{q}_j = \frac{y_j}{r_{jj}}$
	$r_{jj} = \|\|y_j\|\|$, $r_{ij} = \mathbf{q}_i^{\mathrm{T}}\mathbf{a}_j$
		for j = 1,2,...,n do
		  $y \leftarrow \mathbf{a}_j$
		  for i = 1,2,..., j-1 do
		    $r_{ij} \leftarrow \mathbf{q}_i^{\mathrm{T}}\mathbf{a}_j$, $\mathbf{y} \leftarrow \mathbf{y} - r_{ij}\mathbf{q}_i$
		$r_{jj} \leftarrow \|\|\mathbf{y}\|\|$, $\mathbf{q}_j \leftarrow \mathbf{y}/r_{jj}$

	Reduced QR factorization: m x n = (m x n) * (n x n) $(\boldsymbol{a}_1\quad \boldsymbol{a}_2\quad \cdots\quad \boldsymbol{a}_n) = (\boldsymbol{q}_1\quad \boldsymbol{q}_2\quad \cdots\quad \boldsymbol{q}_n)\begin{pmatrix} r_{11} & r_{12} & \cdots & r_{1n} \\ & r_{22} & \cdots & r_{2n} \\ & & \ddots & \vdots \\ & & & r_{nn} \end{pmatrix}$	Full QR factorization: m x n = (m x m) * (m x n) $(\boldsymbol{a}_1\quad \boldsymbol{a}_2\quad \cdots\quad \boldsymbol{a}_n) = (\boldsymbol{q}_1\quad \cdots\quad \boldsymbol{q}_n\ \boldsymbol{q}_{n+1}\quad \cdots\quad \boldsymbol{q}_m) * \begin{pmatrix} r_{11} & \cdots & r_{1n} \\ & \ddots & \vdots \\ & & r_{nn} \\ 0 & \cdots & 0 \\ \vdots & & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$
QR steps	Compute reduced QR factorization of A. Compute Q^Tb. Solve $Rx = Q^Tb$ using backward sub	

Taylor's Thrm	Suppose $f:\mathbb{R} \to \mathbb{R}$ is n+1 times differentiable on some open interval w the n^{th} derivative $f^{(n)}$ cts on [a, x]. Then ∃ c ∈ [a,x] s.t. $f(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \ldots + \frac{1}{n!}f^{(n)}(a)(x-a)^n + \frac{1}{(n+1)!}f^{(n+1)}(c)(x-a)^{n+1}$ = P(x) + approximation error The n^{th} order Taylor polynomial for f at a is $P(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2 + \ldots + \frac{1}{n!}f^{(n)}(a)(x-a)^n$, w approximation error = $\frac{1}{(n+1)!}f^{(n+1)}(c)(x-a)^{n+1}$	
2-point forward-diff formula	If f is twice continuously differentiable, then by Taylor's thrm, let x = x+h, a = x, then $f(x) = f(a) + f'(a)(x-a) + \frac{1}{2!}f''(a)(x-a)^2$ becomes $f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2f''(a)$ OR $f'(x) \approx \frac{f(x+h)-f(x)}{h}$ with error $\frac{h}{2}f''(c)$ where c ∈ [x,x+h]	Error $\frac{h}{2}f''(c)$ is O(h) (proof in l18), i.e. if h decr by $\frac{1}{2}$, error also decr by $\frac{1}{2}$. Since error is O(h), 2-points forward-diff formula is a 1st-order method If error is $O(h^k)$, then formula is a k-order approximation
2nd order finite diff formula	If f is 3 times cts diff-tiable, then $f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(c_1)$ & $f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(c_2)$ where $x-h < c_2 < x < c_1 < x+h$ So $f(x+h) - f(x-h) = 2hf'(x) + \frac{1}{6}h^3f'''(c_1) + \frac{1}{6}h^3f'''(c_2)$, then $f'(x) = \frac{f(x+h)-f(x-h)}{2h} - [\frac{1}{12}h^2f'''(c_1) + \frac{1}{12}h^2f'''(c_2)]$ = approx - error	
Genera-lized IVT	Let f be a cts fn on interval [a,b]. Let $x_1, \ldots, x_n$ be points in [a,b] and $a_1, \ldots, a_n > 0$. Then ∃ c ∈ [a,b] s.t. $(a_1 + \ldots + a_n)f(c) = a_1f(x_1) + \ldots + a_nf(x_n)$	Proof. Let $f(x_i)$ = min and $f(x_j)$ = max of n fn values $a_1f(x_i) + \ldots + a_nf(x_i) \le a_1f(x_1) + \ldots + a_nf(x_n) \le a_1f(x_j) + \ldots + a_nf(x_j)$. $f(x_i) \le \frac{a_1f(x_1) + \ldots + a_nf(x_n)}{a_1 + \ldots + a_n} \le f(x_j)$ By IVT, ∃ c ∈ $[x_i,x_j]$ s.t. $f(c) = \frac{a_1f(x_1) + \ldots + a_nf(x_n)}{a_1 + \ldots + a_n}$ OR $(a_1 + \ldots + a_n)f(c) = a_1f(x_1) + \ldots + a_nf(x_n)$
3-point centered-diff formula	$f'(x) = \frac{f(x+h)-f(x-h)}{2h} - [\frac{1}{12}h^2f'''(c_1) + \frac{1}{12}h^2f'''(c_2)]$. Using Generalized IVT, $\frac{1}{12}h^2f'''(c_1) + \frac{1}{12}h^2f'''(c_2) = a_1g(c_1) + a_2g(c_2) = (a_1+a_2)g(c) = \frac{1}{6}h^2f'''(c)$, where x-h < c < x+h So, $f'(x) \approx \frac{f(x+h)-f(x-h)}{2h}$ w error $\frac{1}{6}h^2f'''(c)$, where x-h < c < x+h	Since error is $\frac{1}{6}h^2f'''(c) = O(h^2)$, approx is better. Generally, higher order appox formula more accurate
Approx formula for higher derivatives	For f''(x), $f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(c_1)$ & $f(x-h) = f(x) - hf'(x) + \frac{h^2}{2}f''(x) - \frac{h^3}{6}f'''(x) + \frac{h^4}{24}f^{(4)}(c_2)$ where $x-h < c_2 < x < c_1 < x+h$ $f(x+h) + f(x-h) = 2f(x) + h^2f''(x) + h^4f^{(4)}(c_1) + h^4f^{(4)}(c_2)$ 3-point centered diff formula for $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$ w error $\frac{h^2}{12}f^{(4)}(c)$ where x-h < c < x+h	
Rounding error	When 2 nums nearly equal, loss of sig digits due to computer rounding error. Suppose $\hat{f}(x+h)$, $\hat{f}(x-h)$ are floating-point version of f(x+h), f(x-h), i.e. $f(x+h) = \hat{f}(x+h) + \epsilon_1$, $f(x-h) = \hat{f}(x-h) + \epsilon_2$, for some machine rounding error ϵ_1, ϵ_2. Then error in approx for 3-point formula = $\|f'(x) - \hat{f}'(x)\| = \left\|\frac{f(x+h)-f(x-h)}{2h} - \frac{h^2}{6}f'''(c) - \frac{\hat{f}(x+h)-\hat{f}(x-h)}{2h}\right\| = \left\|\frac{\epsilon_1-\epsilon_2}{2h} - \frac{h^2}{6}f'''(c)\right\| \le \frac{\|\epsilon_1\|+\|\epsilon_2\|}{2h} + \frac{h^2}{6}\|f'''(c)\| \le \frac{\epsilon}{h} + \frac{h^2M}{6}$, where $\epsilon_1, \epsilon_2 < \epsilon > 0$ and $\|f'''(c)\| \le M$. So $\|f'(x) - \hat{f}'(x)\| \le \frac{\epsilon}{h} + \frac{h^2M}{6}$. Smallest error at $h = \sqrt[3]{\frac{3\epsilon}{M}}$	
Extra-polation	order n formula for approximating Q: $Q = F_n(h) + K(h)h^n$ For 3-point centered formula, $F_2(h) = \frac{f(x+h)-f(x-h)}{2h}$, $K(h) = \frac{f'''(c)}{6}$. If f''' is cts and h is not large, then values of $-\frac{f'''(c)}{6}$ should roughly be constant in a small interval containing c $Q - F_n(h/2) = K(h/2)(h/2)^n = \frac{1}{2^n}K(h/2)h^n \approx \frac{1}{2^n}K(h)h^n = \frac{1}{2^n}(Q - F_n(h))$. Then $Q \approx \frac{2^nF_n(h/2)-F_n(h)}{2^n-1}$: (Richardson) extrapolation formula for $F_n(h)$ Using Taylor expansion at point 0, K(h) = K(0) + O(h). $Q = F_n(h) + K(h)h^n = F_n(h) + (K(0) + O(h))h^n = F_n(h) + Bh^n + O(h^{n+1})$, where B = K(0) Then $Q = F_n(h/2) + B(h/2)^n + O(h^{n+1})$, implying $\frac{2^nF_n(h/2)-F_n(h)}{2^n-1} = \ldots = Q + O(h^{n+1})$. Thus $Q = \frac{2^nF_n(h/2)-F_n(h)}{2^n-1} + O(h^{n+1}) := F_{n+1}(h) + O(h^{n+1})$ Using extrapolation, appox for Q is of higher accuracy as $F_{n+1}(h)$ is at least an order n+1 formula, compared to original $F_n(h)$	
E.g.	Using $f'(x) = \frac{f(x+h)-f(x-h)}{2h} - \frac{h^2}{6}f'''(c)$. $\frac{2^2F_2(h/2)-F_2(h)}{2^2-1} = \ldots = \frac{f(x-h)-8f(x-h/2)+8f(x+h/2)-f(x+h)}{6h}$ = F(h), a five-point centered-diff formula By observation, since F(h) = F(-h), error term can be even powers of h only. Since original order is 2, 2+1 = 3. New order ≥ 3, thus must be 4	

Newton-Cotes approach	Use definite integral of the interpolating polynomial of f to approximate $\int_a^b f(x)\,dx$ Trapezoid Rule: Use Lagrange interpolating polynomial. Let $y_0 = f(x_0)$, $y_1 = f(x_1)$ $f(x) = [y_0\frac{x-x_1}{x_0-x_1} + y_1\frac{x-x_0}{x_1-x_0}] + \frac{(x-x_0)(x-x_1)}{2!}f''(c_x) := P_1(x) + E(x)$ for some c_x depending on x $\int_{x_0}^{x_1} f(x)\,dx = \ldots = \frac{h}{2}(y_0 + y_1) - \frac{h^3}{12}f''(c)$:= approx + error where $h = x_1 - x_0$ and c ∈ $[x_0, x_1]$	Proof in lect 20. Uses MVT for integrals: Let f be cts fn on interval [a,b] and let g be an integrable fn that does not change sign on [a,b]. Then ∃ c ∈ [a,b] s.t. $\int_a^b f(x)g(x)\,dx = f(c)\int_a^b g(x)\,dx$
	Simpson's Rule: Use Lagrange interpolating polynomial. $y_0 = f(x_0)$, $y_1 = f(x_1)$, $y_2 = f(x_2)$ $f(x) = [y_0\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} + y_1\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)} + y_2\frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}] + \frac{(x-x_0)(x-x_1)(x-x_2)}{3!}f'''(c_x) := P_2(x) + E(x)$ for some c_x depending on x.	Assume nodes evely spaced. Proof in lect 20. Deg of precision of a numerical integration mtd is the greatest int k for which all deg k or less polynomials are integrated exactly by the mtd. Deg of precision of Trapezoid rule: 1

	$\int_{x_0}^{x_2} f(x)\,dx = \ldots = \frac{h}{3}(y_0 + 4y_1 + y_2) - \frac{h^5}{90}f^{(4)}(c)$:= approx + error where $h = x_1 - x_0 = x_2 - x_1$ and $c \in [x_0, x_2]$	Deg of precision of Simpson Rule: 3. Proof in tut?
Composite Newton-Cotes Formulas	Trapezoid and Simpson's Rule only operating on single interval Composite numerical integration: Divide interval into several subintervals Composite Trapezoid Rule: $\int_a^b f(x)\,dx = \frac{h}{2}(y_0 + 2\sum_{i=1}^{m-1} y_i + y_m) - \frac{(b-a)h^2}{12}f''(c)$ where $h = (b-a)/m$ and $c \in [a,b]$. Order of comp trapezoid rule = 2, $O(h^2)$. Proof in lect 20.	Composite Simpson's Rule: $\int_a^b f(x)\,dx = \frac{h}{3}(y_0 + 4\sum_{i=1}^{m} y_{2i-1} + 2\sum_{i=1}^{m-1} y_{2i} + y_{2m}) - \frac{(b-a)h^4}{180}f^{(4)}(c)$:= approx + error where $h = (b-a)/(2m)$ and $c \in [a,b]$ Order of comp Simpson's rule: 4. $O(h^4)$
Open Newton-Cotes Mtd	Use if fn not valid on endpoints. Applicable for fn whose f'' is cts on [a,b] Midpoint Rule: $\int_{x_0}^{x_1} f(x)\,dx = hf(w) + \frac{h^3}{24}f''(c)$ where $h = (x_1 - x_0)$, w is midpoint := $x_0 + h/2$ and $c \in [x_0,x_1]$. Proof in lect 21.	Composite Midpoint Rule: $\int_a^b f(x)\,dx = h\sum_{i=1}^{m} f(w_i) + \frac{(b-a)h^2}{24}f''(c)$. $O(h^2)$
Romberg integration	Composite Trapezoid Rule: $\int_a^b f(x)\,dx = \frac{h}{2}(y_0 + 2\sum_{i=1}^{m-1} y_i + y_m) - \frac{(b-a)h^2}{12}f''(c)$ Using Richardson extrapolation would give us a new rule of at least 3rd order For infinitely differentiable fn f, $\int_a^b f(x)\,dx = \frac{h}{2}(y_0 + 2\sum_{i=1}^{m-1} y_i + y_m) + c_2h^2 + c_4h^4 + c_6h^6 + \ldots$, where c_i depends only on higher derivatives of f at a and b and not h (e.g. $c_2 = \frac{1}{12}(f'(a) - f'(b))$) Let $Q := \int_a^b f(x)\,dx$, $F_2(h) := = \frac{h}{2}(y_0 + 2\sum_{i=1}^{m-1} y_i + y_m)$. Then $Q = F_2(h) + c_2h^2 + c_4h^4 + c_6h^6 + \ldots$	Cutting h in half and combining, $F_4(h) := \frac{2^2F_2(h/2)-F_2(h)}{2^2-1}$. $Q = F_4(h) + \tilde{c}_4h^4 + \tilde{c}_6h^6 + \ldots = F_4(h) + O(h^4)$. Proof in lect 21 Cutting h in half again and combining, $F_6(h) := \frac{2^{2(2)}F_4(h/2)-F_4(h)}{2^{2(2)}-1}$. $Q = F_6(h) + \hat{c}_6h^6 + \ldots = F_6(h) + O(h^6)$. Proof in lect 21 So $F_{2k}(h) := \frac{2^{2(k-1)}F_{2(k-1)}(h/2)-F_{2(k-1)}(h)}{2^{2(k-1)}-1}$
	diff h $\frac{2^2F_2(\frac{h}{2})-F_2(h)}{2^2-1}$ $\begin{array}{llll} F_2(h) & & & \\ F_2(h/2) & F_4(h) & & \\ F_2(h/4) & F_4(h/2) & F_6(h) & \\ F_2(h/8) & F_4(h/4) & F_6(h/2) & F_8(h) \\ \vdots & \vdots & \vdots & \vdots \quad \ddots \end{array}$	Romberg triangle: Comp. Trap. rule $\begin{array}{llll} R_{11} & & & \\ R_{21} & R_{22} & & \\ R_{31} & R_{32} & R_{33} & \\ R_{41} & R_{42} & R_{43} & R_{44} \\ \vdots & \vdots & \vdots & \vdots \quad \ddots \end{array}$ where $R_{jk} = \frac{2^{2(k-1)}R_{j,k-1}-R_{j-1,k-1}}{2^{2(k-1)}-1}$ R_{22} : 4nd order R_{33} : 6th order R_{44} : 8th order.
Not tested	Gaussian Quadrature, Legendre polynomials	