CS371/AMATH242 Final Cheat Sheet — Page 1: Pre-midterm (Lec 1–11)

1. FLOATING POINT

***Big idea**: computers store only* finitely many *numbers. Every real x gets "rounded" to the nearest representable FP number, so* every *arithmetic op can introduce a tiny error $\sim \epsilon_m$. These errors accumulate and sometimes* amplify *(see cancellation).*

Repr. $F[b,m,e]$: $\pm 0.x_1\cdots x_m \times b^{\text{exp}}$, $x_1 \neq 0$ (normalised); $F \neq \mathbb{R}$ — finite, discrete, *not* evenly spaced (gap grows with magnitude).

***F[10,3,2]**:* $\pm 0.d_1 d_2 d_3 \times 10^{\pm e_1 e_2}$, $d_1 \in \{1,\dots,9\}$. *Largest:* 0.999×10^{99}; *Smallest pos.:* 0.100×10^{-99}. *Mantissa count:* $9\cdot 10\cdot 10 = 900$; *exponents* $-99..99$: 199. *Total nonzero:* $2\cdot 900\cdot 199 = 358{,}200$ *(plus* ± 0*).*

Errors: $E_{\text{abs}} = |x-\hat{x}|$; $E_{\text{rel}} = \frac{|x-\hat{x}|}{|x|}$ $(x\neq 0)$. Use E_{rel} to compare across scales.
Machine ϵ_m : smallest $\epsilon>0$ with $\text{fl}(1+\epsilon)>1$.
Chopping: $\epsilon_m = b^{1-m}$; Rounding: $\epsilon_m = \frac{1}{2}b^{1-m}$.
Double ($m=53$): $\epsilon_m \approx 1.1\times 10^{-16}$, $x_{\max}\approx 10^{308}$.
Single ($m=24$): $\epsilon_m \approx 6\times 10^{-8}$, $x_{\max}\approx 10^{38}$.
Rounding model: $\text{fl}(x) = x(1+\eta)$, $|\eta|\le\epsilon_m$.

***Represent** $\frac{1}{7} = 0.142857\ldots$ **in** $F[10,4,2]$:*
Chop: 0.1428×10^0. *Round:* 0.1429×10^0.
$E_{rel}^{chop} = |0.142857-0.1428|/0.142857 \approx 4\times 10^{-4} < \epsilon_m = 10^{-3}$ ✓

FP add NOT associative. $a\oplus b := \text{fl}(\text{fl}(a)+\text{fl}(b))$. Order of additions matters: adding small numbers first avoids losing them.
Catastrophic cancellation: subtracting near-equal loses sig. digits — the *leading* matching digits cancel, exposing the low-order rounding noise. **Fix:** algebraic rewrite, e.g. $\sqrt{x+1}-\sqrt{x} = \frac{1}{\sqrt{x+1}+\sqrt{x}}$ (both terms positive, no cancellation).

2. CONDITIONING & STABILITY

Conditioning = PROBLEM; Stability = ALGORITHM. Ill-cond. limits any algo; unstable algo fails on well-cond. problem.

***Plain-English**:* Conditioning *asks "if my input wiggles a tiny bit, does the true answer wiggle a lot?" — intrinsic to the math, you can't fix it with a better algorithm.* Stability *asks "does my algorithm amplify rounding errors as it computes?" — you CAN fix this by choosing a smarter algorithm. Accurate result needs BOTH: well-conditioned problem* and *stable algorithm.*

Cond. of $z=f(x)$: $\kappa_A \approx |f'(x)|$, $\kappa_R \approx |\frac{xf'(x)}{f(x)}|$.
Small κ: well-cond. Large $\kappa \gg 1$: ill-cond.

$f(x)=x/(1-x)$: $\kappa_R = 1/|1-x|$. At $x=0.93$: $\kappa_R\approx 14$; at $x=5$: 0.25.

Stability of recurrence: error obeys *same* recurrence as the solution. So error growth is dictated by char. roots r_i: stable iff all $|r_i|\le 1$. Any root with $|r|>1$ causes tiny initial rounding to explode like r^n.
1st order $I_n = c-\alpha I_{n-1}$: $\varepsilon_n = (-\alpha)^n\varepsilon_0$. $|\alpha|<1$ stable.
2nd order $p_n = c_1 p_{n-1}+c_2 p_{n-2}$: char. $r^2-c_1 r-c_2=0$, $e_n = Ar_1^n + Br_2^n$. Repeated $|r|=1$: $(A+Bn)r^n$ — linear growth *(stable).*

$p_n = 2p_{n-1}-p_{n-2}$: $(r-1)^2=0$, $r=1$. Stable.

Taylor: $f(x) = \sum_{k=0}^{N}\frac{f^{(k)}(a)}{k!}(x-a)^k + R_N$; $R_N = \frac{f^{(N+1)}(\xi)}{(N+1)!}(x-a)^{N+1}$.

Common: $e^x = 1+x+\frac{x^2}{2}+\frac{x^3}{6}+\cdots$; $\sin x = x-\frac{x^3}{6}+\cdots$; $\cos x = 1-\frac{x^2}{2}+\cdots$; $\ln(1+x) = x-\frac{x^2}{2}+\cdots$

3. FINITE DIFFERENCES (Lec 5)

Forward $\frac{f(x+h)-f(x)}{h}$, trunc. $\frac{h}{2}|f''|$, $O(h)$.
Backward $\frac{f(x)-f(x-h)}{h}$, $O(h)$.
Central $\frac{f(x+h)-f(x-h)}{2h}$, err $\frac{h^2}{6}|f'''|$, $O(h^2)$ — $O(h)$ terms cancel.
2nd deriv: $f''(x) \approx \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$, $O(h^2)$.
Total error (rounding + truncation):

$$E_{\text{fwd}} \approx \frac{h}{2}|f''| + \frac{2\epsilon_m|f|}{h} \Rightarrow h_{\text{opt}} \approx \sqrt{\epsilon_m} \approx 10^{-8}$$

$$E_{\text{cen}} \approx \frac{h^2}{6}|f'''| + \frac{\epsilon_m|f|}{h} \Rightarrow h_{\text{opt}} \approx \epsilon_m^{1/3} \approx 10^{-5}$$

Smaller h NOT always better. Balance truncation $\propto h^p$ vs rounding $\propto \epsilon_m/h$.

***Why central wins**: forward uses Taylor expansion only one way, so $O(h)$ error term survives. Central subtracts fwd and bwd, so the* odd-order *terms $(h, h^3, \ldots)$ cancel by symmetry, leaving $O(h^2)$. Same cost (2 evals) but quadratically better.*

4. ROOT FINDING

***Why iterate?** Most nonlinear eqns have no closed-form root. Strategy: start from guess, apply rule that brings you closer. Trade-offs:* ***Bisection**=slow but* guaranteed*;* ***Newton**=fast (quadratic) but needs f' and good start;* ***Secant**=almost as fast, no f' needed;* ***Fixed-pt**=simple but may not converge.*

IVT: f cts on $[a,b]$, $f(a)f(b)<0 \Rightarrow \exists x^*$: $f(x^*)=0$.
Double root: $f(x^*)=0$ *and* $f'(x^*)=0$; Newton drops to *linear* (because it divides by $f'\approx 0$).
Bisection: $x_{k+1} = (a_k+b_k)/2$. Error $|e_n| \le \frac{b_0-a_0}{2^{n+1}}$.
Iters for tol: $n \ge \log_2(\frac{b_0-a_0}{\text{tol}})$.

$f=x^3+4x^2-10$, $[1,2]$, tol$\vDash 0.1$: $n\ge\log_2(10)\approx 3.32 \Rightarrow n=4$. $x_1=1.5$, $f=2.375$; $x_2=1.25$, $f=-1.797$; $x_3=1.375$, $f=0.162$; $x_4=1.3125$, $f=-0.848$. Root ≈ 1.34.

Error bound is UPPER only: actual $|x_k-x^*|$ can increase *even as interval halves (when x^* near endpoint).*

Fixed pt: $x_{n+1}=g(x_n)$, $x^*=g(x^*)$. Converges if $g([a,b])\subseteq[a,b]$, $|g'|\le\lambda<1$. Rate $\approx |g'(x^*)|$.
Error bound: $|e_n| \le \frac{\lambda^{n-1}}{1-\lambda}|x_1-x_0|$.

Newton: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$. **Geometric**: follow tangent line at x_k to its x-intercept. **Quadratic**:

$$e_{k+1} \approx -\frac{f''(\xi)}{2f'(x_k)}e_k^2$$

(digits *double* each step: $10^{-2}\to 10^{-4}\to 10^{-8}$.) Fails if $f'\approx 0$ or double root.

$f=x^2-2$, $x_0=1$: $x_1=1.5$, $x_2\approx 1.4167$, $x_3\approx 1.41421$.

Secant: $x_{k+1} = x_k - f(x_k)\frac{x_k-x_{k-1}}{f(x_k)-f(x_{k-1})}$. Order $q = \frac{1+\sqrt{5}}{2} \approx 1.618$; no deriv. Replace $f'(x_k)$ in Newton with a *finite-difference* estimate using last two iterates — cheaper per step, slightly slower convergence.

Conv. order: $\lim \frac{|e_{k+1}|}{|e_k|^q} = \mu > 0$. $q=1$ linear, $q=2$ quadratic, $q\approx 1.618$ secant.

5. LU DECOMPOSITION

***Big idea**: solving $Ax=b$ directly is $O(n^3)$, but if A is triangular it's only $O(n^2)$ (back-substitution). LU* factors *A once into two triangular matrices, then each new b only costs $O(n^2)$. Gauss elimination = LU in disguise: the multipliers you compute are the entries of L.*

LU: $A=LU$, L unit lower-tri, U upper-tri. Solve $Ly=b$ (forward), $Ux=y$ (back).
Multiplier $\ell_{ij} = a_{ij}^{(j)}/a_{jj}^{(j)}$ zeros (i,j); store in L.

Cost: factor $\frac{2}{3}n^3$; each solve n^2. **Multi-RHS:** factor once, reuse.

PA=LU (partial pivot): swap rows for max $|a_{ij}^{(j)}|$. Solve $Ly=Pb$, $Ux=y$. Always pivot.
Why pivot? A tiny pivot makes $\ell_{ij}=a_{ij}/a_{jj}$ huge, which amplifies rounding errors in the elimination. Picking the largest-magnitude pivot keeps multipliers ≤ 1.
$\det(A) = \det(P)\prod u_{ii} = (-1)^{\#\text{swaps}}\prod u_{ii}$.

$A = \begin{bmatrix}2&1&1\\4&3&3\\8&7&9\end{bmatrix}$, $b=[1,1,1]^T$.
$\ell_{21}=2$, $\ell_{31}=4$, $\ell_{32}=3$.
$L = \begin{bmatrix}1&0&0\\2&1&0\\4&3&1\end{bmatrix}$, $U = \begin{bmatrix}2&1&1\\0&1&1\\0&0&2\end{bmatrix}$.
Fwd: $y=[1,-1,0]^T$. *Back:* $x=[1,-1,0]^T$. ✓

GE pseudocode (no pivot):
```
for c=1:n-1
for r=c+1:n
m ← U(r,c)/U(c,c)
U(r,c:n) ← U(r,c:n) - m·U(c,c:n)
L(r,c) ← m
```

6. NORMS & COND. NUMBERS

Vector: $\|x\|_1 = \sum|x_i|$, $\|x\|_2 = \sqrt{\sum x_i^2}$, $\|x\|_\infty = \max|x_i|$.

$x=[3,-4,0]^T$: $\|x\|_1=7$, $\|x\|_2=5$, $\|x\|_\infty=4$.

Matrix (induced): $\|A\|_p = \max_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p}$.
$\|A\|_1 = \max_j\sum_i|a_{ij}|$ **(max col sum)**.
$\|A\|_\infty = \max_i\sum_j|a_{ij}|$ **(max row sum)**.
$\|A\|_2 = \sqrt{\lambda_{\max}(A^TA)}$ (largest sing. val).
Props: $\|Ax\|\le\|A\|\|x\|$, $\|AB\|\le\|A\|\|B\|$.

$A=\begin{bmatrix}1&2\\3&4\end{bmatrix}$: $\|A\|_1=6$, $\|A\|_\infty=7$.

Cond. number: $\kappa_p(A) = \|A\|_p\|A^{-1}\|_p \ge 1$.
$\kappa_2(A) = \sqrt{\lambda_{\max}(A^TA)/\lambda_{\min}(A^TA)}$.
$\kappa(I)=1$; near-singular $\Rightarrow \kappa\to\infty$; $\kappa(cA)=\kappa(A)$.

Meaning: $\frac{\|\Delta x\|}{\|x\|} \le \kappa(A)\frac{\|\Delta b\|}{\|b\|}$. Large $\kappa \Rightarrow$ tiny b-error can give huge x-error. Rule of thumb: if $\kappa\sim 10^k$ you lose about k digits of accuracy. ***Spot**: nearly proportional rows/cols $\Rightarrow$ $\det\approx 0 \Rightarrow \kappa$ large.*

$A=\begin{bmatrix}1&1\\1&1.01\end{bmatrix}$: $A^TA=\begin{bmatrix}2&2.01\\2.01&2.0201\end{bmatrix}$.
$\lambda^2-4.0201\lambda+10^{-4}=0 \Rightarrow \lambda_{\max}\approx 4.02$, $\lambda_{\min}\approx 2.5\times 10^{-5}$.
$\kappa_2\approx 401$.

7. ITERATIVE METHODS

***Why iterate?** LU on an $n\times n$ matrix is $O(n^3)$ and destroys sparsity (fills in zeros). For sparse problems (PDEs, graphs), iterative methods only touch nonzero entries: each iter $O(nnz)\ll O(n^3)$. You stop when residual is small enough — no need for exact answer.*

When: A large & sparse. Each iter $O(n^2)$ vs LU $O(n^3)$.
SDD: $|a_{ii}| > \sum_{j\neq i}|a_{ij}|$ $\forall i$ $\Rightarrow$ A nonsingular; Jacobi & GS converge. (Diagonal dominates $\Rightarrow$ row eqn. is mostly about one variable, so "solve for x_i using latest estimates of others" works.)

Decomp: $A=D+L+R$.
Jacobi (old vals; parallel; $B=D$):

$$x_i^{(k+1)} = \tfrac{1}{a_{ii}}\big(b_i - \sum_{j\neq i}a_{ij}x_j^{(k)}\big)$$

Vector: $\vec{x}^{\text{new}} = D^{-1}(\vec{b}-(L+R)\vec{x}^{\text{old}})$.
Gauss-Seidel (new vals; $\sim$2× faster; $B=D+L$):

$$x_i^{(k+1)} = \tfrac{1}{a_{ii}}\big(b_i - \sum_{j<i}a_{ij}x_j^{(k+1)} - \sum_{j>i}a_{ij}x_j^{(k)}\big)$$

$4x_1-x_2=7$, $-x_1+3x_2=5$, start $(0,0)$. ***Jac** It.1: $x_1=1.75$, $x_2\approx 1.667$. It.2: $2.167, 2.25$.* ***GS** It.1: $x_1=1.75\to x_2=2.25$. It.2: $2.3125, 2.4375$. True $(2.364, 2.455)$.*

General: $x^{(k+1)} = x^{(k)} + B^{-1}(b-Ax^{(k)})$.
Converges iff $\|I-B^{-1}A\|<1$.
$e^{(k+1)} = (I-B^{-1}A)e^{(k)}$.
Residual: $r=b-Au$, $Ae=r$. Stop: $\|r^{(k)}\|/\|b\|<t$. Note $e\neq r$!

PSEUDOCODE (ALL METHODS)

Bisection:
```
while (b-a)/2 > tol:
m = (a+b)/2
if f(m)==0:  return m
if f(a)*f(m)<0:  b=m else a=m
return (a+b)/2
```
Newton:
```
x = x0
for k=1:maxit:
x = x - f(x)/fp(x)
if |f(x)|<tol:  return x
```
Secant:
```
for k=1:maxit:
x2 = x1 - f(x1)*(x1-x0)/(f(x1)-f(x0))
x0 = x1; x1 = x2
if |f(x1)|<tol:  return x1
```
Fixed-point:
```
for k=1:maxit:
x_new = g(x)
if |x_new-x|<tol:  return x_new
x = x_new
```
Forward sub $Ly=b$**:**
```
for i=1:n:
y(i) = b(i)
for j=1:i-1:
y(i) = y(i)-L(i,j)*y(j)
y(i) = y(i)/L(i,i)
```
Back sub $Ux=y$**:**
```
for i=n:-1:1:
x(i) = y(i)
for j=i+1:n:
x(i) = x(i)-U(i,j)*x(j)
x(i) = x(i)/U(i,i)
```
Jacobi:
```
while ||r||/||b|| > tol:
for i=1:n:
s = 0
for j=1:n, j≠i:
s = s+A(i,j)*x_old(j)
x_new(i) = (b(i)-s)/A(i,i)
x_old = x_new
```
Gauss-Seidel: (overwrite x in place)
```
while ||r||/||b|| > tol:
for i=1:n:
s = 0
for j=1:n, j≠i:
s = s+A(i,j)*x(j)
x(i) = (b(i)-s)/A(i,i)
```

EXTRA WORKED EXAMPLES

***Fixed-pt** $x=\cos x$, $x_0=0.5$:*
$x_1=\cos(0.5)\approx 0.8776$,
$x_2\approx\cos(0.8776)\approx 0.6390$,
$x_3\approx 0.8026$, $x_4\approx 0.6948$, …
Converges to $x^*\approx 0.739$ (slow: $|g'|=|\sin x^*|\approx 0.67$).

***Newton on** $f=x^3-2$, $x_0=1$:*
$f'=3x^2$. $x_1=1-(-1)/3=1.333$.
$f(x_1)=2.370-2=0.370$; $f'=5.333$.
$x_2=1.333-0.0694=1.2639$.
$x_3\approx 1.25993$. *True:* $\sqrt[3]{2}\approx 1.25992$. *Digits doubled.*

***SDD check:** $A=\begin{bmatrix}4&-1&0\\-1&5&2\\1&-2&6\end{bmatrix}$.*
Row 1: $|4|>|-1|+|0|=1$ ✓
Row 2: $|5|>|-1|+|2|=3$ ✓
Row 3: $|6|>|1|+|-2|=3$ ✓
SDD $\Rightarrow$ Jacobi & GS converge for any $b, x^{(0)}$.

***κ_∞ via A^{-1}:** $A=\begin{bmatrix}1&2\\3&4\end{bmatrix}$, $\det=-2$.*
$A^{-1} = \frac{1}{-2}\begin{bmatrix}4&-2\\-3&1\end{bmatrix} = \begin{bmatrix}-2&1\\1.5&-0.5\end{bmatrix}$.
$\|A^{-1}\|_\infty = \max(3,2)=3$. $\|A\|_\infty = 7$.
$\kappa_\infty = 21$.

***Why SDD $\Rightarrow$ nonsingular**: assume $Ax=0$, $x\neq 0$. Let $|x_k| = \max|x_i| > 0$. Row k: $|a_{kk}||x_k| \le \sum_{j\neq k}|a_{kj}||x_j| \le (\sum_{j\neq k}|a_{kj}|)|x_k|$. Divide by $|x_k|$: contradicts SDD.* □

Order-of-convergence test: compute $r_k = \log(|e_{k+1}|/|e_k|)/\log(|e_k|/|e_{k-1}|) \to q$ (order). Useful for checking method convergence numerically.

CS371/AMATH242 Final Cheat Sheet — Page 2: Post-midterm (Lec 12–22)

8. POLYNOMIAL INTERPOLATION

***Big idea**: given data points, find a polynomial passing through all of them exactly. Three* equivalent *forms give the SAME polynomial but differ in cost/conditioning: monomial (solve linear system, ill-cond), Lagrange (no system, but redo basis if node added), Newton-divided-diff (incremental).*

Goal: distinct (x_i, f_i), $i = 0, \dots, n \Rightarrow$ unique $y_n \in \mathcal{P}_n$ with $y_n(x_i) = f_i$.

Monomial / Vandermonde:

$$y_n(x) = \sum_{j=0}^{n} a_j x^j, \quad Va = f, \; V_{ij} = x_i^j.$$

$\det V = \prod_{i<j}(x_j - x_i) \neq 0$ for distinct nodes $\Rightarrow$ unique. *Ill-cond. for large n.*

Lagrange basis:

$$L_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}, \quad L_i(x_j) = \delta_{ij}.$$

$$y_n(x) = \sum_{i=0}^{n} f_i L_i(x).$$

Pros: no solve. **Cons**: redo all basis if node added.

Error:

$$f(x) - y_n(x) = \frac{f^{(n+1)}(\xi(x))}{(n+1)!} \prod_{i=0}^{n} (x - x_i).$$

Two factors: (1) smoothness of f via $f^{(n+1)}$, (2) node placement via $\prod(x - x_i)$. **Runge phenomenon**: with equispaced nodes, increasing n can make error *worse* near endpoints (wild oscillations). Fix: use Chebyshev nodes, or switch to piecewise splines.

Degree detection: p has deg d iff d-th finite diffs are constant $\neq 0$ and $(d+1)$-th are zero. Build diff table!

Assn3 Q1 style: *data* $(-2,-5), (-1,1), (0,1), (1,1), (2,7), (3,25)$. *1st diffs:* $6, 0, 0, 6, 18$; *2nd:* $-6, 0, 6, 12$; *3rd:* $6, 6, 6$ *(const)* $\Rightarrow \deg = 3$.

Lagrange 3 pts $(0,1), (1,3), (2,2)$:
$L_0 = \frac{(x-1)(x-2)}{2}$, $L_1 = \frac{x(x-2)}{-1}$, $L_2 = \frac{x(x-1)}{2}$.
$y = 1 \cdot L_0 + 3 \cdot L_1 + 2 \cdot L_2 = -\frac{3}{2}x^2 + \frac{7}{2}x + 1$.

***Cubic w/ Hermite-style constraints** (Assn3 Q2): $p(0), p'(0), p(1), p''(1)$ given. Write $p = a_0 + a_1 x + a_2 x^2 + a_3 x^3$; plug each condition into 4×4 linear system for (a_0, a_1, a_2, a_3). Unique iff $\det \neq 0$.*

9. HERMITE & CUBIC SPLINES

Hermite cubic on $[x_i, x_{i+1}]$, matches $y_i, y_{i+1}, s_i, s_{i+1}$:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

$$a_i = y_i, \quad b_i = s_i,$$

$$c_i = \frac{3y_i' - 2s_i - s_{i+1}}{\Delta x_i}, \quad d_i = \frac{s_{i+1} + s_i - 2y_i'}{\Delta x_i^2},$$

where $y_i' = (y_{i+1} - y_i)/\Delta x_i$, $\Delta x_i = x_{i+1} - x_i$.

***Why splines?** One high-degree polynomial through many points oscillates wildly (Runge). A* cubic spline *uses a separate cubic on each sub-interval, glued smoothly (S, S', S'' continuous). Each piece is low-degree (no oscillation), and smoothness is enforced globally. Best of both worlds: local control + smooth curve.*

Cubic spline: piecewise cubic with S, S', S'' continuous at interior knots.

Slope system (interior $i = 2, \dots, n-1$):

$$\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i) s_i + \Delta x_{i-1} s_{i+1}$$

$$= 3(\Delta x_i y_{i-1}' + \Delta x_{i-1} y_i')$$

Tridiagonal in $s_i \Rightarrow O(n)$ solve.

Boundary conditions (need 2):

- **Clamped:** $s_1 = s_1^*$, $s_n = s_n^*$ (given).
- **Natural/Free:** $S''(x_1) = S''(x_n) = 0$, giving $s_1 + \frac{s_2}{2} = \frac{3}{2}y_1'$ and $\frac{s_{n-1}}{2} + s_n = \frac{3}{2}y_{n-1}'$.
- **Periodic:** $S'(x_1) = S'(x_n)$, $S''(x_1) = S''(x_n)$.
- **Not-a-knot:** S''' cts at x_2 and x_{n-1}.

Counts: $n-1$ pieces, $4(n-1)$ coeffs. Conditions: $2(n-1)$ interp, $n-2$ for S', $n-2$ for S'', $+2$ BC $= 4(n-1)$. ✓

Exam recipe: (1) compute Δx_i and y_i'; (2) build tridiag $T\vec{s} = \vec{r}$ for slopes using interior eqns + BC; (3) solve for s_i; (4) plug into Hermite form for each piece.

4 pts $(0,1), (2,1), (3,3), (4,-1)$, ***clamped*** $s_1 = 1, s_4 = -1$.
$\Delta x_1 = 2, \Delta x_2 = 1, \Delta x_3 = 1$.
$y_1' = 0, y_2' = 2, y_3' = -4$.
Interior eqn $i=2$: $\Delta x_2 s_1 + 2(\Delta x_1 + \Delta x_2)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y_1' + \Delta x_1 y_2')$
$\Rightarrow s_1 + 6s_2 + 2s_3 = 12$.
Interior eqn $i=3$: $s_2 + 4s_3 + s_4 = 3(1 \cdot 2 + 1 \cdot (-4)) = -6$
$\Rightarrow s_2 + 4s_3 = -5$.
Sub $s_1 = 1$ into first: $6s_2 + 2s_3 = 11$. *Solve 2×2:* $s_2 = \frac{27}{11}$, $s_3 = -\frac{41}{22}$ *(approx).*

***Natural BC version**: replace row 1 by $s_1 + \frac{s_2}{2} = \frac{3}{2}y_1'$ and last row by $\frac{s_{n-1}}{2} + s_n = \frac{3}{2}y_{n-1}'$.*

***Verify S'' continuity** (Assn3 Q3 style): write each piece, compute $S_i''(x_{i+1})$ and $S_{i+1}''(x_{i+1})$, equate. Always check this to confirm "is a cubic spline".*

10. NUMERICAL INTEGRATION

***Big idea**: approximate $\int f$ by evaluating f at a few nodes and taking a* weighted sum. *Different rules differ only in choice of nodes/weights. Derivation: fit a polynomial through the nodes, integrate* that *polynomial exactly — gives Midpoint (const), Trap (linear), Simpson (parabola).*

Newton-Cotes: $\int_a^b f\,dx \approx \sum w_i f(x_i)$; integrate interpolant.
Degree of precision m : exact for all $p \in \mathcal{P}_m$. Rule: test $1, x, x^2, \dots$ until fails.
Surprise: Simpson fits a *parabola* (deg 2) so you'd expect DoP$=2$, but by symmetry it's exact for x^3 too $\Rightarrow$ DoP$=3$. "Free lunch" due to the midpoint being the center.
Single-panel rules on $[a, b]$:

$$\text{Mid: } (b-a)f\Big(\tfrac{a+b}{2}\Big), \; E_M = \frac{(b-a)^3}{24} f''(\xi), \; \deg 1$$

$$\text{Trap: } \frac{b-a}{2}[f(a)+f(b)], \; E_T = -\frac{(b-a)^3}{12} f''(\xi), \; \deg 1$$

$$\text{Simp: } \frac{b-a}{6}\Big[f(a) + 4f\Big(\tfrac{a+b}{2}\Big) + f(b)\Big],$$

$$E_S = -\frac{(b-a)^5}{2880} f^{(4)}(\xi), \; \deg 3$$

Error bound recipe (Assn3 Q5): (1) find $|f^{(k)}|$ max on $[a,b]$ (take deriv, check endpoints/critical pts); (2) plug into E formula; (3) compare with $|I_{\text{exact}} - \hat{I}|$.

***Method of undetermined coefficients:** find c_i in $\int_0^2 f\,dx \approx c_0 f(0) + c_1 f(1) + c_2 f(2)$ exact for deg ≤ 2. Test $f = 1$: $c_0 + c_1 + c_2 = 2$. $f = x$: $c_1 + 2c_2 = 2$. $f = x^2$: $c_1 + 4c_2 = \frac{8}{3}$. Solve: $c_0 = c_2 = \frac{1}{3}$, $c_1 = \frac{4}{3}$ (Simpson!).*

***Reverse-engineer:** Given Trap$=4$ and Simp$=2$ on $[0,2]$. Trap: $(2/2)[f(0)+f(2)] = 4 \Rightarrow f(0)+f(2) = 4$. Simp: $(2/6)[f(0) + 4f(1) + f(2)] = 2 \Rightarrow f(0) + 4f(1) + f(2) = 6 \Rightarrow f(1) = \frac{1}{2}$.*

***Why composite?** Single-panel error scales as $(b-a)^3$ or $(b-a)^5$. Huge interval $\Rightarrow$ huge error. Fix: split $[a,b]$ into n subintervals of size h, apply the rule on each, and sum. Error per subinterval is tiny, and they add up to $O(h^2)$ or $O(h^4)$ globally.*

Composite rules with $h = (b-a)/n$, $x_i = a + ih$:
Comp. Trap: $\frac{h}{2}\big[f_0 + 2\sum_{i=1}^{n-1} f_i + f_n\big]$,
$E = -\frac{(b-a)}{12} h^2 f''(\xi)$, $O(h^2)$.
Comp. Simp (n even): $\frac{h}{3}\big[f_0 + 4\sum_{\text{odd } i} f_i + 2\sum_{\text{even } i} f_i + f_n\big]$,
$E = -\frac{(b-a)}{180} h^4 f^{(4)}(\xi)$, $O(h^4)$.
Comp. Mid: $2h \sum_{i=0}^{n/2} f(x_{2i})$ (odd indices), $O(h^2)$.

Required n for tol τ: set $|E| \leq \tau$, solve for h (hence n).
Trap: $h \leq \sqrt{\frac{12\tau}{(b-a)|f''|_{\max}}}$. **Simp:** $h^4 \leq \frac{180\tau}{(b-a)|f^{(4)}|_{\max}}$.

$\int_0^\pi \sin x\,dx$, *comp. Trap, tol* $= 2 \times 10^{-5}$: $|\sin''| \leq 1$, $h^2 \leq \frac{12 \cdot 2 \times 10^{-5}}{\pi} \Rightarrow h \lesssim 8.7 \times 10^{-3} \Rightarrow n \geq 360$.

11. GAUSSIAN QUADRATURE

***Why Gauss beats Newton-Cotes:** NC fixes nodes equally-spaced and only chooses n weights (n unknowns $\Rightarrow$ DoP $n-1$). Gauss chooses BOTH n nodes and n weights ($2n$ unknowns $\Rightarrow$ DoP $2n-1$). Twice the accuracy for the same number of f-evals.*

Idea: optimize both nodes *and* weights $\Rightarrow$ n-pt Gauss-Legendre exact for $\mathcal{P}_{2n-1}$.
2-pt on $[-1, 1]$:

$$\int_{-1}^{1} f(x)\,dx \approx f\Big(-\tfrac{1}{\sqrt{3}}\Big) + f\Big(\tfrac{1}{\sqrt{3}}\Big), \; \deg 3.$$

Change of interval $[a,b] \to [-1,1]$: $x = \frac{b-a}{2}t + \frac{a+b}{2}$, $dx = \frac{b-a}{2}dt$:

$$\int_a^b f(x)dx = \frac{b-a}{2}\int_{-1}^{1} f\Big(\tfrac{b-a}{2}t + \tfrac{a+b}{2}\Big)dt$$

$$\approx \frac{b-a}{2}\Big[f\Big(\tfrac{a+b}{2} - \tfrac{b-a}{2\sqrt{3}}\Big) + f\Big(\tfrac{a+b}{2} + \tfrac{b-a}{2\sqrt{3}}\Big)\Big].$$

$\int_0^2 x^3 dx$: $t_{1,2} = 1 \mp \frac{1}{\sqrt{3}}$, $\approx 1 \cdot [(1 - \frac{1}{\sqrt{3}})^3 + (1 + \frac{1}{\sqrt{3}})^3] = 4$. *Exact! (deg 3 ≤ 3).*

12. FOURIER SERIES

***Big idea**: any periodic function can be decomposed into sines & cosines of different frequencies. Coefficients a_k, b_k (or c_k) tell you "how much" of frequency k is in f. Orthogonality is the magic that lets you extract one coefficient at a time: integrating $f \cdot \cos(kt)$ kills all other frequencies (they integrate to zero) leaving only the k-th contribution.*

Period T:

$$f(t) = a_0 + \sum_{k=1}^{\infty}\Big[a_k \cos\tfrac{2\pi k t}{T} + b_k \sin\tfrac{2\pi k t}{T}\Big]$$

Period 2π:

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f dt, \;\; a_k = \frac{1}{\pi}\int_0^{2\pi} f\cos(kt)dt,$$

$$b_k = \frac{1}{\pi}\int_0^{2\pi} f\sin(kt)dt.$$

Complex form:

$$f(t) = \sum_{k=-\infty}^{\infty} c_k e^{ikt}, \;\; c_k = \frac{1}{2\pi}\int_0^{2\pi} f(t)e^{-ikt}dt.$$

Orthogonality on $[0, 2\pi]$:

$$\int_0^{2\pi} \cos(kt)\sin(jt)dt = 0 \quad \forall j, k \in \mathbb{Z}.$$

$$\int_0^{2\pi} \cos(kt)\cos(jt)dt = 0 \;\; (k \neq j).$$

$$\int_0^{2\pi} \sin(kt)\sin(jt)dt = 0 \;\; (k \neq j).$$

$$\int_0^{2\pi} \sin(kt)dt = \int_0^{2\pi} \cos(kt)dt = 0 \;\; (k \neq 0).$$

Use to **isolate** one coef at a time.

13. DFT & FFT

***Big idea**: DFT is the discrete cousin of Fourier series — it takes N samples of a signal and returns N complex numbers F_k, each measuring the amount of frequency k present. It implicitly treats your data as* periodic *(wraps around). F_0 is the DC/average, higher k = higher frequency.*

Roots of unity: $W = W_N = e^{2\pi i/N}$. $W^N = 1$, $W^{N/2} = -1$.
DFT pair:

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}, \quad f_n = \sum_{k=0}^{N-1} F_k W^{nk}.$$

DFT orthogonality:

$$\sum_{j=0}^{N-1} W^{jk} W^{-j\ell} = \sum_{j=0}^{N-1} W^{j(k-\ell)} = N\delta_{k\ell}.$$

Facts: $F_0 =$ average; F_k periodic mod N; **real input** $\Rightarrow F_k = \overline{F_{N-k}}$ (conj. sym.); naive cost $O(N^2)$.
Parseval (Assn4 Q4):

$$\sum_{k=0}^{N-1} F_k \overline{F_k} = \frac{1}{N}\sum_{n=0}^{N-1} f_n \overline{f_n}.$$

W_N identity (Assn4 Q5):

$$W_N^{-nk} + W_N^{-(N-n)k} = 2\cos\big(\tfrac{2\pi nk}{N}\big).$$

***DFT by hand,** $N = 4$, $f = (4, 8, -4, 4)$, $W = i$:*
$F_k = \frac{1}{4}\sum f_n W^{-nk} = \frac{1}{4}\sum f_n (-i)^{nk}$.
$F_0 = \frac{1}{4}(4 + 8 - 4 + 4) = 3$.
$F_1 = \frac{1}{4}(4 + 8(-i) + (-4)(-1) + 4(i)) = \frac{1}{4}(8 - 4i) = 2 - i$.
$F_2 = \frac{1}{4}(4 + 8(-1) + (-4)(1) + 4(-1)) = \frac{1}{4}(-12) = -3$.
$F_3 = \frac{1}{4}(4 + 8(i) + (-4)(-1) + 4(-i)) = \frac{1}{4}(8 + 4i) = 2 + i$.
Check $F_3 = \overline{F_1}$. ✓

***Why FFT is fast**: naive DFT does N^2 multiplies. FFT uses divide-&-conquer: split N-pt DFT into two $N/2$-pt DFTs using $W^{N/2} = -1$, then recurse. Total $O(N \log_2 N)$. For $N = 1024$: DFT $\approx 10^6$ ops, FFT $\approx 10^4$ — 100× faster.*

FFT (Cooley-Tukey): split into half-length DFTs:

$$g_n = \tfrac{1}{2}(f_n + f_{n+N/2}),$$

$$h_n = \tfrac{1}{2}(f_n - f_{n+N/2})W^{-n}$$

for $n = 0, \dots, N/2 - 1$. Then $F_{\text{even}} = \text{DFT}(g)$, $F_{\text{odd}} = \text{DFT}(h)$. Cost $O(N \log_2 N)$; needs $N = 2^m$ (pad w/ zeros if not).
Butterfly (one pair $\to$ one pair):
$(f_n, f_{n+N/2}) \to (g_n,\ h_n)$ with W-factor "twiddle".
Output is bit-reversed ; unscramble by reversing binary indices, e.g. $f_5 = f_{101} \to$ pos $101_r = 101$, $f_3 = f_{011} \to$ pos $110 = 6$.

***FFT** $N = 8$, $f = (4, 3, 2, 1, 4, 3, 2, 1)$:*
Stage 1 ($W = e^{i\pi/4}$): $g = (4, 3, 2, 1)$, $h = (0, 0, 0, 0)$ (periodic data!).
Stage 2 ($W = e^{i\pi/2} = i$): on g: $g_{top} = (3, 2)$, $h_{top} = (1, -i)$. Bottom all 0.
Stage 3 ($W = -1$): $(3, 2) \to (2.5, 0.5)$. $(1, -i) \to (0.5 - 0.5i, 0.5 + 0.5i)$.
Bit-reverse out: $F = [2.5, 0, 0.5 - 0.5i, 0, 0.5, 0, 0.5 + 0.5i, 0]$. Note conj. sym.

IFFT: same algorithm with $U = W$ (not W^{-1}), multiply by N at end.
2D FFT: FFT on rows, then cols. Cost $O(MN(\log_2 M + \log_2 N))$.
Compression: FFT $\to$ zero small coefs ($|F_{ij}| <$ tol) $\to$ IFFT $\to$ take real, clamp to $[0, 255]$.
Aliasing & Nyquist: highest representable $|k| \leq N/2$. Freqs above Nyquist fold back. Fix: sample faster or low-pass filter.

***Aliasing intuition**: if you sample a spinning wheel too slowly, it looks like it spins backward (wagon-wheel effect). Same with data: a high-frequency signal undersampled* masquerades *as a low-frequency one. Nyquist rule: to resolve frequency f, sample at rate $> 2f$.*

Exam tricks: (1) periodic real data $\Rightarrow$ FFT halves cancel $\to h = 0$. (2) F_0 is always sum (or avg $\cdot$const). (3) Real $\Rightarrow$ conj. sym.: only need half.

KEY FORMULAS CHEAT SHEET

Error orders (composite): Mid $O(h^2)$, Trap $O(h^2)$, Simp $O(h^4)$, 2-pt Gauss exact deg 3.
Deg of precision : Mid 1, Trap 1, Simp 3, n-pt Gauss $2n-1$.
Spline slope tridiag row i: $\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i)s_i + \Delta x_{i-1} s_{i+1} = 3(\Delta x_i y_{i-1}' + \Delta x_{i-1} y_i')$.
Hermite coefs: $a = y$, $b = s$, $c = \frac{3y' - 2s - s_+}{\Delta x}$, $d = \frac{s_+ + s - 2y'}{\Delta x^2}$.
Quadratic formula : $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$.
Common derivs: $(e^x)' = e^x$, $(\sin x)' = \cos x$, $(\ln x)' = 1/x$, $(\sqrt{x})' = 1/(2\sqrt{x})$, $(1/x)' = -1/x^2$.
Geometric sum: $\sum_{k=0}^{n-1} r^k = \frac{1 - r^n}{1 - r}$ $(r \neq 1)$.
Roots i: $i^0 = 1$, $i^1 = i$, $i^2 = -1$, $i^3 = -i$; $e^{i\pi/4} = \frac{\sqrt{2}}{2}(1 + i)$.