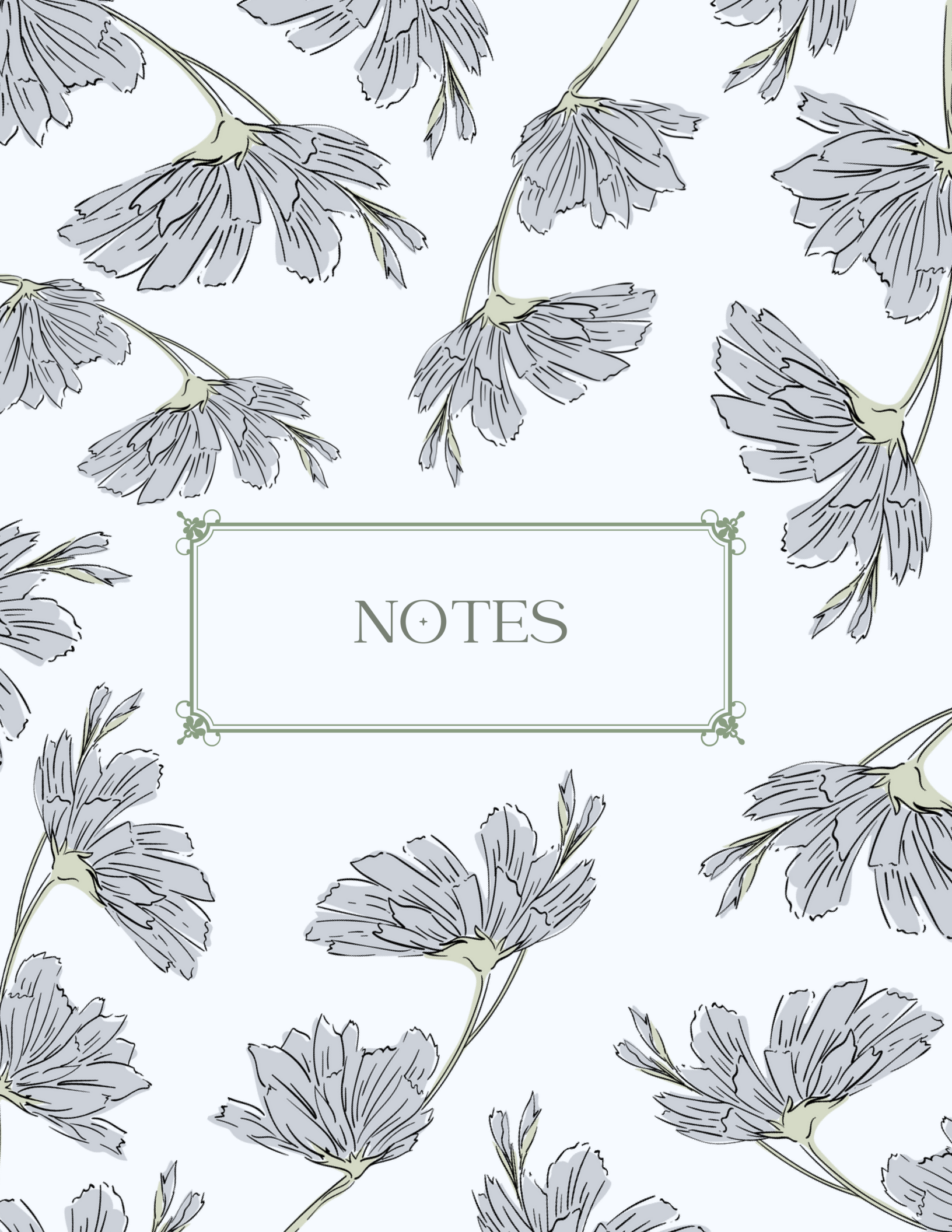
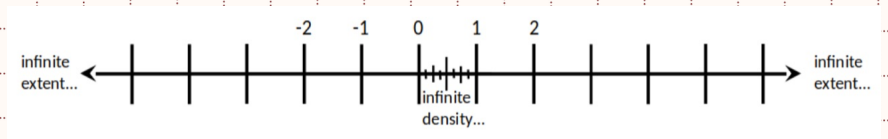

NOTES

floating point number systems

date 01/06/2025

- defn of **infinite**:
  - ↳ in **extent**: $\exists x$ st $|x|$ is arbitrarily large
    - e.g. for a # $r_1 \in \mathbb{R}$, $\exists$ another # r_2 st $|r_1| < |r_2|$
  - ↳ in **density**: any interval $a \leq x \leq b$ contains infinitely many #s
    - $I = \{x \mid a \leq x \leq b\}$
  - ↳ e.g.



  - ↳ problem: computers can't rep infinite / arbitrarily large #s
    - standard / partial solution is to use floating point #s to apx reals
- **floating point systems**: apx rep of real #s using finite #bits
- **analytical** soln is exact ans to problem
- **numerical** soln is apx soln produced by algo
  - ↳ often in floating point
- **numerical errors**:
  - ↳ e.g.

$$12 + \sum_{i=1}^{100} 0.01 = 12 + 100(0.01)$$
$$= 12 + 1$$
$$= 13$$

True ans: 13

Assume we can only keep 2 digits of accuracy each time. Perform sum 1 addition at a time, then round.

$$(\underbrace{(12 + 0.01)}_{12.01 \simeq 12} + 0.01) + \ldots \simeq 12$$

Numerical ans: 12 ⟶ wrong

Taylor series lets us apx fcn val in neighbourhood, assuming we know its val + derivatives at some nearby point a.

  - ↳ e.g. Taylor series

We want to apx eval $e^{-5.5}$ w/o exponential function.

Taylor series: $f(x) = f(a) + (x-a)f'(a) + \frac{(x-a)^2}{2!}f''(a) + \frac{(x-a)^3}{3!}f^{(3)}(a) + \ldots$

- apply to $f(x) = e^x$ at $a = 0$:

$$e^x \simeq e^0 + (x-0)e^0 + \frac{(x-0)}{2!}e^0 + \frac{(x-0)}{3!}e^0 + \ldots$$
$$\simeq 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots$$

- plug in $x = -5.5$ w/ 5 digits of accuracy

$$e^{-5.5} \simeq 1 - 5.5 + 15.125 - 27.730 + \ldots$$
$$\simeq 0.0026363$$

Correct val: $e^{-5.5} \simeq 0.0040868$

Another Taylor series apx: $e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots}$

- plug in $x = -5.5$ w/ 5 digits of accuracy to get $e^{-5.5} \simeq 0.0040865$
- instead of working w/ $e^{-5.5}$, work w/ $(e^{5.5})^{-1}$ since $e^{-r} = (e^r)^{-1}$
- this gives us numerical apx of 0.0040865 which is close to acc val of 0.0040868

- this works b/c all terms in this sum are +ve so there's no cancellation of info

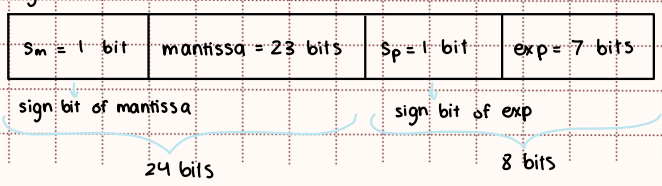
- numerical algos must work effectively under floating point arithmetic
- can express real # as **infinite expansion** rltive to some base β
  - ↳ e.g. $\frac{73}{3}$
    - base 10: 24.333...
    - base 2: 11000.010101...
- multiply # by power of β to shift it into **normalized form**: $0.d_1 d_2 d_3 \ldots \times \beta^p$
  - ↳ d_i are digits in base β
    - i.e. $0 \leq d_i < \beta$
  - ↳ $d_1 \neq 0$
  - ↳ exponent p is int

> May see other normalized conventions like $d_1.d_2 d_3 \ldots \times \beta^p$ (not in this course).

- **density/precision** is bounded by limiting #digits, t
- **extent/range** is bounded by limiting range of values for exp p
- non-zero floating point rep has finite form: $\pm\, 0.d_1 d_2 \ldots d_t \times \beta^p$
  - ↳ $L \leq p \leq U$
  - ↳ $d_1 \neq 0$
  - ↳ 0 is special case
  - ↳ floating point system w/ 4 int params is $F = \{\beta, t, L, U\}$
    - β: base
    - t: mantissa
    - L: lower bound
    - U: upper bound
- if exp too big ($> U$) or too small ($< L$), system can't rep #
  - ↳ when arithmetic ops gen this type of #:
    - **overflow**: too big
      - → produce $\pm\infty$ (or NaN, which means not a #)
    - **underflow**: too small
      - → rounds to 0
  - ↳ e.g. exception handling

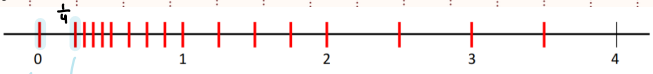
	Example	Result
Invalid op	0/0	NaN
Division by 0	1/0	$\pm\infty$
Overflow	$N_{max}+1$	$\pm\infty$
Underflow	$N_{min}/2$	0

- 2 most common floating point systems:
  - ↳ IEEE single precision (32 bits): $\{B=2, t=24, L=-126, U=127\}$
  - ↳ IEEE double precision (64 bits): $\{B=2, t=53, L=-1022, U=1023\}$
  - ↳ almost always implemented directly in hardware (e.g. CPU, GPU)
    - e.g. IEEE 32-bit



date

- fixed point #s : # digits after decimal point is fixed
  - ↳ e.g. 10.234 , 171.001
- unlike fixed point , floating point #s aren't evenly spaced
  - ↳ e.g. $F = \{\beta = 2,\ t = 3,\ L = -1,\ U = 2\}$



$(0.100)_2 = 0 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} = \frac{1}{4}$

$0, 0.100 \times 2^{-1}, 0.101 \times 2^{-1}, 0.110 \times 2^{-1}, 0.111 \times 2^{-1}, 0.100 \times 2^{0},$
$0.101 \times 2^{0}, 0.110 \times 2^{0}, \ldots\ldots\ldots, 0.110 \times 2^{2}, 0.111 \times 2^{2}$

- rounding modes for converting real # into FP # :
  - ↳ round-to-nearest : rounds to closest avail # in F
    - usually default
    - break ties by rounding $\frac{1}{2}$ up
  - ↳ truncation (chopping) : rounds to next # in F twd 0
    - i.e. discard digits after t^{th}
- e.g.

Express 253.9 in a floating point system with base $\beta = 10, t = 6$ digits, $L = -5, U = 5$.

Process:

1. Express in base $\beta = 10$: 253.9
2. Shift to get leading 0, set exponent p: 0.2539×10^{3} , $p = 3$
3. Pad or round/truncate to t digits: 0.253900×10^{3}

- e.g.

Express 8.25 in a floating point system with base $\beta = 2, t = 7$ digits, $L = -5, U = 5$.

$8.25 = (1000.01)_2$

$0.25 = \frac{1}{4} = 0 \times 2^{-1} + 1 \times 2^{-2} = (0.01)_2$

$1000.01 = 0.1000001 \times 2^{4}$, $p = 4$

Ans: 0.1000010×2^{4}

- e.g.

Express 1030.9671 in a floating point system with base $\beta = 10$, $t = 6$ digits, $L = -5, U = 5$

Process:

1. Express in base $\beta = 10$: 1030.9671
2. Shift to get leading 0, set exponent p: 0.10309671×10^{4} , $p = 4$
3. Pad or round/truncate to t digits:
   - With rounding: 0.103097×10^{4}
   - With truncation: 0.103096×10^{4}

- error of algo is diff btwn x_{exact} (true soln) + x_{apx} (apx soln)

date

↳ absolute error : $E_{abs} = |x_{exact} - x_{apx}|$

↳ rltive error: $E_{rel} = \frac{|x_{exact} - x_{apx}|}{x_{exact}}$

Might sometimes consider signed error (i.e. no abs).

↳ e.g.

Let $x_{exact} = 220$, $x_{approx} = 198$

Compute E_{abs} and E_{rel}.

What does each tell us?

$E_{abs} = |220 - 198| = 22$

$E_{rel} = \frac{|220-198|}{220} = 0.1$

· rltive error often more useful b/c it's indep of magnitudes of #s involved + relates to #significant digits in result

↳ significant digits are those starting w/ leftmost non-zero digit

• e.g. $e^{-5.5} \simeq 0.0040868$ (5 significant digits)

↳ result is correct to roughly s digits if $E_{rel} \leq 10^{-s}$ or $0.5 \times 10^{-s} \leq E_{rel} \leq 5 \times 10^{-s}$

• e.g.

$x_1 \simeq 0.0026363$

$E_{rel} \simeq 3.5 \times 10^{-1}$

$0.5 \times 10^{-1} \leq E_{rel} \leq 5 \times 10^{-1} \longrightarrow s = 1$

• e.g.

$x_2 \simeq 0.0040865$

$E_{rel} \simeq 0.7 \times 10^{-4}$

$0.5 \times 10^{-4} \leq E_{rel} \leq 5 \cdot 10^{-4} \longrightarrow s = 4$

· for FP system F, rltive err btwn $x \in \mathbb{R}$ + its FP apx, $fl(x)$, has bound E

↳ $E_{rel} = \frac{|x - fl(x)|}{|x|} \leq E$ for +ve val of E

$|x - fl(x)| \leq E \cdot |x|$

Since $|x| - |fl(x)| \leq |x - fl(x)|$:

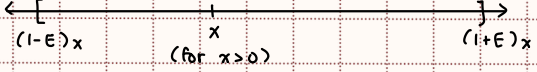
$|x| - |fl(x)| \leq |x| \cdot E$

$(1-E)|x| \leq |fl(x)|$ (1)

$|fl(x)| \leq (1+E)|x|$ (2)

(1), (2) $\Rightarrow (1-E)|x| \leq |fl(x)| \leq (1+E)|x|$

• i.e. $fl(x)$ is somewhere in neighbourhood of x



· machine epsilon: max rltive error, E, for FP system

↳ aka unit round-off error

↳ smallest val st $fl(1+E) > 1$ under given FP system

↳ e.g. $F = \{\beta, t, L, U\}$, assuming chopping

$1 = 0.100\ldots00 \times \beta^1$ (t)

$1 + E = 0.100\ldots01 \times \beta^1$ (t)

$E = 0.00\ldots01 \times \beta^1$ (t)

date

$= \beta^{-t} \times \beta^{1}$

$E = \beta^{1-t}$

↳ rule: $fl(x) = x(1+\delta)$ for some $|\delta| \leq E$

↳ for FP system w/ (β, t, L, U):

- rounding: $E = \frac{1}{2}\beta^{1-t}$
- truncation: $E = \beta^{1-t}$

↳ e.g

What is E for $F = \{\beta = 10, t = 3, L = -5, U = 5\}$?

Rounding:

$E = \frac{1}{2}(10)^{1-3}$

$= 5 \times 1^{-2}$

$= 5 \times 10^{-3}$

Truncation:

$E = (10)^{1-3}$

$= 10^{-2}$

error analysis + stability

date 01/09/2025

ERROR ANALYSIS

- IEEE std requires for $w, z \in F$, $w \oplus z = fl(w+z) = (w+z)(1+\delta)$
  - ↳ $\oplus$ denotes FP addition
  - ↳ $|\delta| \leq E$
    - i.e. assuming no over/underflows occur
  - ↳ i.e. same as FP apx of exact real result
    - $w \oplus z = fl(w+z)$

      $fl(x) = (1+\delta)x$

      $fl(w+z) = (1+\delta)(w+z) = w \oplus z$
  - ↳ only applies to individual FP ops, not sequence of ops
    - generally not true that $(a \oplus b) \oplus c = a \oplus (b \oplus c) = fl(a+b+c)$
      - → in $\mathbb{R}$, it's true that $(a+b)+c = a+(b+c) = a+b+c$
    - result is order-dependent → no associativity
- FP error can accumulate across multiple ops
  - ↳ e.g. use E to give bound on relative error of $(a \oplus b) \oplus c$

$$E_{rel} = \frac{|(a \oplus b) \oplus c - (a+b+c)|}{|a+b+c|}$$

$$= \frac{|((a+b)(1+\delta_1)) \oplus c - a - b - c|}{|a+b+c|}$$

$$= \frac{|((a+b)(1+\delta_1)+c)(1+\delta_2) - a - b - c|}{|a+b+c|}$$

$$= \frac{|a+b+(a+b)\delta_1 + c)(1+\delta_2) - a - b - c|}{|a+b+c|}$$

$$= \frac{|a+b+c + (a+b)\delta_1 + (a+b+c+(a+b)\delta_1)\delta_2 - a - b - c|}{|a+b+c|}$$

$$= \frac{|(a+b)\delta_1 + (a+b)\delta_1\delta_2 + (a+b+c)\delta_2|}{|a+b+c|}$$

$$\leq \frac{|(a+b)\delta_1| + |(a+b)\delta_1\delta_2| + |(a+b+c)\delta_2|}{|a+b+c|}$$

$$\leq \frac{|a+b||\delta_1|}{|a+b+c|} + \frac{|a+b||\delta_1||\delta_2|}{|a+b+c|} + \frac{|a+b+c||\delta_2|}{|a+b+c|}$$

$$\leq \frac{|a+b|E}{|a+b+c|} + \frac{|a+b|E^2}{|a+b+c|} + E$$

$$\leq \frac{|a+b|}{|a+b+c|}(E+E^2) + E$$

$$E_{rel} \leq \frac{|a|+|b|+|c|}{|a+b+c|}(2E+E^2)$$

  - ↳ many useful computations take $O(10^m)$ arithmetic ops
    - $m = 2, 3, 4, 5, \ldots$
  - ↳ analysis describes worst case error magnification as fcn of input data
    - actual error could be much less
  - ↳ $\frac{|a|+|b|+|c|}{|a+b+c|}$ describes how much E can be scaled by to give total error when summing 3 #s (a, b, c)
    - worst case magnification when $|a+b+c| \ll |a|+|b|+|c|$
      - → occurs when quantities have differing signs + similar magnitudes, leading to cancellation

- e.g.

Perform $(a \oplus b) \oplus c$ with $a = 2000, b = -3.234, c = -2000$ for $F = \{10, 4, -10, 10\}$, with rounding.

We'll consider:

1. the computed and true results for $(a \oplus b) \oplus c$.
2. the factor $\frac{|a|+|b|+|c|}{|a+b+c|}$ and resulting error bound.

  - ↳ scenario 1:

True result: -3.234

FP result: -3.000

$2000 - 3.234 = 1996.766 \rightsquigarrow 1997$

$(2000 \oplus -3.234) \oplus -2000 = -3.000$

Error bound:

$E = \frac{1}{2}\beta^{1-t}$

$= \frac{1}{2}(10)^{-3}$

$E_{rel} \leq \frac{|a|+|b|+|c|}{|a+b+c|}(2\varepsilon + \varepsilon^2)$ (ignored)

$\leq \frac{4003.234}{3.234} \cdot 2\left(\frac{1}{2}(10)^{-3}\right)$

≤ 1.238 → no correct digits

- $E_{rel} \leq 1.238$ is weak bound
- actual relative error $= \frac{0.234}{3.234} \approx 0.072$
  - → $0.5 \cdot 10^{-1} \leq 0.72 \cdot 10^{-1} \leq 5 \cdot 10^{-1}$, $s = 1$
  - → roughly 1 digit correct

↳ scenario 2: $(a \oplus c) \oplus b$ → diff order

True result: -3.234

FP result: -3.234

$(2000 \oplus 2000) \oplus -3.234$

$= 0 \oplus -3.234$

$= -3.234$

- relative error = 0
- reordering ops often changes results
- error bound is same as before b/c it only gives worst case

↳ scenario 3: $(a \oplus b) \oplus c$ for $a = -2000$, $b = -3.234$, $c = -2000$, $F = \langle 10, 4, -10, 10 \rangle$

True: -4003.234

FP: -4003

$(-2000 \oplus -3.234) \oplus -2000$

$= -2003.234 \oplus -2000$

$= -4003$

Error bound:

$E_{rel} \leq \frac{|a|+|b|+|c|}{|a+b+c|}(2\varepsilon + \varepsilon^2)$

$\leq 1\,(2\varepsilon + \varepsilon^2)$

$\approx 2\varepsilon$

$= 10^{-3}$

Actual $E_{rel} \approx 5.8 \cdot 10^{-5}$

- exprs w/o cancellation often have better bounds on error

- catastrophic cancellation occurs when subtracting #s of abt same magnitude, when input #s contain error
  ↳ e.g.

$$\underbrace{173.00}_{\text{correct digits}}\underbrace{064}_{\text{Erroneous digits}} - \underbrace{173.00}_{\text{correct digits}}\underbrace{017}_{\text{Erroneous digits}} = 0.00\underbrace{046}_{\text{Only erroneous digits left!}}$$

- all sig digs cancelled out so result may have no correct digits

- benign cancellation: if input quantities are known to be exact, have usual guarantee on FP ops
  ↳ $w \ominus z = fl(w - z) = (w - z)(1 + \delta)$ for $w, z \in F$

date

- common FP errors:
  - ↳ adding large + small #s (i.e. very diff magnitudes)
    - smaller digits get lost / swamped
    - rule of thumb: prefer to sum #s of apx same size + sign
  - ↳ subtracting nearby #s that contain error
    - loss of accuracy due to catastrophic cancellation
    - rule of thumb: reformulate computations to avoid cancellation
    - e.g.

So, why did we earlier see that

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \dots$$

performs so much worse than

$$e^{-x} = \frac{1}{e^x} = \frac{1}{1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots}$$

as an algorithm to evaluate $e^{-5.5}$ in FP?

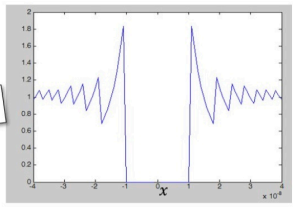
→ $e^{-x} = \frac{1}{e^x}$ avoids alternating signs that caused cancellation

- e.g. visualization of cancellation error
  - ↳ $1 - \cos^2 x = \sin^2 x$

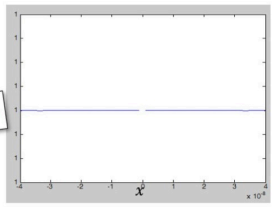
Compare: $f(x) = \dfrac{1 - \cos^2 x}{x^2}$ and $g(x) = \dfrac{\sin^2 x}{x^2}$ near $x = 0$.

True solution approaches 1 as $x \to 0$.

$f(x)$ Bad!!



$g(x)$ Good!!



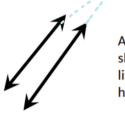
(Plot computed with IEEE double precision)

STABILITY

- problem P w/ input I + output O is well-conditioned if small change in input ΔI gives small change in output ΔO
  - ↳ otherwise, P is ill-conditioned
    - ↳ property of problem + independent of specific implementation (e.g. algo, hardware)
- conditioning is all relative
- e.g. geometric conditioning

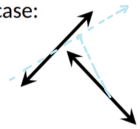
Do two (infinite) lines intersect, and if so where?

Ill-conditioned case: (parallel)



A tiny error (change in the slope) of one of these lines changes the answer hugely!

Well-conditioned case:



A tiny change in one of these lines changes the answer only a tiny bit.

date

- algo is **numerically unstable** if any initial error in data is magnified by algo
  - ↳ can lead to meaningless results for seemingly reasonable methods on reasonable problems
- conditioning vs stability:
  - ↳ conditioning is how sensitive problem itself is to errors / changes in input
  - ↳ stability is how sensitive algo is to errors / changes in input
  - ↳ algo can be unstable for well-conditioned problem
  - ↳ ill-conditioned problems limits how well algo can perform
- e.g. stability analysis

Consider the integration problem

$$I_n = \int_0^1 \frac{x^n}{x+\alpha} dx$$

for a given n where α is some fixed constant.

Course notes give a recursive algorithm to solve it, for $n \geq 0$:

$$I_0 = \log \frac{1+\alpha}{\alpha},$$

and

$$I_n = \frac{1}{n} - \alpha I_{n-1}.$$

$I_1 = \frac{1}{1} - \alpha I_0 = 1 - \alpha I_0$

$I_2 = \ldots$

This is well-conditioned problem.

Let $(I_n)_E$ be exact val of I_n.

Let $(I_n)_A$ be apx val of I_n.

ε_0 is diff btwn exact + apx vals of I_0.

$\varepsilon_0 = (I_0)_A - (I_0)_E$ → initial error

$\varepsilon_n = (I_n)_A - (I_n)_E$

$(I_n)_E = \frac{1}{n} - \alpha (I_{n-1})_E$ (1)

$(I_n)_A = \frac{1}{n} - \alpha (I_{n-1})_A$ (2)

Subtracting (1), (2):

$(I_n)_A - (I_n)_E = \frac{1}{n} - \alpha (I_{n-1})_A - \left(\frac{1}{n} - \alpha (I_{n-1})_E\right)$

$$\begin{aligned} \varepsilon_n &= -\alpha((I_{n-1})_A - (I_{n-1})_E) \\ &= -\alpha(\varepsilon_{n-1}) \\ &= -\alpha(-\alpha\,\varepsilon_{n-2}) \\ &\vdots \\ &= (-\alpha)^n \varepsilon_0 \end{aligned}$$

$\varepsilon_n = (-\alpha)^n \varepsilon_0$

↳ if $|\alpha| > 1$, ε_n grows fast so algo is unstable

↳ if $|\alpha| < 1$, ε_n decreases fast so algo is stable

↳ hence, recurrence algo is unstable for solving $I_n = \int_0^1 \frac{x^n}{x+\alpha} dx$ for vals of $|\alpha| > 1$

intro to interpolation

date 01/22/2025

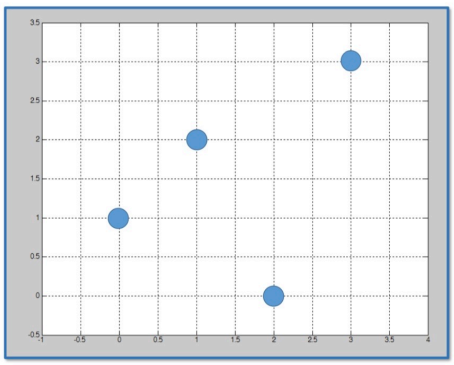
- given set of data pts from unknown fcn $y = p(x)$, can we apx p's val at other pts?
  - ↳ data pts are pairs (x_i, y_i) that satisfy $y_i = p(x_i)$
  - ↳ e.g.

e.g., Given $p(x_1) = y_1, p(x_2) = y_2, \ldots, p(x_n) = y_n$.

Estimate $y = p(x)$ for any point x such that $x_1 \leq x \leq x_n$.

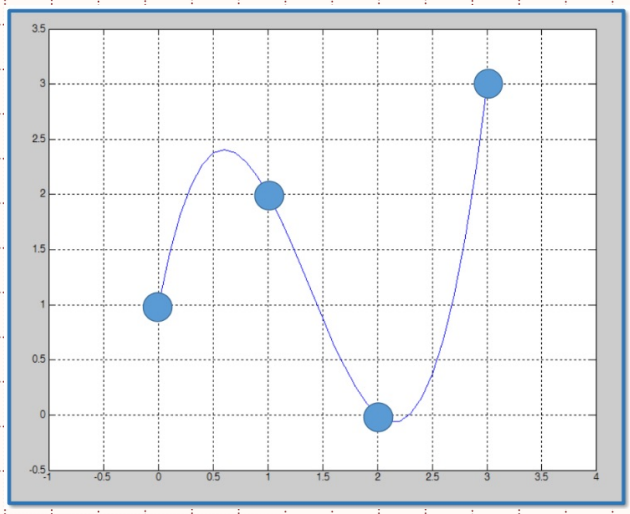
  - ↳ e.g. visualization

E.g. given points (x_i, y_i): $(0,1), (1,2), (2,0), (3,3)$



Find a function $p(x)$ that goes exactly through or **interpolates** all the points.

- 1 solution: $y = \frac{4}{3}x^3 - \frac{11}{2}x^2 + \frac{31}{6}x + 1$



→ $y(x)$ is interpolating fcn / interpolant
  - > not necessarily unique

- uses of interpolation:
  - ↳ fitting curves to data
    - regression in stats/ML
  - ↳ estimating unknown fcn's props
    - e.g. vals, derivatives
  - ↳ plays role in numerical methods for differentiation, integration, differential eqns, optimization, etc.

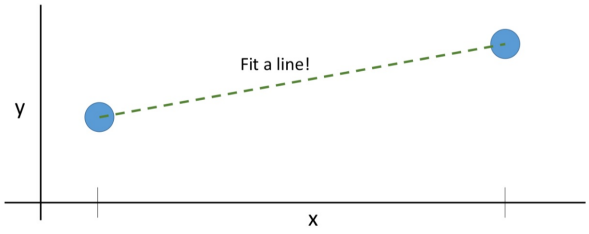
date

- methods for interpolating **few pts** (often ≤ 6):
  - ↳ polynomial interpolation:
    - Vandermonde matrices
    - Lagrange form
- methods for interpolating **many pts**:
  - ↳ piecewise interpolants:
    - piecewise linear
    - cubic splines

POLYNOMIAL INTERPOLATION

- **linear interpolation** is fitting line to 2 pts

Given just two points, how might we approximate points that lie in between?



- ↳ line eqn $y = ax + b$ for unknowns (a, b)
- ↳ solve since we have 2 eqns + 2 unknowns
  - for (x_1, y_1), (x_2, y_2), $\begin{cases} y_1 = ax_1 + b \\ y_2 = ax_2 + b \end{cases}$
- ↳ e.g.

Example: Given $(x_1, y_1) = (1, 2)$, $(x_2, y_2) = (-1, 4)$, find a and b for the line passing through the points.

$$\begin{cases} y_1 = ax_1 + b \\ y_2 = ax_2 + b \end{cases}$$

$$= \begin{cases} 2 = a + b \\ 4 = -a + b \end{cases}$$

$2 = a + b$ 1)

$+ \; 4 = -a + b$ 2)

$6 = 2b$

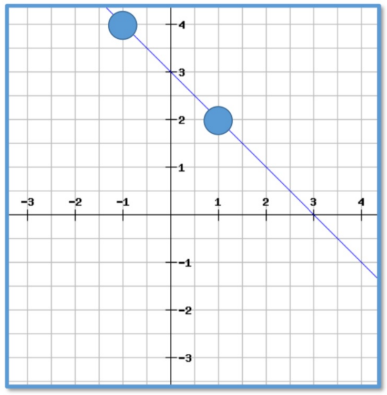
$b = 3$

Using 1):

$2 = a + 3$

$a = -1$

Ans: $y = -x + 3$



date

- **fit quadratic to 3 pts**
  - for 3 data pts, we find 3 unknown coeffs for $y = ax^2 + bx + c$
  - solve linear 3×3 system: $\begin{cases} ax_1^2 + bx_1 + c = y_1 \\ ax_2^2 + bx_2 + c = y_2 \\ ax_3^2 + bx_3 + c = y_3 \end{cases}$
    - matrix form: $\begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ x_3^2 & x_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$
- **polynomial interpolation** is generalizing to arbitrarily many pts
  - $p(x) = c_1 + c_2 x + c_3 x^2 + \dots + c_n x^{n-1}$
  - **unisolvence thm**: given n data pairs (x_i, y_i), $i = 1, \dots, n$ w/ distinct x_i, there's unique polynomial $p(x)$ of deg $\le n-1$ that interpolates data
  - each (x_i, y_i) pt gives 1 linear eqn: $y_i = c_1 + c_2 x_i + c_3 x_i^2 + \dots + c_n x_i^{n-1}$
  - solve **n×n linear system**
  - e.g.

> Very first example had 4 (x_i, y_i) pairs: $(0,1)$, $(1,2)$, $(2,0)$, $(3,3)$.
>
> What is the linear system needed to recover the coefficients of the cubic polynomial?
>
> $$y_i = c_1 + c_2 x_i + c_3 x_i^2 + c_4 x_i^3$$

For $(0,1)$: $c_1 + c_2(0) + c_3(0) + c_4(0) = 1$

For $(1,2)$: $c_1 + c_2(1) + c_3(1) + c_4(1) = 2$

For $(2,0)$: $c_1 + c_2(2) + c_3(4) + c_4(8) = 0$

For $(3,3)$: $c_1 + c_2(3) + c_3(9) + c_3(27) = 3$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 9 & 27 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \\ 3 \end{bmatrix}$$

$c_1 = 1$, $c_2 = \frac{31}{6}$, $c_3 = -\frac{11}{2}$, $c_4 = \frac{4}{3}$

- in general, we get linear system:

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

  - denote as $V\vec{c} = \vec{y}$
    - V is **Vandermonde matrix**
      - → invertible if x_i's are distinct
  - polynomial interpolation reduces to solving system of eqns
- **monomial form** is $p(x) = c_1 + c_2 x + c_3 x^2 + \dots + c_n x^{n-1}$
  - also written as $p(x) = \sum_{i=1}^{n} c_i x^{i-1}$
  - **monomial basis** is sequence $1, x, x^2, \dots$
  - monomial form is sum of coeffs c_i times basis functions x^i
- **Lagrange basis** is diff basis for interpolating polynomials
  - define Lagrange basis fcns $L_i(x)$ to construct polynomial as:
    $p(x) = y_1 L_1(x) + y_2 L_2(x) + \dots + y_n L_n(x) = \sum_{i=1}^{n} y_i L_i(x)$
    - y_i are coeffs + data vals ($y_i = p(x_i)$)
  - given n data pts (x_i, y_i), define $L_i(x) = \frac{(x-x_1)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$

- in numerator, $(x - x_i)$ DNE
- solving $L_i(x_j)$:
  - → $i = j$:

$$L_i(x_i) = \frac{(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)}{(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)} = 1$$

Numerator + denominator are same

  - → $i \neq j$:

$$L_i(x_j) = \frac{(x_j - x_1)\cdots(x_j - x_{i-1})(x_j - x_{i+1})\cdots(x_j - x_n)}{(x_i - x_1)\cdots(x_i - x_{i-1})(x_i - x_{i+1})\cdots(x_i - x_n)} = 0$$

(1 of these terms = 0)

- $L_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$
  - → ensures $p(x)$ must interpolate each x_i since:

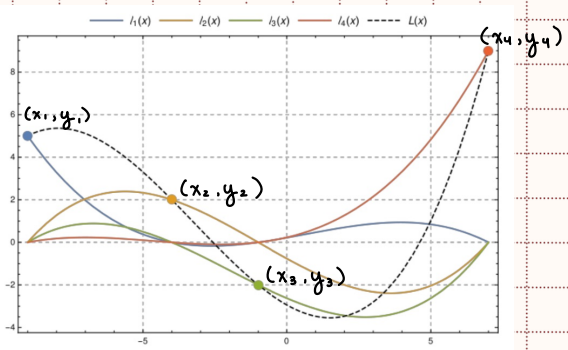
$$p(x_i) = y_1 \underbrace{L_1(x_i)}_{0} + y_2 \underbrace{L_2(x_i)}_{0} + \ldots + y_i \underbrace{L_i(x_i)}_{1} + \ldots + y_n \underbrace{L_n(x)}_{0}$$

$$p(x_i) = y_i$$

↳ e.g.

Each (scaled) Lagrange polynomial here is zero at all but one data point.

Summing them gives the desired curve (dashed black).



↳ e.g.

Consider the two points we fit a line to earlier:

$$(x_1, y_1) = (1, 2), \quad (x_2, y_2) = (-1, 4)$$

What are the corresponding L_i functions, and the polynomial $p(x)$ in terms of the L_i's?

$$L_1(x) = \frac{(x - x_2)}{(x_1 - x_2)} = \frac{x+1}{1-(-1)} = \frac{x+1}{2}$$

$$L_2(x) = \frac{(x - x_1)}{(x_2 - x_1)} = \frac{x-1}{-1-1} = \frac{x-1}{-2}$$

$$p(x) = \sum_{i=1}^{2} y_i L_i(n)$$

$$= 2\left(\frac{x+1}{2}\right) + 4\left(\frac{x-1}{-2}\right)$$

$$= x + 1 - 2(x-1)$$

$$= x + 1 - 2x + 2$$

$$= -x + 3$$

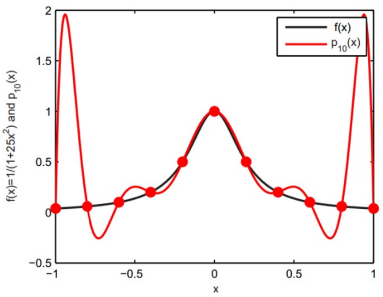
Ans: $p(x) = -x + 3$

- monomial + Lagrange bases give same interpolating polynomial
- might prefer Lagrange b/c no need to solve linear system

- e.g. polynomial interpolation for many pts

Consider fitting a degree 10 polynomial to 11 points sampled evenly from $f(x) = \frac{1}{1+25x^2}$. Is this a "good" approximation? No

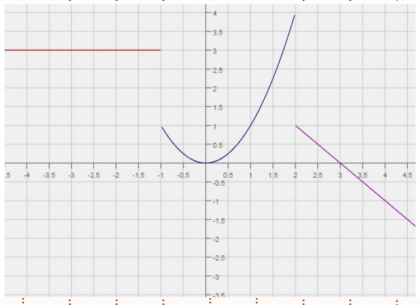
What do we observe?



- fitting high deg (i.e. $n \simeq 7, 8, 9, \ldots$) polynomial often gives excessive oscillation
  - ↳ aka **Runge's phenomenon**
  - ↳ strategies to avoid this:
    - select data pts in smarter way
      - → oscillations get worse at edges
      - → only works when we can choose data pts
    - fit even higher deg polynomials, but constrain derivatives to reduce "wiggliness"
    - fit lower deg polynomials that can minimize some error measure
      - → i.e. find apx polynomial instead
    - **piecewise polynomials**
- **piecewise fcn**: fcn w/ diff defns for distinct intervals of domain
  - ↳ e.g.

e.g.,

$$p(x) = \begin{cases} 3 & \text{if} \quad x \leq 1 \\ x^2 & \text{if} \quad -1 < x \leq 2 \\ -x+3 & \text{if} \quad 2 < x \end{cases}$$

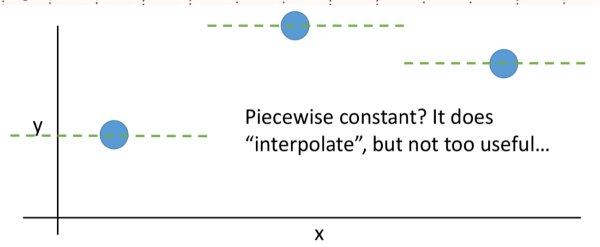


CUBIC SPLINES

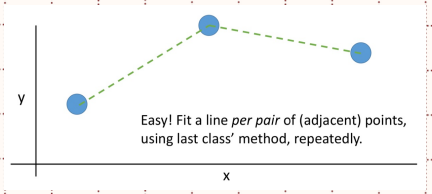
date 01/24/2024

INTERPOLATING WITH PIECEWISE FUNCTIONS

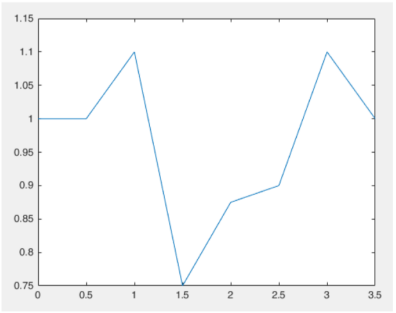
- piecewise constant is simplest piecewise fcn that interpolates given set of pts
  - ↳ e.g.



- at least want continuity
  - ↳ piecewise linear fcns constructed by fitting line per pair of pts
    - e.g.



    - e.g.



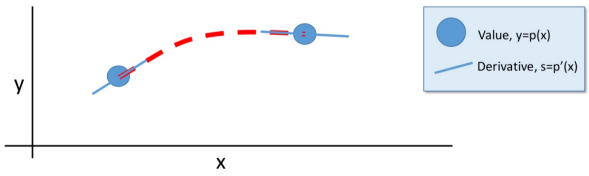
$$p(x) = \begin{cases} 1 & 0 \le x \le \frac{1}{2} \\ \frac{x}{5} + \frac{9}{10} & \frac{1}{2} \le x \le 1 \\ \frac{-7x}{10} + \frac{9}{5} & 1 \le x \le \frac{3}{2} \\ \cdots & \cdots \\ \cdots & \cdots \\ \frac{-x}{5} + \frac{17}{10} & 3 \le x \le \frac{7}{2} \end{cases}$$

Transition points have the same value, so the overall piecewise function is continuous.

    - often not satisfactory b/c we want to avoid kinks at data pts
  - ↳ aim for smoothness (i.e. continuity of derivatives)
    - aesthetic purposes
    - mathematical / numerical apps needing derivative info

HERMITE INTERPOLATION

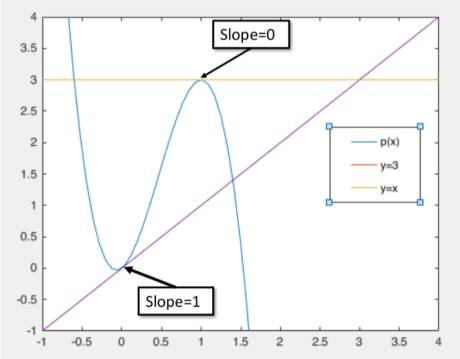
- Hermite interpolation is problem of fitting polynomial given fcn vals + derivatives
  - ↳ e.g.



date

↳ e.g. cubic

Given $p(0) = 0$, $p'(0) = 1$, $p(1) = 3$ and $p'(1) = 0$, determine the coefficients of the cubic polynomial $p(x) = a + bx + cx^2 + dx^3$.



$p(x) = a + bx + cx^2 + dx^3$

$p(0) = 0 \longrightarrow a = 0$

$p(1) = 3 \longrightarrow a + b + c + d = 3$

$0 + b + c + d = 3$

$b + c + d = 3$ (1)

$p'(x) = b + 2cx + 3dx^2$

$p'(0) = 1 \longrightarrow b = 1$

$p'(1) = b + 2c + 3d \longrightarrow b + 2c + 3d = 0$

$1 + 2c + 3d = 0$

$2c + 3d = -1$

From (1):

$1 + c + d = 3$

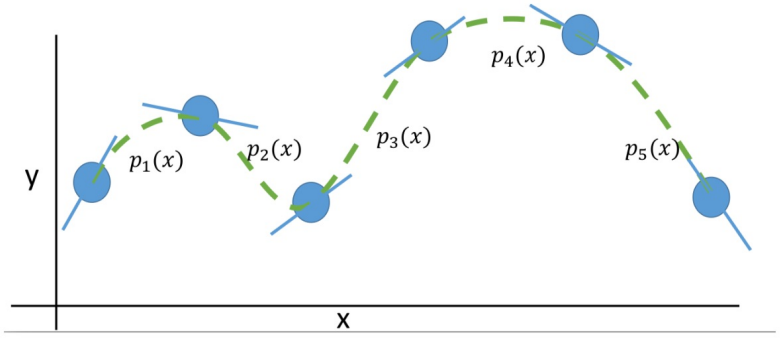
$c + d = 2$

$\begin{cases} a = 0 \\ b = 1 \\ c + d = 2 \\ 2c + 3d = -1 \end{cases} \longrightarrow c = 7,\ d = -5$

Ans: $p(x) = x + 7x^2 - 5x^3$

- used cubics for 2-pt Hermite interpolation problem b/c quadratics have only 3 coeffs per interval
  - ↳ insufficient # unknowns to satisfy 4 given vals

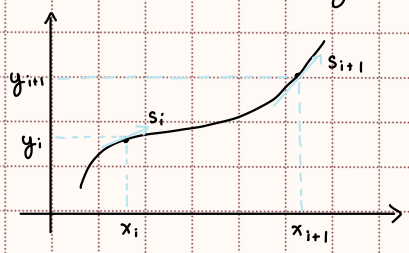
- general strategy for fitting polynomial w/ higher deg or higher derivative data:
  1) express $p(x)$ in general form w/ unknown coeffs
  2) differentiate to find exprs for needed derivatives of $p(x)$
  3) plug in val + derivative data to give linear eqns
  4) solve for polynomial coeffs
- fit many pts (given vals + 1st derivatives) w/ piecewise Hermite interpolation
  - ↳ use 1 cubic per pair of pts
  - ↳ sharing slope / derivative at pts ensures 1st derivative continuity
  - ↳ e.g.



date

↳ closed-form soln:

Given (x_i, y_i), (x_{i+1}, y_{i+1}), s_i, s_{i+1}, find $p(x)$ of deg ≤ 3 st $p(x_i) = y_i$, $p'(x_i) = s_i$, $p(x_{i+1}) = y_{i+1}$, $p'(x_{i+1}) = s_{i+1}$



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

$p'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$

Let $\Delta x_i = x - x_i$

$p(x_i) = y_i \rightarrow a_i = y_i$

$p'(x_i) = s_i \rightarrow b_i = s_i$

$p(x_{i+1}) = y_{i+1} \rightarrow a_i + b_i \Delta x_i + c_i \Delta x_i^2 + d_i \Delta x_i^3 = y_{i+1}$

$p'(x_{i+1}) = s_{i+1} \rightarrow b_i + 2c_i \Delta x_i + 3d_i \Delta x_i^2 = s_{i+1}$

Linear system:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & \Delta x_i & \Delta x_i^2 & \Delta x_i^3 \\ 0 & 1 & 2\Delta x_i & 3\Delta x_i^2 \end{bmatrix} \begin{bmatrix} a_i \\ b_i \\ c_i \\ d_i \end{bmatrix} = \begin{bmatrix} y_i \\ s_i \\ y_{i+1} \\ s_{i+1} \end{bmatrix}$$

Solving system:

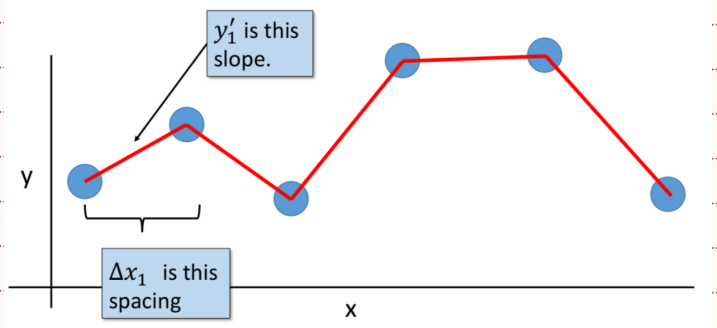
$a_i = y_i$

$b_i = s_i$

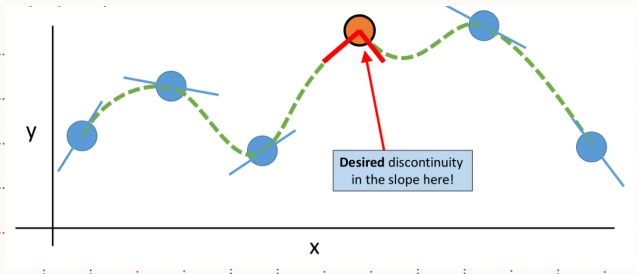
$c_i = \frac{3y_i' - 2s_i - s_{i+1}}{\Delta x_i}$

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

- $\Delta x_i = x_{i+1} - x_i$
- $y_i' = \frac{y_{i+1} - y_i}{\Delta x_i}$
- e.g.



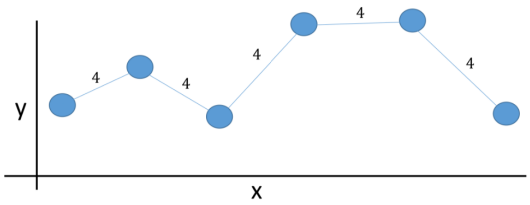
- knots: pts where interpolant transitions from 1 polynomial / interval to another
- nodes: pts where control pts / data are specified
- knots + nodes are same for Hermite interpolation, but can differ for other curve types (e.g. Bezier)
- when we intend to create/control kinks, specify diff derivatives on either side of node
  - ↳ e.g.



CUBIC SPLINES

- more common problem is fitting piecewise cubic to set of pts w/o derivative info s_i given + only y_i
- fit cubic $S_i(x)$ on each interval, but now require matching 1st + 2nd derivatives
  - ↳ require interpolating conditions on each interval $[x_i, x_{i+1}]$
    - $S_i(x_i) = y_i$
    - $S_{i+1}(x_{i+1}) = y_{i+1}$
  - ↳ require derivative conditions at each interior pt x_{i+1}
    - $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$
    - $S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$
    - i.e. at x_{i+1}, left + right 1st + 2nd derivatives agree
- w/n data pts, each interval has a cubic (4 unknowns) for n-1 intervals
  - ↳ $4n-4$ unknowns
  - ↳ e.g.

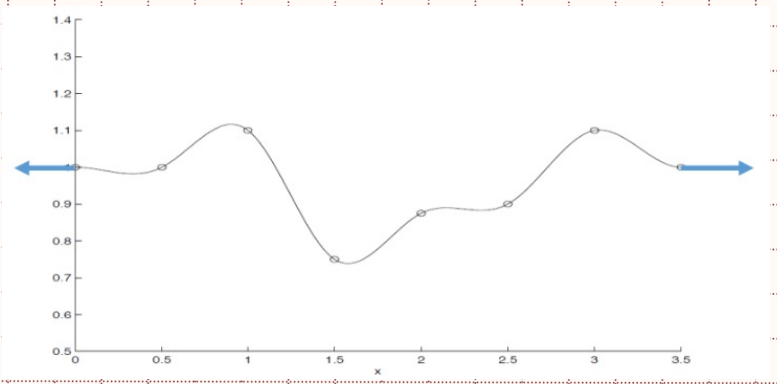
e.g.,
$n = 6$ points,
$n - 1 = 5$ intervals,
$\therefore 4 \times 5 = 20$ unknowns.



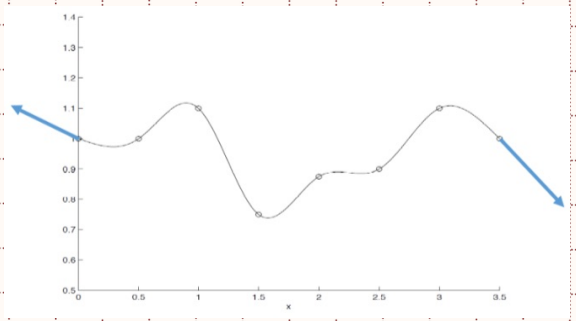
  - ↳ $4n-6$ eqns
    - each interval has 2 endpoint eqns: $\begin{cases} S_i(x_i) = y_i \\ S_{i+1}(x_{i+1}) = y_{i+1} \end{cases}$
      - → n-1 intervals ⇒ $2(n-1)$ eqns
    - each interior pt has 2 derivative conds: $\begin{cases} S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \\ S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \end{cases}$
      - → n-2 interior pts ⇒ $2(n-2)$ eqns
    - total $= 2(n-1) + 2(n-2)$
      $= 2n - 2 + 2n - 4$
      $= 4n - 6$
  - ↳ since $4n-6 < 4n-4$, there's more unknowns than eqns so not enough info for unique soln
    - need 2 more eqns (usually at domain endpoints) called boundary/end conditions
- choices for boundary conds (BC):
  - ↳ clamped BC: slope set to specific val
    - $S'(x_1)$, $S'(x_n)$ are specified
    - if both boundaries clamped, it's complete/clamped spline

date

- e.g. $S'(x_1) = S'(x_n) = 0$



↳ free BC: 2nd derivatives set to 0
- $S''(x_1) = S''(x_n) = 0$
- if both boundaries free, it's natural cubic spline
- curvature goes to 0 at end pts so curves straighten out
- e.g.



↳ periodic BC: start + end derivatives match each other
- $S'(x_1) = S'(x_n)$
- $S''(x_1) = S''(x_n)$

↳ not-a-knot BC: match 3rd derivatives on end segments
- $S_1'''(x_2) = S_2'''(x_2)$
- $S_{n-2}'''(x_{n-1}) = S_{n-1}'''(x_{n-1})$
- last 2 segments on end become same polynomial
- since there's no switch btwn polynomials, it's not a knot

↳ can mix + match BCs except for periodic

- Hermite vs cubic splines system size:
  - ↳ Hermite: each interval can be found independently
    - solve $n-1$ separate systems of 4 eqns each
  - ↳ cubic spline: must solve for all polynomials tgr
    - solve 1 system w/ $4(n-1)$ eqns
- naïve cubic spline system has $4n-4$ unknowns for n pts
  - ↳ basic algos (e.g. Gaussian elimination) for linear systems take $O(N^3)$ time for N unknowns
  - ↳ for special case of cubic splines, can do much better in $O(N)$ linear time
- comparing 2 strategies for smooth curves:

date

↳ Hermite interpolation:
- given vals + slopes
- separate small system per interval
- continuous 1st derivatives

↳ cubic spline interpolation:
- given vals only
- 1 big system for all coeffs
- continuous 1st + 2nd derivatives

use Hermite interpolation to build cubic spline:
1) express unknown polys w/ closed form Hermite eqns
2) treat s_i as unknowns
3) solve for s_i that gives it continuous 2nd derivs
   - i.e. force to satisfy cubic spline defn
4) given s_i, plug into closed form Hermite to recover coeffs (a_i, b_i, c_i, d_i)

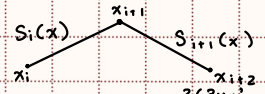
for cubic splines, 3 types of conds (ignoring ends):
↳ vals match at all interval endpoints } all satisfied using closed form Hermite eqns
↳ 1st derivs match at interior pts
↳ 2nd derivs match at interior pts
   - find s_i satisfying $S_i''(x) = S_{i+1}''(x)$

derivation of cubic splines eqns:
Start from Hermite formulas:

$S_i(x)$ is poly + small s_i are slopes at pts.

$$S_i(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$$
$$S_i''(x) = 2c_i + 6d_i(x-x_i)$$
$$= \frac{2(3y_i' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_i + s_{i+1} - 2y_i')}{\Delta x_i^2}(x-x_i)$$

To force matching 2nd derivs:
$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$ for $i = 1, 2, \ldots, n-2$



$$S_i''(x_{i+1}) = \frac{2(3y_i' - 2s_i - s_{i+1})}{\Delta x_i} + \frac{6(s_i + s_{i+1} - 2y_i')}{\Delta x_i^2}(x_{i+1} - x_i)$$
$$S_{i+1}''(x_{i+1}) = \frac{2(3y_{i+1}' - 2s_{i+1} - s_{i+2})}{\Delta x_{i+1}} + \frac{6(s_i + s_{i+1} - 2y_i')}{\Delta x_{i+1}^2}(x_{i+1} - x_{i+1})$$

(Δx_i; $(x_{i+1} - x_i)$; $(x_{i+1} - x_{i+1})^0$)

$$S_i''(x_{i+1}) = S_{i+1}''(x_{i+1})$$
$$\frac{2s_i + 4s_{i+1} - 6y_i'}{\Delta x_i} = \frac{6y_{i+1}' - 4s_{i+1} - 2s_{i+2}}{\Delta x_{i+1}}$$

↓ cross-multiply + take s_i terms to left

$$\Delta x_{i+1} s_i + 2(\Delta x_i + \Delta x_{i+1}) s_{i+1} + \Delta x_i s_{i+2} = 3(\Delta x_{i+1} y_i' + \Delta x_i y_{i+1}')$$

Reindexing w/ $i \to i-1$, get 1 eqn per interior node for $i = 2, \ldots, n-1$:

$$\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i) s_i + \Delta x_{i-1} s_{i+1} = 3(\Delta x_i y_{i-1}' + \Delta x_{i-1} y_i')$$

BC:
↳ need 2 more eqns for slopes at $i=1$, $i=n$ to complement $n-2$ interior node eqns
↳ clamped (given slope):
$S_1'(x_1) = s_1^*$
$S_{n-1}'(x_n) = s_n^*$
   - s_1^*, s_n^* are given

↳ natural / free / variational:

$S_1''(x_1) = S_{n-1}''(x_n) = 0$

$S_1''(x_1) = 2c_1 + 6d_1(x_1 - x_1)$

$0 = 2c_1$

$0 = \frac{3y_1' - 2s_1 - s_2}{\Delta x_1}$

simplifying ↓

$s_1 + \frac{1}{2}s_2 = \frac{3}{2}y_1'$

$S_{n-1}''(x_n) = 2c_{n-1} + 6d_{n-1}(x_n - x_{n-1})$ (Δx_{n-1})

$0 = \frac{2(3y_{n-1}' - 2s_{n-1} - s_n)}{\Delta x_{n-1}} + \frac{6(s_n + s_{n-1} - 2y_{n-1}')}{\Delta x_{n-1}}$

simplifying ↓

$\frac{1}{2}s_{n-1} + s_n = \frac{3}{2}y_{n-1}'$

· efficient splines summary:

↳ interior nodes ($i = 2, \dots, n-1$): $\Delta x_i s_{i-1} + 2(\Delta x_{i-1} + \Delta x_i)s_i + \Delta x_{i-1} s_{i+1} = 3(\Delta x_i y_{i-1}' + \Delta x_{i-1} y_i')$

↳ clamped BC ($i = 1, n$):

- $s_1 = s_1^*$
- $s_n = s_n^*$

↳ free BC ($i = 1, n$):

- $s_1 + \frac{1}{2}s_2 = \frac{3}{2}y_1'$
- $\frac{1}{2}s_{n-1} + s_n = \frac{3}{2}y_{n-1}'$

↳ solve for all s_i, then plug into closed-form Hermite exprs for each interval to recover final polys

· can write linear system for slopes in matrix/vector form: $T \cdot s = r \longrightarrow \begin{bmatrix} T \end{bmatrix}\begin{bmatrix} s_1 \\ \dots \\ s_i \\ \dots \\ s_n \end{bmatrix} = \begin{bmatrix} r_1 \\ \dots \\ r_i \\ \dots \\ r_n \end{bmatrix}$

↳ after solving for s, can recover a, b, c, d coeffs

· e.g.

> What is the linear system for s_i to fit a spline to the 4 points $(0,1), (2,1), (3,3), (4,-1)$ with clamped BC of $s_1 = 1$ and $s_4 = -1$?

Compute all Δx_i, $\Delta y_i'$ for $i = 1, \dots, 3$ intervals:

$\Delta x_i = x_{i+1} - x_i \longrightarrow$ $\Delta x_1 = 2 - 0 = 2$, $\Delta x_2 = 3 - 2 = 1$, $\Delta x_3 = 4 - 3 = 1$

$\Delta y_i' = \frac{y_{i+1} - y_i}{\Delta x_i} \longrightarrow$ $\Delta y_1 = \frac{1-1}{2} = 0$, $\Delta y_2 = \frac{3-1}{1} = 2$, $\Delta y_3 = \frac{-1-3}{1} = -4$

Find rows for $i = 1, \dots, 4$ in T (1 per node):

$i = 1$:

↳ boundary so $s_1 = 1$

↳ $T_1 = [1 \ 0 \ 0 \ 0]$, $r_1 = 1$

$i = 2$:

↳ interior

↳ $\Delta x_2 s_1 + 2(\Delta x_2 + \Delta x_1)s_2 + \Delta x_1 s_3 = 3(\Delta x_2 y_1' + \Delta x_1 y_2')$

$s_1 + 2(1+2)s_2 + 2s_3 = 3(1-0+2-2)$

$s_1 + 6s_2 + 2s_3 = 12$

↳ $T_2 = [1 \quad 6 \quad 2 \quad 0]$, $r_2 = 12$

We do same for $i = 3, 4$. System is:

$T\vec{s} = \vec{r}$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 6 & 2 & 0 \\ 0 & 1 & 4 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ s_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 12 \\ -6 \\ -1 \end{bmatrix}$$

Solve for s_i's + plug into Hermite closed form to get coeffs a_i, b_i, c_i, d_i for each $S_i(x)$ polynomial (cubic).

- using new system for s_i's, matrix size is $n \times n$
  - ↳ 1 eqn per node
  - ↳ old matrix had 4 coeffs each for n-1 intervals so size is $(4n-4) \times (4n-4)$
  - ↳ system is smaller by constant factor of 4
- new matrix is tridiagonal
  - ↳ i.e. only entries on diagonal + its 2 neighbours are ever non-zero
- e.g.

Give the conditions that should be satisfied for the piecewise function $S(x)$ to be a valid cubic spline, assuming we already know:

$$S(x) = \begin{cases} \frac{5}{3} + \frac{16}{3}x + ax^2 + x^3 & \text{on } [-1, 1], \\ -\frac{7}{3} + bx + \frac{22}{3}x^2 + \frac{2}{3}x^3 & \text{on } [1, 2], \end{cases}$$

Is there a choice of a and b to make $S(x)$ a valid cubic spline?

↳ only need to check interpolating + derivative conds hold at $x = 1$ since no BCs stated

$S_1(1) = S_2(1)$

$\frac{5}{3} + \frac{16}{3}(1) + a + 1 = -\frac{7}{3} + b + \frac{22}{3} + \frac{2}{3}$ (1)

$S_1'(x) = \frac{16}{3} + 2ax + 3x^2$

$S_2'(x) = b + \frac{44}{3}x + 2x^2$

$S_1'(1) = S_2'(1)$

$\frac{16}{3} + 2a + 3 = b + \frac{44}{3} + 2$ (2)

$S_1''(x) = 2a + 6x$

$S_2''(x) = \frac{44}{3} + 4x$

$S_1''(1) = S_2''(1)$

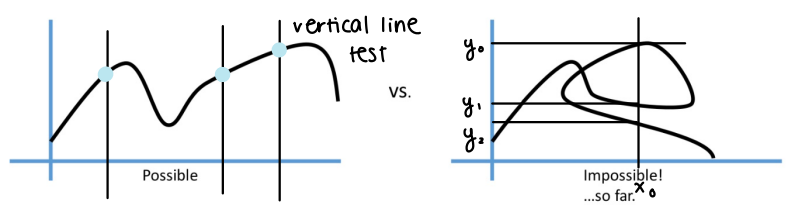
$2a + 6x = \frac{44}{3} + 4x$ (3)

↳ no soln satisfying 1, 2, 3 for both a, b so this can nvr be valid cubic spline

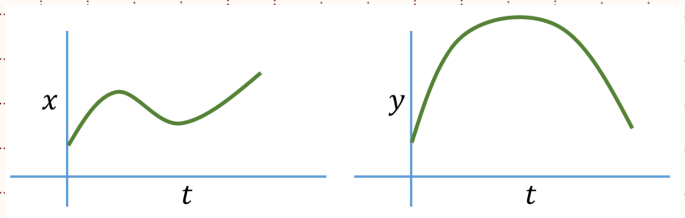
PARAMETRIC CURVES

date 02/06/2025

- only handled fcns $y = p(x)$ so shortcoming is that we can't have curve that folds back on itself
  - ↳ e.g.



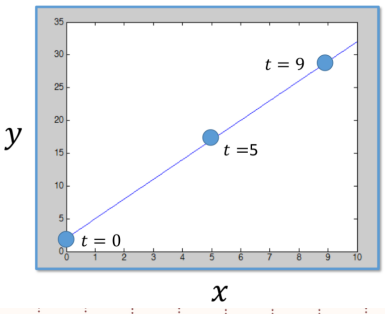
  - ° (x_0, y_0), (x_0, y_1), (x_0, y_2) can't be interpolated as fcn of 1 var
  - ↳ x uniquely dictates y so 2 distinct pts can't have same x coord
- parametric curves will handle more general curves
  - ↳ let x, y be separate fcns of new param t
  - ↳ pt's position given by vector $\vec{P(t)} = (x(t), y(t))$
  - ↳ t inc. monotonically along curve but x, y may inc/dec as needed to describe any shape
    - ° e.g. t is time so object's coords (x, y) change as time passes
  - ↳ e.g.



  - ↳ notion isn't specific to any type of curve or interpolating fcn (e.g. piecewise linear, Hermite, cubic splines, Bezier)
  - ↳ curve is parametrized by t when (x, y) position in curve is dictated by t
- e.g. line

The simple line $y = 3x + 2$ can equivalently be described by the two coordinate functions:

$$x(t) = t,$$
$$y(t) = 3t + 2$$



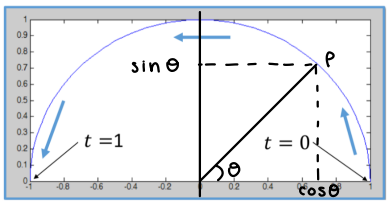
- e.g. semi-circle

Consider a curve along a semi-circle in the upper half plane, oriented from $(1, 0)$ to $(-1, 0)$.

The usual *implicit* equation for a unit circle is $x^2 + y^2 = 1$.

One parametric form is:

$$x(t) = \cos(\pi t)$$
$$y(t) = \sin(\pi t)$$
$$\text{for } \; 0 \le t \le 1.$$

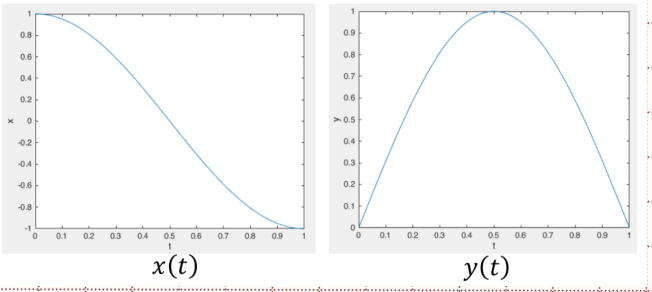


date

$p = (\cos\theta, \sin\theta)$, $0 \le \theta \le \pi$

↳ if $\theta = \pi t$, where $0 \le t \le 1$, we get all the pts

↳ $x(t)$, $y(t)$ create semi-circle tgt



$x(t)$ $y(t)$

given curve can be parameterized in diff ways while yielding same shape

↳ e.g.

1) $x(t) = \cos(\pi t)$, $y = \sin(\pi t)$ for $0 \le t \le 1$

2) $x(t) = \cos(\pi(1-t))$, $y = \sin(\pi(1-t))$ for $0 \le t \le 1$

- 2 traverses curve in opposite dir (left to right) as t goes from 0 to 1

↳ parametric curves are **non-unique**

- 2 parameterizations can also traverse curve in same dir, but **diff speeds/rates**

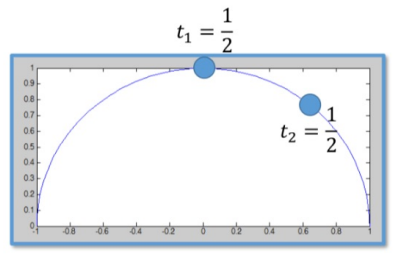
↳ e.g.

1) $x_1(t) = \cos(\pi t)$, $y_1(t) = \sin(\pi t)$ for $0 \le t \le 1$

2) $x_2(t) = \cos(\pi t^2)$, $y_2(t) = \sin(\pi t^2)$ for $0 \le t \le 1$

- 2 covers same curve but returns diff pts for any given val of t

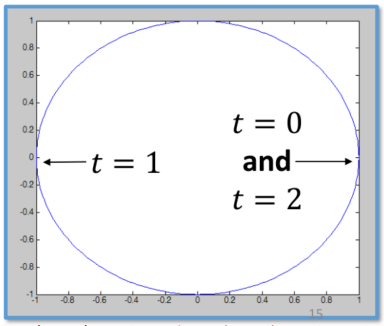
→ e.g.



- parametric curves allow multiple pts to have same x or y coord

↳ e.g. full circle

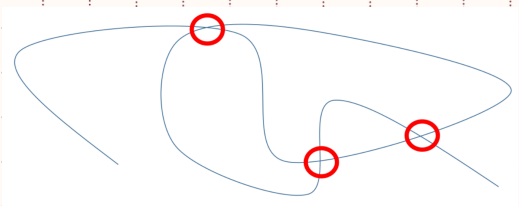
$x(t) = \cos(\pi t)$, $y(t) = \sin(\pi t)$ for $0 \le t < 2$



- for $t \ge 2$, circle continues to wrap back over itself

- curves w/ self-intersections are supported easily

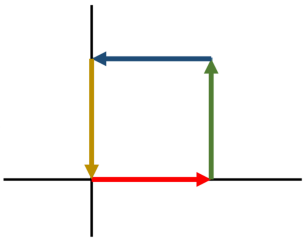
↳ e.g.



· e.g. piecewise parametric curves : unit square

e.g. The unit square:

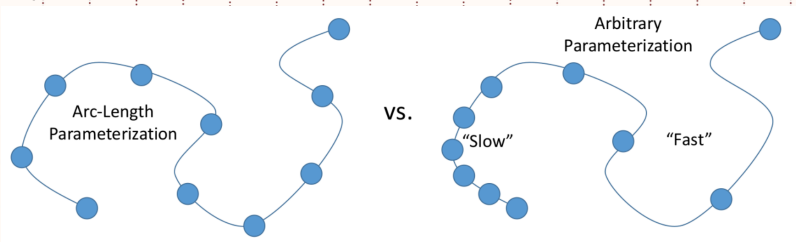
1. $x(t) = t, y(t) = 0, 0 \le t \le 1$
2. $x(t) = 1, y(t) = t - 1, 1 \le t \le 2$
3. $x(t) = 1 - (t - 2), y(t) = 1, 2 \le t \le 3$
4. $x(t) = 0, y(t) = 1 - (t - 3), 3 \le t \le 4$



· common parameterization is to choose t as distance along curve

↳ gives unit speed traversal (i.e. travel 1 unit of distance in 1 unit of time)

↳ e.g. blue circles evenly spaced in time t



· combine existing interpolant types w/parametric curves by considering x(t), y(t) separately

↳ use 2 Lagrange polys (x(t), y(t)) to fit small set of (t_i, x_i, y_i) pt data

↳ use Hermite interpolation for x(t), y(t) given $(t_i, x_i, y_i, sx_i, sy_i)$ pt/derivative data

↳ fit separate cubic splines to x(t), y(t) given many pts

· given ordered (x_i, y_i) pt data, need data for t to form (t_i, x_i), (t_i, y_i) pairs to fit curves to t form parameterization

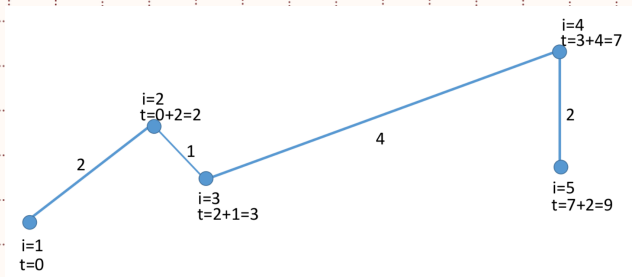
↳ option 1: use node index as parameterization

- i.e. $t_i = i$ at each node

↳ option 2: apx arc-length parameterization

- set $t_1 = 0$ at 1st node
- recursively compute $t_{i+1} = t_i + \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$

  distance btwn (x_i, y_i), (x_{i+1}, y_{i+1})
- e.g.



ODEs & forward euler time-stepping

date 02/08/2025

ORDINARY DIFFERENTIAL EQUATIONS

- general form of ODE: $F(x, y(x), y'(x), y''(x), \ldots, y^{(n)}(x)) = 0$
  - ↳ $y^{(i)}(x)$ is i^{th} derivative of $y(x)$
- order of ODE is highest order of derivatives
- closed form often isn't avail in practice
  - ↳ instead, may know rlmship btwn y (depends on t), y' (wrt t) given by known fcn f
    - $y'(t) = f(t, y(t))$
    - e.g. $v'(t) = F(t)/m$
  - ↳ e.g. temp T in house at t is $T(t) = 21 + 2\cos t$, which is not differential eqn
- often need to construct/define mathematical model of problem
  - ↳ all models are wrong, but some are useful
  - ↳ always tradeoffs in designing models
    - e.g. accuracy vs cost vs complexity
- e.g. simple population model

Consider a mouse population, $y(t)$, over time. With enough food, we'll say the population changes as

$$y'(t) = a.y(t)$$

where a is some constant (reproduction rate).

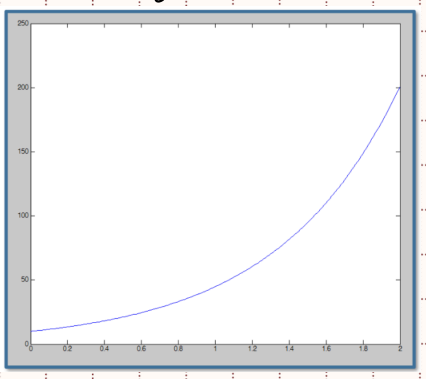
i.e., # of mice born per unit time, $y'(t)$, is a constant, a, times the current number of mice, $y(t)$.

$y(t)$ **is not given explicitly (in closed form)!**

- ↳ w/ initial pop of $y_0 = y(t_0)$, closed-form soln is $y(t) = y_0 e^{a(t-t_0)}$
- ↳ verification:

$$\begin{aligned} y'(t) &= \frac{d}{dt}\left(y_0 e^{a(t-t_0)}\right) \\ &= y_0 e^{a(t-t_0)} \cdot a \\ &= y(t) \cdot a \end{aligned}$$

- ↳ exponential growth (not realistic)



- e.g. more complex model

In reality, food supplies (and space and partners and ...) are limited.

Consider a new model,

$$y'(t) = y(t).(a - b.y(t))$$

where the b term expresses the effect of resource limits.

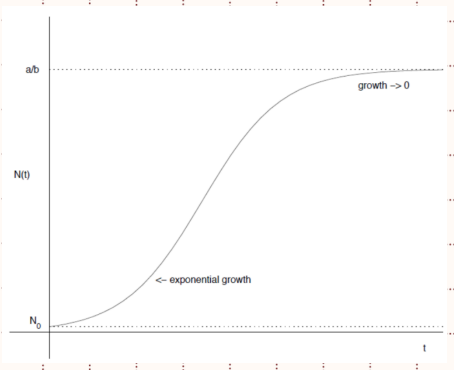
date

↳ for small $y(t)$, $y'(t) \simeq a \cdot y(t)$
- exponential growth

↳ for $y(t) \simeq \frac{a}{b}$, $y'(t) \simeq 0$
- pop growth levels off

↳ closed-form soln: $y(t) = \frac{a y_0 e^{a(t-t_0)}}{b y_0 e^{a(t-t_0)} + (a - y_0 b)}$

↳ logistic growth



- e.g. even more complex model

What other real-world factors might affect population growth?

- Seasonal variation in birth rate.
- Seasonal variation in food supply.
- Ratio of males to females.
- Presence and population of predators.
- Many, **many** others ... (and no model can perfectly capture all possible factors!)

A very slightly more complex model is:

$$y'(t) = y(t).\Big(a(t) - b(t).y(t)^{\alpha}\Big)$$

↳ $a(t)$, $b(t)$ are fcns of time

↳ α is constant

↳ no general closed-form soln

↳ using numerical models, can still make meaningful predictions for it

- since even fairly simple mathematical models lack closed-form solns, we'll develop numerical methods to find apx / partial solns

INITIAL VALUE PROBLEM

- general form of differential eqn: $y'(t) = f(t, y(t))$

↳ f is specified

↳ initial vals are $y(t_0) = y_0$

↳ e.g.

In our population example, we had $y'(t) = a \cdot y(t)$, with some initial population at time $t_0 = 0$, of say $y_0 = 100$ mice.

- dynamics fcn is $f(t, y(t)) = a \cdot y(t)$
- initial cond is $y_0 = 100$

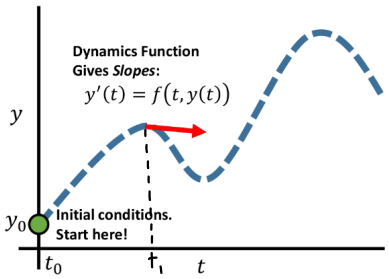
date

· e.g. visualizing IVP

Given:

- Initial value, y_0 at $t = t_0$.
- Dynamics function, $y' = f(t, y(t))$

Problem: Can we (approximately) find the value at later times?



· 2 possible complications for ODE problems, $y'(t) = f(t, y(t))$:

1) systems of differential eqns : involving more than 1 unknown var (i.e. not just y)
2) higher order differential eqns : involving higher derivatives (i.e. not just y')

· to deal w/ systems of differential eqns, consider model w/ multiple vars that interact

↳ e.g. x, y coords of moving object

↳ gives system of differential eqns like:

$$x'(t) = f_x(t, x(t), y(t)) \quad w/\ x(t_0) = x_0$$
$$y'(t) = f_y(t, x(t), y(t)) \quad w/\ y(t_0) = y_0$$

2 initial conds

2 dynamics fcns

· system of 2 eqns can be written in vector form:

$$x_1'(t) = f_1(t, x_1(t), x_2(t)) \quad w/ x_1(t_0) = a$$
$$x_2'(t) = f_2(t, x_1(t), x_2(t)) \quad w/ x_2(t_0) = b$$

$$\begin{bmatrix} x_1'(t) \\ x_2'(t) \end{bmatrix} = \begin{bmatrix} f_1(t, x_1(t), x_2(t)) \\ f_2(t, x_1(t), x_2(t)) \end{bmatrix} \quad w/ \begin{bmatrix} x_1(t_0) \\ x_2(t_0) \end{bmatrix} = \begin{bmatrix} a \\ b \end{bmatrix}$$

↳ vector notation: $\vec{x}'(t) = \vec{f}(t, \vec{x}(t))$ w/ $\vec{x}(t_0) = \vec{x}_0$

↳ matrix notation is helpful when f_1, f_2 are linear b/c $\vec{f}$ can be decomposed as constant matrix · vector containing fcns

° e.g.

$$x_1'(t) = 5x_1(t) + 7x_2(t)$$
$$x_2'(t) = 3x_1(t) - 5x_2(t)$$

We have:

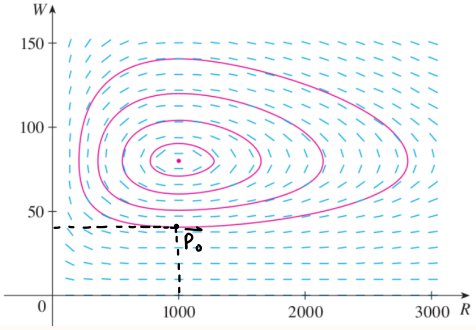
$$\begin{bmatrix} x_1' \\ x_2' \end{bmatrix} = \begin{bmatrix} 5x_1 + 7x_2 \\ 3x_1 - 5x_2 \end{bmatrix} = \begin{bmatrix} 5 & 7 \\ 3 & -5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

· e.g. predator-prey systems

$$R'(t) = k \cdot R(t) - a \cdot R(t) \cdot W(t)$$
$$W'(t) = -r \cdot W(t) + b \cdot R(t) \cdot W(t)$$

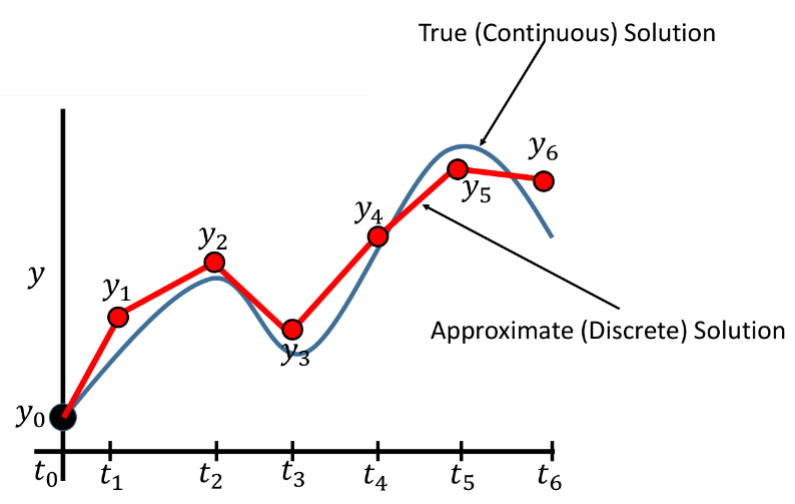
- $R(t)$: # of rabbits at time t
- $W(t)$: # of wolves at time t
- a, b, k and r are positive constants

We are looking for $\vec{P}(t) = (R(t), W(t))$, with an initial position, $\vec{P}_0 = (1000, 40)$

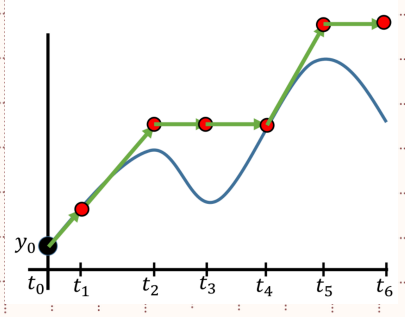


date

- to deal w/ higher order differential eqns, start by considering 1st order ODEs, $y'(t) = f(t, y(t))$
  - ↳ numerical soln: discrete set of time-val pairs (t_i, y_i)
    - y_i should apx true val $y(t_i)$
    - e.g.

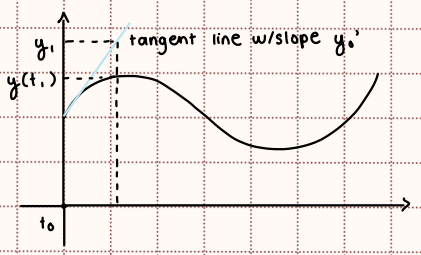


- time-stepping: given initial conds, repeatedly step sequentially fwd to next time instant, using derivative info y' + timestep h (gives length of step)
  - ↳ set $n = 0$, $t = t_0$, $y = y_0$
  - ↳ steps:
    1) compute y_{n+1}
       → key step
    2) increment time: $t_{n+1} = t_n + h$
    3) advance: $n = n + 1$
    4) repeat
  - ↳ many methods exist + we must consider several categories:
    - single-step vs multistep: only use info from current time or previous timesteps too
    - explicit vs implicit: is y_{n+1} given as explicit fcn to eval or do we need to solve implicit eqn?
    - timestep size: use constant h or allow it to vary
- fwd Euler is explicit, single-step scheme
  1) compute current slope: $y_n' = f(t_n, y_n)$
  2) step in straight line w/ that slope: $y_{n+1} = y_n + h \cdot y_n'$
  3) repeat
  - ↳ e.g.



date

↳ e.g.



$y_0' = y'(t_0) = f(t_0, y_0)$

Tangent line:

$y - y_0 = y_0'(t - t_0)$

General form of line: $y - y_0 = m(x - x_0)$

↓ take $t = t_1$

$y_1 - y_0 = y_0'(t_1 - t_0)$ ($t_1 - t_0 = h$)

$y_1 = y_0 + hy_0'$

y_1 is apx of $y_1(t)$

At t_n, we have value y_n so our pt is (t_n, y_n) + slope at tangent line is $y_n' = f(t_n, y_n)$.

Tangent line:

$y - y_n = y_n'(t - t_n)$

At t_{n+1}:

$y_{n+1} - y_n = y_n'(t_{n+1} - t_n)$

$y_{n+1} = y_n + y_n' h$

Keep repeating process.

↳ fwd Euler scheme is $y_{n+1} = y_n + hf(t_n, y_n)$ (y_n')

- time-stepping applies recurrence rlm to apx fcn vals at later + later times
- e.g.

Consider the simple IVP $y'(t) = 2y(t)$, with initial conditions at $t_0 = 1$ of $y(t_0) = 3$.

1. Write down the recurrence for Forward Euler, with step size $h = 1$.
2. Use forward Euler to approximate y at time $t = 5$.
3. Compare against the true solution, $y(t) = 3e^{2(t-t_0)}$.
4. Repeat for $h = 1/2$.

$y'(t) = 2y(t)$ (dynamics fcn) $\qquad y(t_0) = y_0 = 3$ (initial cond)

Fwd Euler

$y_{n+1} = y_n + hf(t_n, y_n)$

$= y_n + 1 \cdot 2y_n$

$= 3y_n$

$y_0 = 3, \; y_1 = 9, \; \ldots$

Note that $y(t) = 3e^{2(t-t_0)}$, so:

$t = 1 \rightarrow y(1) = 3e^{2(1-1)} = 3$

date

$t = 2 \to y(2) = 3e^{2(2-1)} = 3e^2 \approx 22.2$

↳ numerical soln is not close to exact soln

Try $h = \frac{1}{2}$:

FE: $y_{n+1} = y_n + \frac{1}{2} \cdot 2y_n$

$= 2y_n$

↳ for $t = 4$, we get slightly more accurate ans

Discrete data is indexed by ints n: t_n, y_n, x_n, etc.

↳ $y_n(t)$ is exact val of true fcn y at time t

↳ y_n is apx / discrete data at step n (i.e. time t)

° $y_n \approx y(t_n)$

- h = 1

n	t_n	y_n	$y(t_n)$
0	1	3	3
1	2	9	22.2
2	3	27	163.8
3	4	81	1210
4	5	243	8943

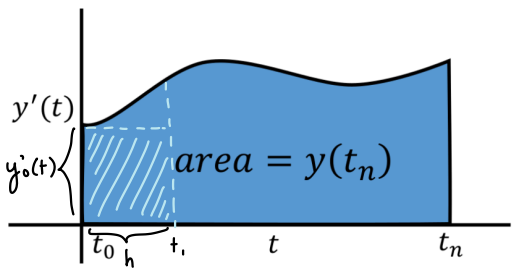
- h = 0.5

n	t_n	y_n	$y(t_n)$
0	1	3	3
1	1.5	6	8.15
2	2	12	22.2
3	2.5	24	60.3
4	3	48	163.8
5	3.5	96	445
6	4	192	1210

· time-stepping (aka time-integration) is when we integrate over time to apx y from y'

↳ plotting $y'(t) = f(t, y(t))$, time-stepping is apxing area under curve

° e.g.



$h = t_1 - t_0$

$y_n = y_0$

$y_{n+1} = y_1$

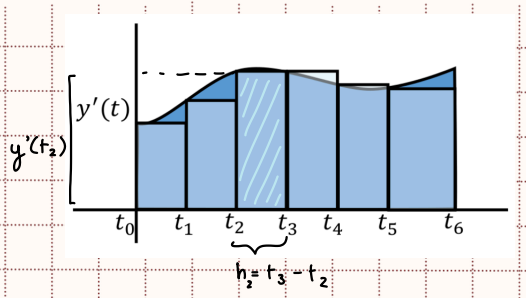
FE: $y_1 = y_0 + hf(t_0, y_0)$

$= y_0 + \underbrace{h \cdot y_0'(t_0)}_{\text{area of rectangle on } [t_0, t_1]}$

↳ FE apx area using rectangles of area $h_i y'(t_i)$ w/ height $y'(t) = f(t, y(t))$ evaled at left side (i.e. current time)

° better time-stepping schemes correspond to better apxs of integral

° e.g.



Area of 3rd rectangle = $h_2\, y'(t_2)$

HIGHER ORDER TIME-STEPPING SCHEMES

date 02/09/2025

FORWARDS EULER

- for systems of 1st order ODEs, apply FE in each row in same way
  - ↳ e.g.

$$\begin{bmatrix} y'(t) \\ z'(t) \end{bmatrix} = \begin{bmatrix} z(t) \\ tz(t) - ay(t) + \sin(t) \end{bmatrix} \quad with \quad \begin{bmatrix} y(0) \\ z(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

Applying FE:

$$\begin{bmatrix} y_{n+1} \\ z_{n+1} \end{bmatrix} = \begin{bmatrix} y_n \\ z_n \end{bmatrix} + \begin{bmatrix} z_n \\ t_n z_n - a y_n + \sin(t_n) \end{bmatrix} \quad w/ \begin{bmatrix} y_0 \\ z_0 \end{bmatrix} = \begin{bmatrix} 1 \\ 3 \end{bmatrix}$$

- e.g.

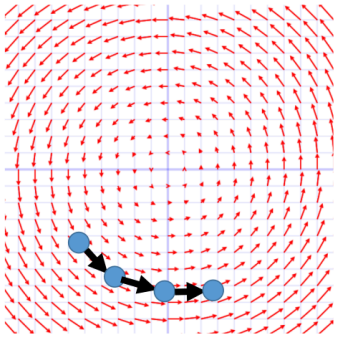
Consider a particle with coordinates $(x(t), y(t))$ satisfying the ODE system

$$x'(t) = -y(t)$$
$$y'(t) = x(t)$$

with

$$x(t_0) = 2, \quad y(t_0) = 0, \quad \text{and} \quad t_0 = 2.$$

1. Write down the recurrence for Forward Euler applied to this problem, with time step $h = 2$.
2. Apply forward Euler up to $t = 6$.



A particle moving in a rotational wind field.

$$\begin{bmatrix} x'(t) \\ y'(t) \end{bmatrix} = \begin{bmatrix} -y(t) \\ x(t) \end{bmatrix}$$

Apply FE:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + h \begin{bmatrix} -y_n \\ x_n \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + 2 \begin{bmatrix} -y_n \\ x_n \end{bmatrix} = \begin{bmatrix} x_n - 2y_n \\ y_n + 2x_n \end{bmatrix}$$

Step 1:

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 2 - 2(0) \\ 0 + 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

Step 2:

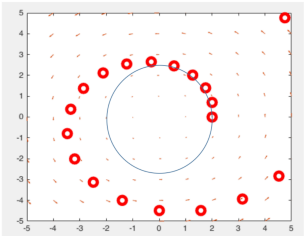
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 2 - 2(4) \\ 4 + 2(2) \end{bmatrix} = \begin{bmatrix} -6 \\ 8 \end{bmatrix}$$

Step 3:

$$\begin{bmatrix} x_3 \\ y_3 \end{bmatrix} = \begin{bmatrix} -6 - 2(8) \\ 8 + 2(-6) \end{bmatrix} = \begin{bmatrix} -22 \\ -4 \end{bmatrix}$$

n	t_n	x_n	y_n
0	2	2	0
1	4	2	4
2	6	-6	8
3	8	-22	-4

↳ red dots are numerical soln using FE + blue circle is correct ans



- FE exhibits large error / drift

- Taylor series apx a fcn in some neighbourhood using infinite weighted sum of its

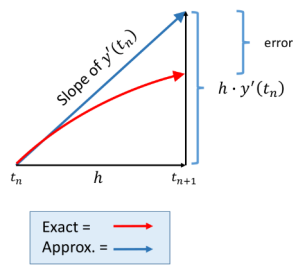
date

derivatives
↳ including more terms in series gives better estimates of fcn val
↳ $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$

· 2 ways to derive FE:
↳ finite diff view: apx derivative y' w/ finite diff
• ODE form: $y'(t) = f(t, y(t))$
• finite diff apx of y' gives $y'(t_n) \approx \frac{y_{n+1} - y_n}{t_{n+1} - t_n} = \frac{y_{n+1} - y_n}{h} = f(t_n, y_n)$
• rearranging gives FE: $y_{n+1} = y_n + hf(t_n, y_n)$
↳ Taylor series view: use Taylor series + drop/ truncate higher order terms
• std Taylor series is $f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \dots$
• want to apx $y(t_{n+1})$ from t_n data
→ $f \Rightarrow y$
→ $a \Rightarrow t_n$
→ $x \Rightarrow t_{n+1}$
• we get: $y(t_{n+1}) = y(t_n) + y'(t_n)(t_{n+1} - t_n) + \frac{y''(t_n)}{2}(t_{n+1} - t_n)^2 + \frac{y'''(t_n)}{6}(t_{n+1} - t_n)^3 + \dots$
$= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \dots$
→ $h = t_{n+1} - t_n$
• assume order h^2 + higher will be small as $h \to 0$ so we drop them
• replace y' w/ eval of dynamics fcn f
• FE: $y(t_{n+1}) \approx y_{n+1} = y_n + hf(t_n, y_n)$

ERROR OF TIME-STEPPING SCHEMES

· absolute/global error at step n is $|y_n - y(t_n)|$
↳ can't measure error exactly w/o knowing $y(t)$ (i.e. exact val from unknown closed-form soln)
• rely on Taylor series
· FE makes linear apx at each step, incurring local error
↳ smaller step size h → more frequent slope estimates → less error
↳ e.g.



↳ characterizing error:
• FE is $y_{n+1} = y_n + hf(t_n, y_n)$
• Taylor series is $y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + \dots$
• error for 1 step is diff btwn above 2 eqns, assuming we have exact data at t_n

date

(i.e. $y_n = y(t_n)$)

→ $y_{n+1} = y(t_n) + hf(t_n, y(t_n))$

$= y(t_n) + hy'(t_n)$

- difference: $y_{n+1} - y(t_{n+1}) = -\frac{h}{2} y''(t_n) + O(h^3)$

$= O(h^2)$

Big O used to emphasize that asymptotically, dominant term is slowest dec as $h \to 0$

→ local truncation error (LTE) of FE

→ as h dec., LTE dec quadratically

TRAPEZOIDAL RULE

- can also derive FE error using Taylor expansion
  - ↳ $y(t_{n+1}) = y(t_n) + hy'(t_n) + O(h^2)$

    $y'(t_n) = \frac{y(t_{n+1}) - y(t_n)}{h} + O(h)$
  - ↳ error in derivative apx is $O(h)$
- keep more terms in Taylor series so error is higher order (i.e. smaller)
  - ↳ $y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{6} y'''(t_n) + \dots$
  - ↳ problem is that we don't know y''
    - might be hard/costly to find exactly
  - ↳ ODE only gives 1st derivative $y'(t) = f(t, y)$ so we want to treat f like black box
  - ↳ soln is to use finite diff to apx y''
    - $y''(t_n) = \frac{y'(t_{n+1}) - y'(t_n)}{h} + O(h)$
    - plug into Taylor series:

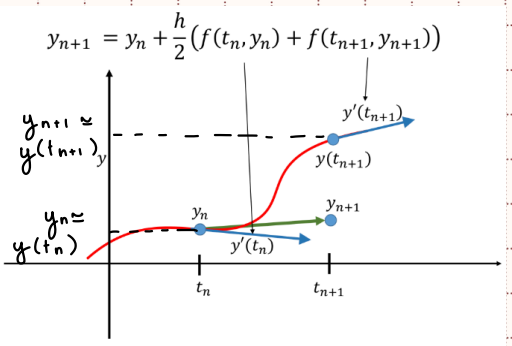
$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}\left(\frac{y'(t_{n+1}) - y'(t_n)}{h} + O(h)\right) + O(h^3)$$

$$= y(t_n) + hy'(t_n) + \frac{h}{2}(y'(t_{n+1}) - y'(t_n)) + O(h^3)$$

$$= y(t_n) + \frac{h}{2}(y'(t_{n+1}) + y'(t_n)) + O(h^3)$$

$$y(t_{n+1}) = y(t_n) + \underbrace{\frac{h}{2}(f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n)))}_{\text{trapezoidal}} + \underbrace{O(h^3)}_{\text{LTE}}$$

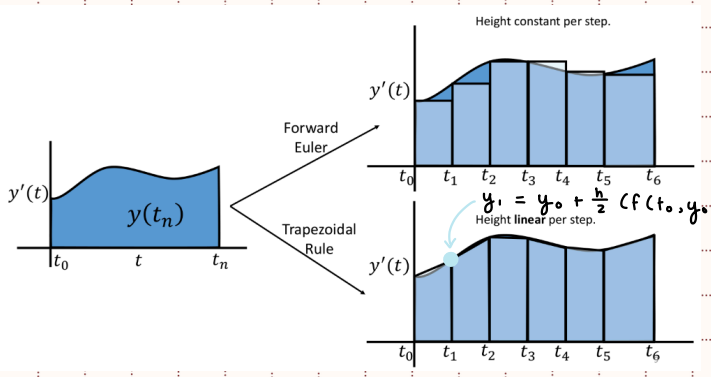
  - ↳ trapezoidal rule/scheme is $y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(t_{n+1}, y(t_{n+1})) + f(t_n, y(t_n))) + O(h^3)$
    - LTE is $O(h^3)$
    - reducing h reduces per-step error cubically in h
- geometric intuition for trapezoidal rule is to eval slope $y' = f$ at start + end of time step, then step along avg slope
  - ↳ e.g.

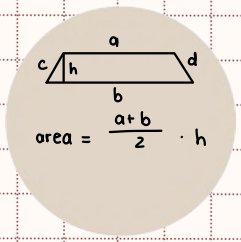


date

- avg slope = $\frac{f(t_n, y_n) + f(t_{n+1}, y_{n+1})}{2}$

- FE vs trapezoidal rule in area integration view:

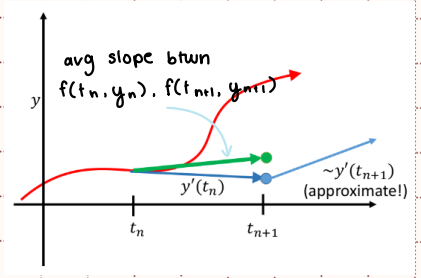




↳ trapezoidal: $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$

gives area of trapezoids under curve

- FE is explicit scheme
  ↳ $y_{n+1} = y_n + hf(t_n, y_n)$
  unknown — known — computable based on known vals t_n, y_n
  ↳ RHS only involves known quantities so can plug in + eval
- trapezoidal is implicit scheme
  ↳ $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$
  unknown — known — computable based on known vals t_n, y_n — known — unknown
  ↳ RHS involves quantities from currently unknown time t_{n+1}
- deriving another scheme that avoids implicitness of trapezoidal (i.e. explicit trapezoidal):
  ↳ take FE step to estimate end pt
  ↳ eval slope there
  ↳ use apx end-of-step slope in trapezoidal formula
  ↳ e.g.



Single-var Taylor series:
$f(x+h) = f(x) + h\frac{\partial f}{\partial x} + O(h^2)$
Multivar Taylor series:
↳ can expand in 1 var:
$f(x, y+h_y) = f(x,y) + h_y\frac{\partial f}{\partial y} + O(h_y^2)$
↳ can expand in multiple vars:
$f(x+h_x, y+h_y) = f(x,y) + h_x\frac{\partial f}{\partial x} + h_y\frac{\partial f}{\partial y} + O(h_x^2) + O(h_y^2)$

- modified / improved Euler method is explicit scheme + has LTE $O(h^3)$
  ↳ $y^*_{n+1} = y_n + hf(t_n, y_n)$ ← FE
  ↳ $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y^*_{n+1}))$ ← explicit trapezoidal
  ↳ to derive:
  Trapezoidal: $y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1})) + O(h^3)$
  FE: $y(t_{n+1}) = y(t_n) + hf(t_n, y(t_n)) + O(h^2)$

date

Looking at $y^{\sim}_{n+1} = y(t_n) + hf(t_n, y(t_n))$:

↳ $y(t_{n+1}) - y^{\sim}_{n+1} = O(h^2)$

↳ error is $O(h^2)$

Looking at $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y^{\sim}_{n+1}))$:

↳ Taylor expand f at $y^{\sim}_{n+1}$ gives:

$$f(t_{n+1}, y(t_{n+1})) = f(t_{n+1}, y^{\sim}_{n+1}) + \frac{\partial f}{\partial y}(t_{n+1}, y^{\sim}_{n+1}) \cdot \underbrace{(y(t_{n+1}) - y^{\sim}_{n+1})}_{O(h^2)} + \underbrace{O((y(t_{n+1}) - y^{\sim}_{n+1})^2)}_{O(h^4)}$$

∘ $h = y(t_{n+1}) - y^{\sim}_{n+1}$

↳ we see that $f(t_{n+1}, y(t_{n+1})) = f(t_{n+1}, y^{\sim}_{n+1}) + O(h^2)$

↳ using this apx end-of-step in trapezoidal expr gives:

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(t_n, y(t_n)) + f(t_{n+1}, y^{\sim}_{n+1}) + O(h^2)) + O(h^3)$$

$$y(t_{n+1}) = y(t_n) + \frac{h}{2}(f(t_n, y(t_n)) + f(t_{n+1}, y^{\sim}_{n+1})) + O(h^3)$$

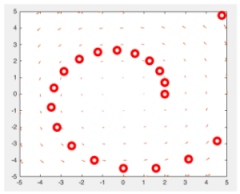
↳ LTE is $O(h^3)$

∘ same as trapezoidal, but diff error coeffs

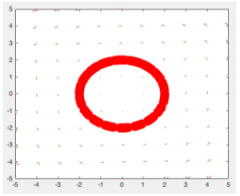
ODEs : more schemes

date 02/11/2025

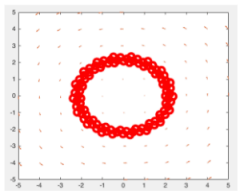
- categorizing explicit vs implicit data:
  - ↳ explicit : if only y_n or earlier step data on RHS
    - FE : $y_{n+1} = y_n + h f(t_n, y_n)$
    - improved Euler : $y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, \overbrace{y_n + h f(t_n, y_n)}^{y^*_{n+1}})\right)$
    - simpler + faster to compute per step
    - less stable
      - → requires smaller timesteps to avoid blowing up
  - ↳ implicit : unknown y_{n+1} appears on both sides
    - trapezoidal : $y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y_{n+1})\right)$
    - often more complex + expensive to solve per step
    - more stable
      - → can safely use larger timesteps
  - ↳ visualization :



FE



trapezoidal



improved Euler

e.g.

We previously applied F.E. to the ODE system:

$$x'(t) = -y(t)$$
$$y'(t) = x(t)$$

with initial conditions $x(t_0) = 2$, $y(t_0) = 0$, $t_0 = 0$.

1. Apply improved Euler with time step size $h = 2$ to find x, y at $t = 4$.
2. What are the time-stepping equations if we apply trapezoidal to this problem?

1) $y^*_{n+1} = y_n + h f(t_n, y_n)$

$y_{n+1} = y_n + \frac{h}{2}\left(f(t_n, y_n) + f(t_{n+1}, y^*_{n+1})\right)$

FE step:

$$\begin{bmatrix} x^*_{n+1} \\ y^*_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + 2\begin{bmatrix} -y_n \\ x_n \end{bmatrix} = \begin{bmatrix} x_n - 2y_n \\ y_n + 2x_n \end{bmatrix}$$

date

Trapezoidal step:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n \\ y_n \end{bmatrix} + \frac{2}{2}\begin{bmatrix} -y_n - y^*_{n+1} \\ x_n + x^*_{n+1} \end{bmatrix}$$

$$= \begin{bmatrix} x_n - y_n - y^*_{n+1} \\ y_n + x_n + x^*_{n+1} \end{bmatrix}$$

Apply to IVP:

$$\begin{bmatrix} x_1^* \\ y_1^* \end{bmatrix} = \begin{bmatrix} x_0 - 2y_0 \\ y_0 + 2x_0 \end{bmatrix} = \begin{bmatrix} 2 - 2(0) \\ 0 + 2(2) \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} x_0 - y_0 - y_1^* \\ y_0 + x_0 + x_1^* \end{bmatrix} = \begin{bmatrix} 2 - 0 - 4 \\ 0 + 2 + 2 \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} x_2^* \\ y_2^* \end{bmatrix} = \begin{bmatrix} x_1 - 2y_1 \\ y_1 + 2x_1 \end{bmatrix} = \begin{bmatrix} -2 - 2(4) \\ 4 + 2(-2) \end{bmatrix} = \begin{bmatrix} -10 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} x_1 - y_1 - y_2^* \\ y_1 + x_1 + x_2^* \end{bmatrix} = \begin{bmatrix} -2 - 4 - 0 \\ 4 - 2 - 10 \end{bmatrix} = \begin{bmatrix} -6 \\ -8 \end{bmatrix}$$

n	t_n	x_n	y_n	x_n^*	y_n^*
0	0	2	0	–	–
1	2	-2	4	2	4
2	4	-6	-8	-10	0

Ans: $x = -6, y = -8$ at $t = 4$

2) $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$

$x_{n+1} = x_n + \frac{2}{2}(-y_n - y_{n+1})$

$= x_n - y_n - y_{n+1}$

$y_{n+1} = y_n + \frac{2}{2}(x_n + x_{n+1})$

$= y_n + x_n + x_{n+1}$

Linear system of eqns per time step:

$$\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n - y_n - y_{n+1} \\ y_n + x_n + x_{n+1} \end{bmatrix}$$

$$\begin{bmatrix} x_{n+1} + y_{n+1} \\ -x_{n+1} + y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n - y_n \\ x_n + y_n \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}\begin{bmatrix} x_{n+1} \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} x_n - y_n \\ x_n + y_n \end{bmatrix}$$

$$A\vec{x} = \vec{b}$$

↳ for other ODE problems, system could be nonlinear (i.e. more costly to solve) so we won't consider them

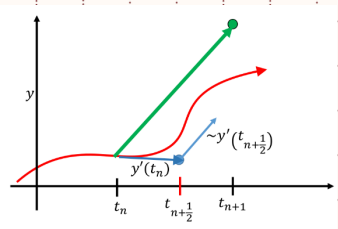
- bwd / implicit Euler method uses slope from only end of step
  - ↳ $y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$
  - ↳ LTE is $O(h^2)$
- Runge Kutta is a family of explicit schemes
  - ↳ improved Euler rewritten:
    - $k_1 = hf(t_n, y_n)$
    - $k_2 = hf(t_n + h, y_n + k_1)$
    - $y_{n+1} = y_n + \frac{k_1}{2} + \frac{k_2}{2}$
  - ↳ explicit midpoint method:
    - $k_1 = hf(t_n, y_n)$
    - $k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})$

$t_{n+1} - t_n = h$

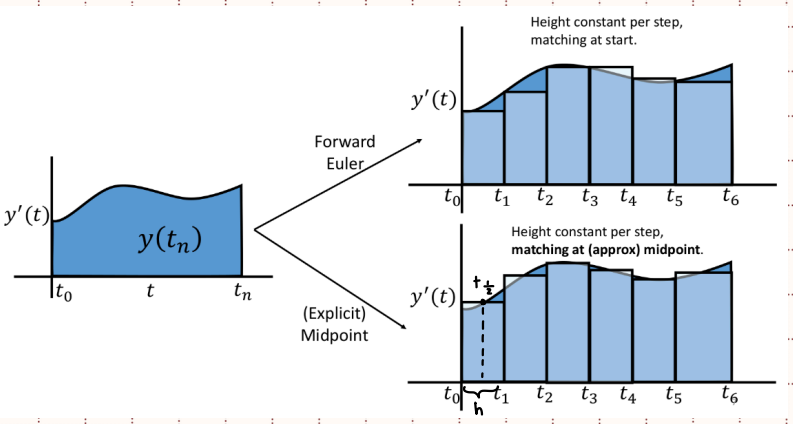
Reminder $y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_n, y^*_{n+1}))$

date

- $y_{n+1} = y_n + \frac{k}{2}$
- geometric intuition:
  1) take FE step to halfway pt in time
  2) eval slope
  3) use that slope to take full step from start
  → e.g.



- equiv exprs:
  → $y^*_{n+\frac{1}{2}} = y_n + \frac{h}{2} f(t_n, y_n) = y_n + \frac{k_1}{2}$
  → $y_{n+1} = y_n + hf(t_n + \frac{h}{2}, y^*_{n+\frac{1}{2}}) = y_n + k_2$
- area integration overview:



↳ similar schemes exist for higher orders $O(h^\alpha)$ for $\alpha = 3, 4, 5, \ldots$
- e.g. classical Runge-Kutta (RK4) has LTE of $O(h^5)$
  → $k_1 = hf(t_n, y_n)$
  → $k_2 = hf(t_n + \frac{h}{2}, y_n + \frac{k_1}{2})$
  → $k_3 = hf(t_n + \frac{h}{2}, y_n + \frac{k_2}{2})$
  → $k_4 = hf(t_n + h, y_n + k_3)$
  → $y_{n+1} = y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

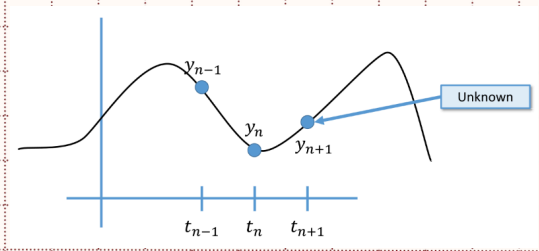
· local / truncation error: order of error for single step
· global error: total error accumulated at final time t_{final}
  ↳ usually more important
  ↳ for constant h, $\#steps = \frac{t_{final} - t_0}{h} = O(h^{-1})$
  ↳ for given method, we have roughly global error $\leq$ (local error) $O(h^{-1})$
  - i.e. 1 deg lower
  - note that LTE $\neq$ #steps
  - e.g.
    → FE: $O(h^2) \cdot O(h^{-1}) = O(h)$ is first-order accurate

date

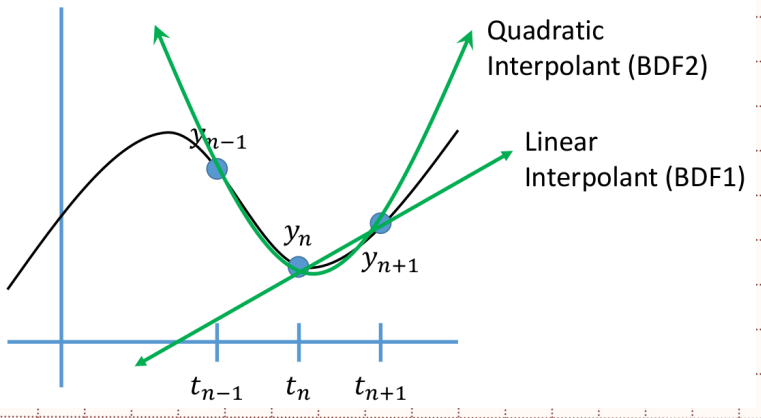
→ implicit Euler / trapezoidal : $O(h^3) \cdot O(h^{-1}) = O(h^2)$

→ RK4 : $O(h^5) \cdot O(h^{-1}) = O(h^4)$

- **single-step** methods use info from t_n (+ fwd) to solve for y_{n+1}
- **multi-step** methods use info from earlier timestamps (e.g. t_{n-1}, t_{n-2})
  - ↳ e.g. Adams-Bashforth, Adams-Moulton, bwd differentiation formula (BDF) schemes
  - ↳ 1 way to derive is to fit curves to current + earlier data pts for better slope estimates
    - e.g. fit quadratic to previous, current, + next step



- **bwd differentiation formulas (BDF)** are implicit multi-step schemes
  - ↳ # after BDF indicates order of global error
    - e.g. BDF1 is bwd Euler
    - e.g. BDF2 uses current + prev step data
      - → $y_{n+1} = \frac{4}{3} y_n - \frac{1}{3} y_{n-1} + \frac{2}{3} h f(t_{n+1}, y_{n+1})$
      - → LTE of $O(h^3)$
- deriving BDFs via **interpolation**:
  1) fit interpolant $p(t)$ w/ Lagrange polys to unknown pt (t_{n+1}, y_{n+1}) + 1+ earlier pts
  2) determine $p'(t)$ by differentiating
  - ↳ require end-of-step slope to match $p'(t_{n+1}) = f(t_{n+1}, y_{n+1})$
    - rearranging for unknown y_{n+1} gives BDF scheme
  - ↳ e.g.



↳ e.g. deriving bwd Euler

$$p(t) = y_n \underbrace{\left(\frac{t - t_{n+1}}{t_n - t_{n+1}}\right)}_{L_1(t)} + y_{n+1} \underbrace{\left(\frac{t - t_n}{t_{n+1} - t_n}\right)}_{L_2(t)}$$

$$= \frac{y_n}{-h}(t - t_{n+1}) + \frac{y_{n+1}}{h}(t - t_n)$$

$p'(t) = \frac{y_{n+1} - y_n}{h}$ gives line's slope

We require :

date

$$\frac{y_{n+1}-y_n}{h} = f(t_{n+1}, y_{n+1})$$

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}) \quad \leftarrow \text{BDF1}$$

↳ same approach extends to higher order, which yield implicit schemes w/ general form:

$$\sum_{k=-s+1}^{1} a_k y_{n+k} = h f(t_{n+1}, y_{n+1})$$

- a_k are coeffs

- explicit multi-step schemes are also common
  - ↳ e.g. 2nd order Adams-Banforth
    - $y_{n+1} = y_n + \frac{3}{2} h f(t_n, y_n) - \frac{1}{2} h f(t_{n-1}, y_{n-1})$
    - LTE is $O(h^3)$

- comparison of various time-stepping methods:

Name	Single/Multi-Step	Explicit/Implicit	Global Truncation Error
Forward Euler	Single	Explicit	$O(h)$
Improved Euler and Midpoint (2nd order Runge Kutta schemes)	Single	Explicit	$O(h^2)$
4th Order Runge Kutta	Single	Explicit	$O(h^4)$
Trapezoidal	Single	Implicit	$O(h^2)$
Backwards/Implicit Euler (BDF1)	Single	Implicit	$O(h)$
BDF2	Multi	Implicit	$O(h^2)$
2-step Adams-Bashforth	Multi	Explicit	$O(h^2)$
3rd order Adams-Moulton	Multi	Implicit	$O(h^3)$

HIGH ORDER ODES, STABILITY

date 02/12/2025

HIGH ORDER ODES

- order of ODE is highest derivative that appears
  - ↳ e.g.

$y''''(t) = y(t)$ → 4th order

$\theta''(t) = -\frac{g}{\ell}\sin(\theta)$, for an oscillating pendulum. → 2nd order

$F(t, x(t)) = m.x''(t)$, Newton's 2nd law ($F = m$), for some force function F.

- general form for higher order ODE : $y^{(n)}(t) = f(t, y(t), y'(t), y''(t), \ldots, y^{(n-1)}(t))$
  - ↳ i.e. all derivatives may be inter-related
  - ↳ convert to system of 1st order ODEs
    - for each var y w/ more than 1st derivative, intro new vars $y_i = y^{(i-1)}$ for $i = 1, \ldots, n$ so each derivative has corresponding var
    - subbing new vars into original ODE leads to:
      - → 1 1st order eqn for each original eqn
      - → 1+ additional eqns relating new vars
    - e.g.

Consider the IVP

$$y''(t) = ty(t)$$

with initial conditions, $y(1) = 1$, $y'(1) = 2$.

$y_1(t) = y^{(0)}(t) = y(t)$

$y_2(t) = y^{(1)}(t) = y'(t)$

System of eqns:

$y_2'(t) = t y_1'(t)$

$y_1'(t) = y_2(t)$

→ initial conds: $y_1(1) = 1$, $y_2(1) = 2$

    - e.g.

Convert

$$x''(t) + y'(t)x(t) + 2t = 0$$
$$y''(t) + (y(t))^2 x(t) = 0$$

to first order.

$x_1(t) = x(t)$ $\qquad$ $y_1(t) = y(t)$

$x_2(t) = x'(t)$ $\qquad$ $y_2(t) = y'(t)$

Plugging in:

$x_2'(t) + y_2(t)x_1(t) + 2t = 0$

$y_2'(t) + (y_1(t))^2 x_1(t) = 0$

$x_2(t) = x_1'(t)$

$y_2(t) = y_1'(t)$

$$\begin{cases} x_2'(t) = -y_2(t)x_1(t) - 2t \\ y_2'(t) = -(y_1(t))^2 x_1(t) \\ x_2(t) = x_1'(t) \\ y_2(t) = y_1'(t) \end{cases}$$

Vector form using $\vec{z} = \begin{bmatrix} x_1 \\ x_2 \\ y_1 \\ y_2 \end{bmatrix}$:

date

$$\begin{bmatrix} x_1'(t) \\ x_2'(t) \\ y_1'(t) \\ y_2'(t) \end{bmatrix} = \begin{bmatrix} x_2(t) \\ -y_2(t)x_1(t) - 2t \\ y_2(t) \\ -(y_1(t))^2 x_1(t) \end{bmatrix}$$

$$\vec{z}\,'(t) = \vec{f}(t, \vec{z})$$

STABILITY

- since errors are generally $O(h^p)$ for some p, schemes are less accurate for large h
  - ↳ errors in initial conds can lead to very diff / incorrect ans
    - $y'(t) = f(t, y(t))$, $y(0) = y_0 + \epsilon_0$
      - → ϵ_0 is initial error
    - if ϵ_0 grows exponentially for many steps (i.e. $n \to \infty$), time-stepping scheme is unstable

- e.g.

We'll consider a simple linear ODE, our "test equation",

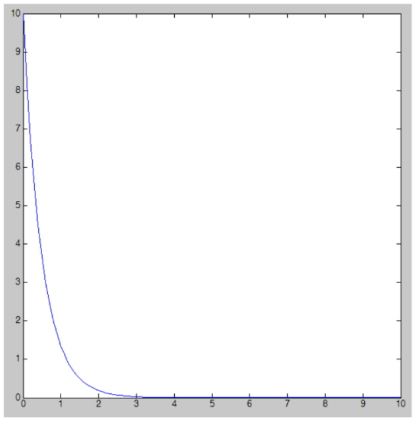
$y'(t) = -\lambda y(t),\ y(0) = y_0$

for constant $\lambda > 0$.

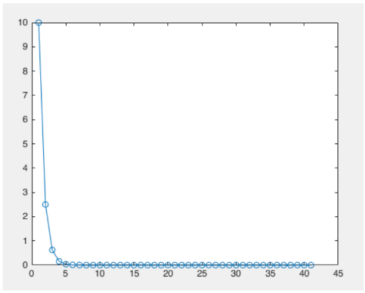
The exact solution is $y(t) = y_0 e^{-\lambda t}$.

Tends to 0 as $t \longrightarrow \infty$.

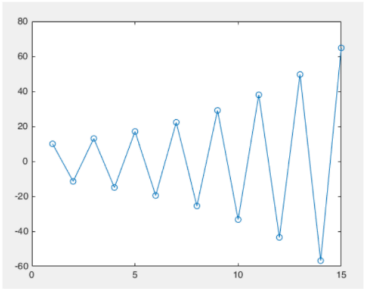
Any "reasonable" numerical approximation should do the same ...



↳ FE on test eqn:



Small h: Stable behavior, (correctly) decays to zero.



Large h: Unstable behavior, "blows up"!

- to examine stability of time-stepping schemes:
  1) apply given time-stepping scheme to test eqn
  2) find closed form of its numerical soln + error behaviour
  3) find conds on timestep h that ensure stability (i.e. error → 0)

- stability of FE:

$y_{n+1} = y_n + hf(t_n, y_n)$

Test eqn: $y'(t) = -\lambda y(t)$, $\lambda > 0$

Apply FE to test eqn:

$y_{n+1} = y_n + h(-\lambda y_n)$

date

$= y_n(1 - h\lambda)$

$y_n = y_{n-1}(1 - h\lambda)$

$y_{n+1} = y_{n-1}(1 - h\lambda)^2$

$\vdots$

$y_n = y_0(1 - h\lambda)^n$ ← closed form

↳ true soln has $y(t) = y_0 e^{-\lambda t}$ decaying to 0

↳ FE soln only goes to 0 when $|1 - h\lambda| < 1 \rightarrow -1 < 1 - h\lambda < 1$

- LS:

$-1 < 1 - h\lambda$

$h < \frac{2}{\lambda}$

- RS:

$1 - h\lambda < 1$

$h > 0$

↳ numerical soln decays to 0 when $0 < h < \frac{2}{\lambda}$

- otherwise, it blows up

Perturb initial conds w/ error ϵ_0:

$y_0^{(p)} = y_0 + \epsilon_0$

$y_n^{(p)} = (y_0 + \epsilon_0)(1 - h\lambda)^n$

$\epsilon_n = y_n^{(p)} - y_n$

$= \epsilon_0(1 - h\lambda)^n$

↳ error has same behaviour + blows up for $h > \frac{2}{\lambda}$

↳ FE is **conditionally stable**, meaning it's stable when h satisfies stability cond we derived

- stability of **BE (implicit)**:

$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1})$

Test eqn: $y'(t) = -\lambda y(t)$, $\lambda > 0$

Apply BE:

$y_{n+1} = y_n + h(-\lambda y_{n+1})$

$= y_n - h\lambda y_{n+1}$

$y_{n+1} = \frac{y_n}{1 + h\lambda}$

$y_n = \frac{y_{n-1}}{1 + h\lambda}$

$y_{n+1} = \frac{y_{n-1}}{(1 + h\lambda)^2}$

$\vdots$

$y_{n+1} = \frac{y_0}{(1 + h\lambda)^n}$ ← closed form

↳ since $h > 0$, $\lambda > 0$, val dec in magnitude as $n \to \infty$ for any $h > 0$

↳ error follows same form: $\epsilon_n = \frac{\epsilon_0}{(1 + h\lambda)^n}$

↳ BE is **unconditionally stable**

Stability does not imply accuracy. Large time steps still usually induce large (truncation) error.

- generally, implicit schemes are more stable than explicit ones at cost of potentially more expensive to solve

- stability of **improved Euler**:

$y^*_{n+1} = y_n + hf(t_n, y_n)$

$y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y^*_{n+1}))$

date

Test eqn: $y'(t) = -\lambda y(t)$, $\lambda > 0$

Apply improved Euler:

$$\tilde{y}_{n+1} = y_n + h(-\lambda y_n)$$

$$y_{n+1} = y_n + \frac{h}{2}(-\lambda y_n - \lambda \tilde{y}_{n+1})$$

$$= y_n + \frac{h}{2}(-\lambda y_n - \lambda(y_n + h(-\lambda y_n)))$$

$$= y_n - \frac{h\lambda}{2} y_n - \frac{h\lambda}{2}(y_n - h\lambda y_n)$$

$$= y_n - h\lambda y_n + \frac{h^2\lambda^2}{2} y_n$$

$$= y_n\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)$$

$$y_n = y_{n-1}\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)$$

$$y_{n+1} = y_{n-1}\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)^2$$

$$\vdots$$

Closed form:

$$y_{n+1} = y_0\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)^n$$

↳ error will likewise follow: $\epsilon_n = \epsilon_0\left(1 - h\lambda + \frac{h^2\lambda^2}{2}\right)^n$

↳ stable when $\left|1 - h\lambda + \frac{h^2\lambda^2}{2}\right| < 1$ or $-1 < 1 - h\lambda + \frac{h^2\lambda^2}{2} < 1$

- LS: $-1 < 1 - h\lambda + \frac{h^2\lambda^2}{2}$

$$0 < 2 - h\lambda + \frac{h^2\lambda^2}{2}$$

→ quadratic has no real roots so it's always satisfied for any h

- RS: $1 - h\lambda + \frac{h^2\lambda^2}{2} < 1$

$$-h\lambda + \frac{h^2\lambda^2}{2} < 0$$

$$h\lambda\left(-1 + \frac{h\lambda}{2}\right) < 0$$

→ $h\lambda > 0$ always so check when $-1 + \frac{h\lambda}{2} < 0 \longrightarrow h < \frac{2}{\lambda}$

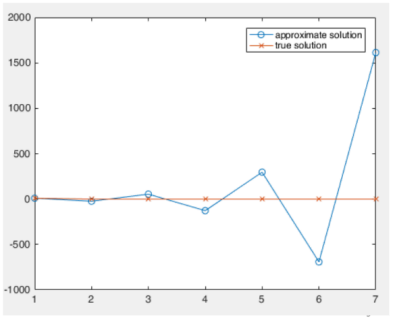
↳ improved Euler is conditionally stable when $0 < h < \frac{2}{\lambda}$

- same as FE

e.g. FE on test eqn

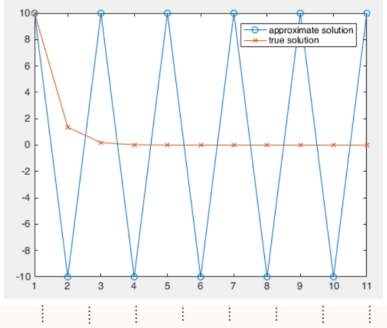
↳ $h > \frac{2}{\lambda}$ (unstable):

Numerical solution rapidly blows up.



↳ $h = \frac{2}{\lambda}$ (borderline stable):

Numerical solution remains constant, without decreasing to zero.

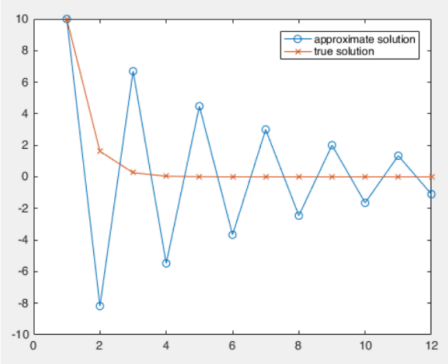


date

↳ $h < \frac{2}{\lambda}$ (stable, but inaccurate):

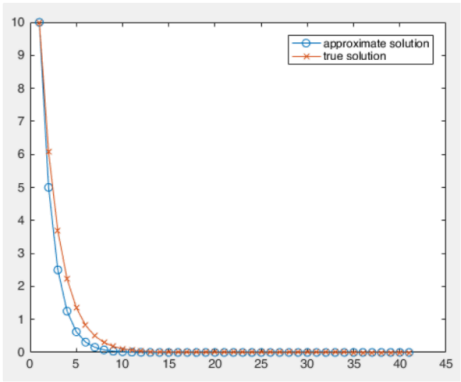
Numerical solution decreases to zero but still very inaccurate and oscillatory.

Note that the sign flips at each step! The true solution is always positive.



↳ $h << \frac{2}{\lambda}$ (stable, more accurate):

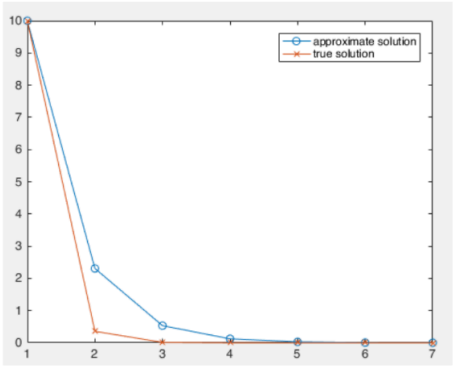
With even smaller steps, the numerical solution begins to approximate the true solution.



· e.g. BE on test eqn

↳ $h > \frac{2}{\lambda}$ (still stable):

Even with very large time steps, backward Euler solution is stable, and rapidly decreases to zero.



· for linear test eqn. FE gave bound related to $\lambda = \left|\frac{\partial}{\partial y}(-\lambda y)\right| = \left|\frac{\partial f}{\partial y}\right|$

· for nonlinear problems, linear stability depends on $\frac{\partial f}{\partial y}$ (for dynamics fcn f) evaled at given pt in time/space

· for systems of ODEs, stability relates to eigenvalues of Jacobian matrix $\frac{\partial(f_1, f_2, \ldots, f_n)}{\partial(y_1, y_2, \ldots, y_n)}$

truncation error, adaptive time-stepping

date 02/17/2025

TRUNCATION ERROR

- stability tells us what numerical algo itself does to small errors/perturbations
  - ↳ e.g. FE + improved Euler require $h > \frac{2}{\lambda}$ for test eqn or error will overwhelm ans
- truncation error tells us how accuracy of numerical soln scales w/ time step h, in absence of stability issues
- both of above factors must be considered to get useful soln
- LTE is error from taking 1 step
  - ↳ LTE = $y(t_{n+1}) - y_{n+1}$
    - $y(t_{n+1})$ is exact soln
    - y_{n+1} is apx soln
- process of finding LTE, given time-stepping scheme $y_{n+1} = \text{RHS}$:
  1) replace apxs on RHS w/ exact versions
     - e.g. $y_n \to y(t_n)$, $f(t_{n+1}, y_{n+1}) \to y'(t_{n+1})$
  2) Taylor expand all RHS quantities abt t_n (if necessary)
  3) Taylor expand exact soln $y(t_{n+1})$ to compare against
  4) compute diff $y(t_{n+1}) - y_{n+1}$
     - lowest deg non-cancelling power of h gives LTE

May not know in advance how many Taylor series terms needed. Try expanding up to given order + if all terms cancel out, redo using more terms.

- e.g. FE

$$y_{n+1} = y_n + hf(t_n, y_n)$$
$$= y(t_n) + hy'(t_n) \longrightarrow \text{nothing to Taylor expand on}$$

Exact:

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + O(h^3)$$

Diff:

$$\text{LTE} = y(t_{n+1}) - y_{n+1}$$
$$= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + O(h^3) - (y(t_n) + hy'(t_n))$$
$$= \frac{h^2}{2} y''(t_n) + O(h^3)$$
$$= O(h^2)$$

- e.g. trapezoidal LTE

$$y_{n+1} = y_n + \frac{h}{2}(f(t_n, y_n) + f(t_{n+1}, y_{n+1}))$$
$$= y(t_n) + \frac{h}{2}(y'(t_n) + y'(t_{n+1}))$$
$$= y(t_n) + \frac{h}{2}(y'(t_n) + y'(t_n) + hy''(t_n) + \frac{h^2}{2} y'''(t_n) + O(h^3))$$
$$= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{4} y'''(t_n) + O(h^4)$$

Diff:

$$\text{LTE} = y(t_{n+1}) - y_{n+1}$$
$$= y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{6} y'''(t_n) + O(h^4) -$$
$$(y(t_n) + hy'(t_n) + \frac{h^2}{2} y''(t_n) + \frac{h^3}{4} y'''(t_n) + O(h^4))$$
$$= -\frac{1}{12} h^3 y'''(t_n) + O(h^4)$$
$$= O(h^3)$$

- e.g. BDF2 LTE

$$y_{n+1} = \frac{4}{3} y_n - \frac{1}{3} y_{n-1} + \frac{2h}{3} f(t_{n+1}, y_{n+1})$$
$$= \frac{4}{3} y(t_n) - \frac{1}{3} y(t_{n-1}) + \frac{2h}{3} y'(t_{n+1})$$

Taylor expansions on RHS:

date

$$y'(t_{n+1}) = y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + O(h^3)$$
$$y(t_{n-1}) = y(t_n - h)$$
$$= y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y'''(t_n) + O(h^4)$$

Careful of "bwd in time" expansion for t_{n-1}. Use fact that $t_{n-1} = t_n - h$.

Plug in:

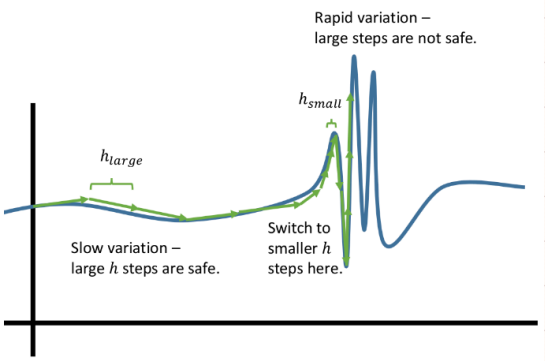
$$y_{n+1} = \frac{4}{3}y(t_n) - \frac{1}{3}\left(y(t_n) - hy'(t_n) + \frac{h^2}{2}y''(t_n) - \frac{h^3}{6}y'''(t_n) + O(h^4)\right) + \frac{2h}{3}\left(y'(t_n) + hy''(t_n) + \frac{h^2}{2}y'''(t_n) + O(h^3)\right)$$
$$= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{7h^3}{18}y'''(t_n) + O(h^4)$$

Diff:

$$LTE = y(t_{n+1}) - y_{n+1}$$
$$= y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y'''(t_n) + O(h^4) - \left(y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{7h^3}{18}y'''(t_n) + O(h^4)\right)$$
$$= -\frac{2}{9}y'''(t_n) + O(h^4)$$
$$= O(h^3)$$

ADAPTIVE TIME-STEPPING

- time-stepping methods advance by discrete time step size h
  - ↳ so far, h held constant
  - ↳ since error was $O(h^p)$ for some p, smaller h ($\to 0$) implied smaller error, both locally + globally
- smaller h requires more steps to reach t_{final}
  - ↳ # steps $= \frac{t_{final} - t_0}{h} = O(h^{-1})$
  - ↳ higher computational cost
  - ↳ more accurate
- larger h has fewer steps (i.e. cheaper), but intros excessive error + stability issues for some schemes
  - ↳ less accurate
- minimize effort / size by choosing largest h w/ error < desired tolerance
- rate of change of soln often varies over time
  - ↳ adapt time step during computation to keep error small + minimize wasted effort
  - ↳ e.g.



- "chicken + egg" problem b/c if we knew error for given h, could choose good h to always satisfy error < tolerance, but if we need soln to know exact error
  - ↳ key idea: run 2 schemes w/ diff truncation error orders, then compare

date

results to adjust h

- e.g.
  1) y^A_{n+1} = <method A> w/ $O(h^4)$
  2) y^B_{n+1} = <method B> w/ $O(h^5)$
  → apx error as $err = |y^A_{n+1} - y^B_{n+1}|$

- adaptive time-stepping using order p, order p+1 methods: if err > chosen threshold, reduce h (e.g. by $\frac{1}{2}$) + recompute step again, repeating until bound / tolerance is satisfied
  ↳ $y^A_{n+1} = y(t_{n+1}) + O(h^p) = y(t_{n+1}) + Ch^p + O(h^{p+1})$
    - for some C, p
  ↳ if B is 1 order more accurate (i.e. $O(h^{p+1})$), then $y^B_{n+1} = y(t_{n+1}) + O(h^{p+1})$
  ↳ A's true error is $|y^A_{n+1} - y(t_{n+1})| = Ch^p + O(h^{p+1})$
  ↳ estimated error is $|y^A_{n+1} - y^B_{n+1}| = Ch^p + O(h^{p+1})$ as well
  ↳ since dominant (i.e. lowest order) component of error matches, estimate is decent apx to adjust h

- to estimate next time step size:
  ↳ estimate error coeff C as $C \simeq \frac{|y^A_{n+1} - y^B_{n+1}|}{(h_{old})^p}$
    - h_{old} is most recent time step size
  ↳ if C changes slowly in time, $err_{next} = |y^A_{n+2} - y(t_{n+2})|$

$$\simeq C(h_{new})^p$$
$$\simeq \frac{|y^A_{n+1} - y^B_{n+1}|}{(h_{old})^p}(h_{new})^p$$
$$err_{next} \simeq \left(\frac{h_{new}}{h_{old}}\right)^p |y^A_{n+1} - y^B_{n+1}|$$

  ↳ given desired error tolerance tol:

$$err_{next} = tol$$
$$\left(\frac{h_{new}}{h_{old}}\right)^p |y^A_{n+1} - y^B_{n+1}| = tol$$
$$h_{new} = h_{old}\left(\frac{tol}{|y^A_{n+1} - y^B_{n+1}|}\right)^{\frac{1}{p}}$$

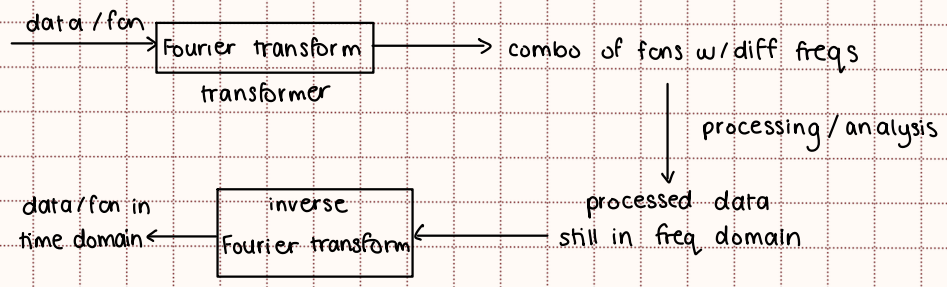
  ↳ to compensate for apxs, can scale tol down by factor of α (e.g. $\frac{1}{2}, \frac{3}{4}$) so it'll allow step size to grow larger again when it's likely safe to do so

- basic algo for adaptive time-stepping:
  1) compute apx solns w/ 2 schemes of diff orders
  2) estimate error by taking diff
  3) while err > tolerance, set $h = \frac{h}{2}$ + repeat steps 1, 2
  4) estimate error coeff + predict good h_{new}
  5) repeat until end time reached

fourier transforms intro

date 02/22/2025

- basic idea of Fourier analysis:
  1) transform some fcn / data into form that reveals freqs of info in data
  2) process / analyze in freq domain
  3) transform back again



- og data is:
  - ↳ in time domain if f is fcn of time, f(t)
  - ↳ in spatial domain if f is fcn of space/pos. f(x)
- e.g. JPEG compression

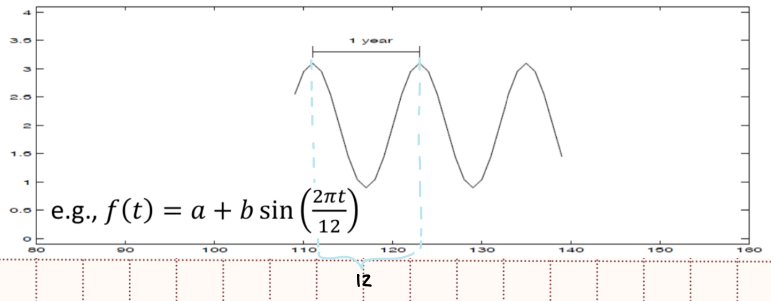
By discarding less "important" information in the signal/data, we can save memory.

(a) Original (b) Compressed by 85%

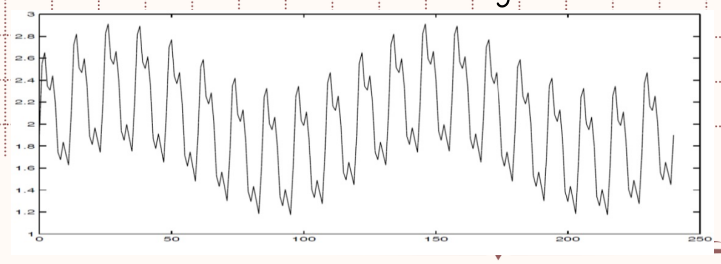
- e.g. orange juice pricing

Consider the price of orange juice over many months.

Typically *cyclical* or *periodic* (i.e., pattern repeats over time) due to e.g. seasonal variation in supply and demand. (We might approximate it with a sine curve.)



↳ other recurring phenomena can have effects on diff time scales (e.g. weather fluctuations, import cost variations) so data might look more like:

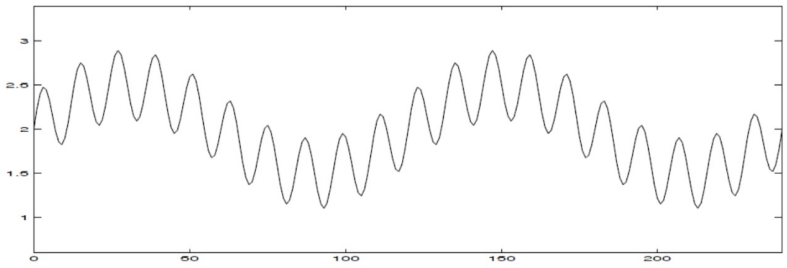


↳ to rep additional fluctuations in data, add more sinusoidal terms w/ diff periods/freqs
- general form: $f(t) = a_0 + a_1\cos(qt) + b_1\sin(qt) + a_2\cos(2qt) + b_2\sin(2qt) + \dots$
  → in example, $q = \frac{2\pi}{T}$ where T is total time period (e.g. 240 months of pricing data)

↳ e.g. 2 sources of fluctuations:

Frequency of fcn w/ period T is $f = \frac{1}{T}$.

$$f(t) = a_0 + b_2\sin(2q.t) + b_{20}\sin(20q.t)$$



- basic idea of Fourier analysis is to convert time/space-dependent data into rep $f(t) = a_0 + a_1\cos(qt) + b_1\sin(qt) + a_2\cos(2qt) + b_2\sin(2qt) + \dots$
  ↳ i.e. infinite sum of increasingly high freq sine/cosine
  ↳ a_i, b_i determine fcn

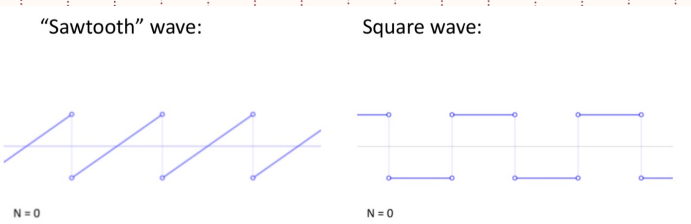
CONTINUOUS FOURIER SERIES

- consider some continuous periodic fcn $f(t)$ w/ period T
  ↳ $f(t \pm T) = f(t)$
    - i.e. f repeats after 1 period T
  ↳ e.g.

e.g., $\sin\left(\frac{2\pi kt}{T}\right)$ or $\cos\left(\frac{2\pi kt}{T}\right)$ satisfy this for all integers k.

$$\cos\left(\frac{2\pi k(t+T)}{T}\right) = \cos\left(\frac{2\pi kt}{T} + 2k\pi\right) = \cos\left(\frac{2\pi kt}{T}\right)$$

  ↳ goal is to rep any $f(t)$ as infinite sum of trig fcns:
  $f(t) = a_0 + \sum_{k=1}^{\infty} a_k\cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{\infty} b_k\sin\left(\frac{2\pi kt}{T}\right)$
    - a_k, b_k are info/amplitude of each sinusoid of a specific period $\frac{T}{k}$ or freq $\frac{k}{T}$
    - higher k → shorter period + higher wave freq
- e.g. more sinusoids give better apx



  ↳ square wave apx:

date

- assuming $t \in [0, 2\pi]$, period $T = 2\pi$, then to find coeffs of $f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$, use some identities to find each coeff alone:
  - $\int_0^{2\pi} \cos(kt)\sin(jt)\,dt = 0$ for any ints k, j
    - 2 fcns are orthogonal to each other
  - $\int_0^{2\pi} \cos(kt)\cos(jt)\,dt = 0$ for $k \neq j$
  - $\int_0^{2\pi} \sin(kt)\sin(jt)\,dt = 0$ for $k \neq j$
  - $\int_0^{2\pi} \sin(kt)\,dt = 0$
  - $\int_0^{2\pi} \cos(kt)\,dt = 0$
- e.g. determining coeffs of Fourier series: $f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$

$$\int_0^{2\pi} f(t)\,dt = a_0 \int_0^{2\pi} dt + \sum_{k=1}^{\infty} a_k \int_0^{2\pi} \cos(kt)\,dt + \sum_{k=1}^{\infty} b_k \int_0^{2\pi} \sin(kt)\,dt$$

$$a_0 = \frac{\int_0^{2\pi} f(t)\,dt}{\int_0^{2\pi} dt}$$

$$a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,dt$$

  - a_0 is avg val of f over $[0, 2\pi]$

Determining coeff a_ℓ for $\ell > 0$:

$$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$$

$$\int_0^{2\pi} f(t)\cos(\ell t)\,dt = \int_0^{2\pi} a_0 \cos(\ell t)\,dt + \sum_{k=1}^{\infty} a_k \int_0^{2\pi} \cos(kt)\cos(\ell t)\,dt + \sum_{k=1}^{\infty} b_k \int_0^{2\pi} \sin(kt)\cos(\ell t)\,dt$$

$$= a_\ell \int_0^{2\pi} \cos^2(\ell t)\,dt$$

$$= a_\ell \pi$$

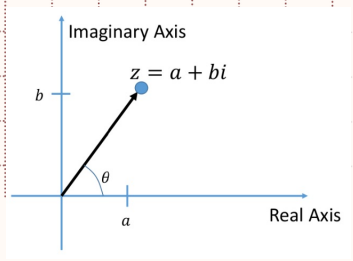
$$a_\ell = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(\ell t)\,dt$$

- if we have f, can find coeffs by solving integrals:
  - $a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,dt$
  - $a_k = \frac{\int_0^{2\pi} f(t)\cos(kt)\,dt}{\int_0^{2\pi} \cos^2(kt)\,dt} = \frac{1}{\pi}\int_0^{2\pi} f(t)\cos(kt)\,dt$
  - $b_k = \frac{\int_0^{2\pi} f(t)\sin(kt)\,dt}{\int_0^{2\pi} \sin^2(kt)\,dt} = \frac{1}{\pi}\int_0^{2\pi} f(t)\sin(kt)\,dt$
- changing phase of input fcn transfers some weight btwn a_k, b_k coeffs

COMPLEX NUMBER REVIEW

- for $z \in \mathbb{C}$, we write $z = a + bi$
  - $a, b \in \mathbb{R}$
  - $i = \sqrt{-1}$ is imaginary #
  - visualized as pts on complex plane:

$i^2 = -1$



date

- vector length is modulus
- vector angle is argument

additional operations:

↳ conjugate: $\bar{z} = a - bi$

↳ modulus / norm: $|z| = \sqrt{a^2+b^2}$

↳ arg / phase / angle: $Arg(z) = atan2(b,a) = arctan(\frac{b}{a})$

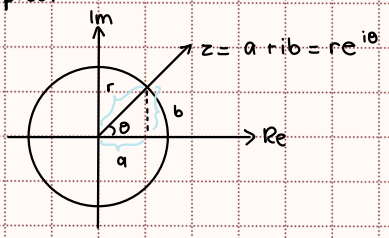
- converts regular coords (x,y) to polar (r,θ)

fourier transforms: discrete setting

date 03/08/2025

- can write any periodic fcn over period T as $f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos\left(\frac{2\pi kt}{T}\right) + \sum_{k=1}^{\infty} b_k \sin\left(\frac{2\pi kt}{T}\right)$
  - ↳ $a_0 = \frac{1}{2\pi}\int_0^{2\pi} f(t)\,dt$
  - ↳ $a_k = \frac{\int_0^{2\pi} f(t)\cos(kt)\,dt}{\int_0^{2\pi}\cos^2(kt)\,dt}$
  - ↳ $b_k = \frac{\int_0^{2\pi} f(t)\sin(kt)\,dt}{\int_0^{2\pi}\sin^2(kt)\,dt}$
- Euler's formula: $e^{i\theta} = \cos\theta + i\sin\theta$
  - ↳ for any $\theta \in \mathbb{R}$
  - ↳ conjugate of $e^{i\theta}$: $e^{-i\theta} = \cos(-\theta) + i\sin(-\theta)$

    $e^{-i\theta} = \cos\theta - i\sin\theta$
  - ↳ adding / subtracting $e^{-i\theta}$, $e^{i\theta}$ leads to 2 identities:
    - $\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2}$
    - $\sin\theta = \frac{e^{i\theta} - e^{i\theta}}{2i}$
    - proof:



$r = \sqrt{a^2+b^2}$

$\cos\theta = \frac{a}{\sqrt{a^2+b^2}} = \frac{a}{r}$

$\sin\theta = \frac{b}{\sqrt{a^2+b^2}} = \frac{b}{r}$

$\cos\theta + i\sin\theta = \frac{a}{r} + i\frac{b}{r}$

$= \frac{1}{r}(a + ib)$

$= \frac{z}{r}$

$= \frac{re^{i\theta}}{r}$

$= e^{i\theta}$ ✓

$\frac{e^{i\theta} + e^{-i\theta}}{2} = \frac{(\cos\theta + i\sin\theta) + (\cos\theta - i\sin\theta)}{2}$

$= \cos\theta$ ✓

$\frac{e^{i\theta} - e^{-i\theta}}{2} = \frac{(\cos\theta + i\sin\theta) - (\cos\theta - i\sin\theta)}{2i}$

$= \sin\theta$ ✓

- can also express earlier sinusoidal expression of f(t) as $f(t) = \sum_{k=-\infty}^{+\infty} c_k e^{ikt}$
  - ↳ c_k coeffs are complex #s
  - ↳ to convert btwn $c_k \leftrightarrow a_k, b_k$:

$f(t) = a_0 + \sum_{k=1}^{\infty} a_k \cos(kt) + \sum_{k=1}^{\infty} b_k \sin(kt)$

$= a_0 + \sum_{k=1}^{\infty} a_k \frac{e^{ikt} + e^{-ikt}}{2} + \sum_{k=1}^{\infty} b_k \frac{e^{ikt} - e^{-ikt}}{2i}$

$= a_0 + \sum_{k=1}^{\infty} \frac{a_k}{2} e^{ikt} + \frac{a_k}{2} e^{-ikt} + \frac{b_k}{2i} e^{ikt} - \frac{b_k}{2i} e^{-ikt}$

$= a_0 + \sum_{k=1}^{\infty} \left(\frac{a_k}{2} + \frac{b_k}{2i}\right) e^{ikt} + \sum_{k=1}^{\infty} \left(\frac{a_k}{2} - \frac{b_k}{2i}\right) e^{-ikt}$

$= a_0 + \sum_{k=1}^{\infty} \left(\frac{a_k}{2} - \frac{b_k}{2}i\right) e^{ikt} + \sum_{k=1}^{\infty} \left(\frac{a_k}{2} + \frac{b_k}{2}i\right) e^{-ikt}$

$f(t) = a_0 + \sum_{k=1}^{\infty} \left(\frac{a_k}{2} - \frac{b_k}{2}i\right) e^{ikt} + \sum_{k=-\infty}^{-1} \left(\frac{a_k}{2} + \frac{b_k}{2}i\right) e^{ikt}$

$a_0 e^{i(0)t}$ ($k = 0$) — k is +ve — k is -ve

date

↳ for $k>0$:
- $a_0 = c_0$
- $c_k = \frac{a_k}{2} - \frac{ib_k}{2}$
- $c_{-k} = \frac{a_k}{2} + \frac{ib_k}{2}$

↳ rltnships:
- $|a_0| = |c_0|$
- $|c_k| = |c_{-k}| = \frac{1}{2}\sqrt{a_k^2 + b_k^2}$

↳ can find c_k directly by using orthogonality property: $\int_0^{2\pi} e^{ikt} e^{-i\ell t} = \begin{cases} 0, & k \neq \ell \\ 2\pi, & k = \ell \end{cases}$
- $c_k = \frac{1}{2\pi}\int_0^{2\pi} e^{-ikt} f(t)\,dt$

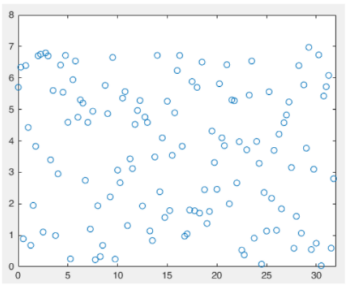
DISCRETE INPUT DATA

· apx of fcn can be done by truncating series to finite # sinusoids
↳ $f(t) \simeq \sum_{k=-M}^{+M} c_k e^{ikt}$

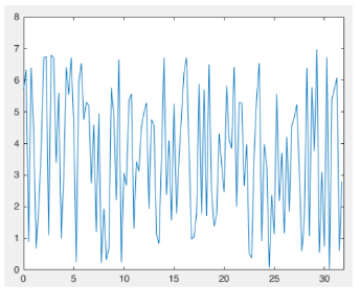
· consider vector of discrete data
↳ e.g. $f_0, f_1, \ldots, f_{N-1}$ for N uniformly spaced data pts
- assume N is even

↳ assume data are from unknown fcn $f(t)$, evaled at $t_n = n\Delta t = \frac{nT}{N}$ for $n = 0, 1, \ldots, N-1$
- i.e. $f_n = f(t_n)$

↳ e.g.



N=128 discrete data samples over a domain [0,32].



Piecewise linear plot.

12

- $N = 128$, $T = 32$
- for N pts, use N degs of freedom (i.e. N coeffs) to exactly interpolate data
- assuming N is even, apx w/ truncated Fourier series as $f(t) \simeq \sum_{k=-\frac{N}{2}+1}^{N/2} c_k e^{\frac{(2\pi i)kt}{T}}$
  → if $T = 2\pi$, then $f(t) \simeq \sum_{k=-\frac{N}{2}+1}^{N/2} c_k e^{ikt}$
- plugging each data pt (t_n, f_n) into expr gives us N eqns involving unknown coeffs c_k

· deriving (inverse) discrete Fourier transform (DFT)

$f(t) = \sum_{k=-\frac{N}{2}+1}^{N/2} c_k e^{\frac{2\pi i k t}{T}}$

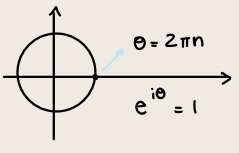
For n^{th} discrete pt, $t_n = \frac{nT}{N}$, rltn is:

$$f_n = \sum_{k=-\frac{N}{2}+1}^{N/2} c_k e^{\frac{2\pi i k t_n}{T}}$$
$$= \sum_{k=-\frac{N}{2}+1}^{N/2} c_k e^{\frac{2\pi i k n}{N}}$$
$$= \sum_{k=0}^{N/2} c_k e^{\frac{i2\pi nk}{N}} + \sum_{k=-\frac{N}{2}+1}^{-1} c_k e^{\frac{i2\pi nk}{N}}$$

Apply change of var $j = N+k$ on second sum:

$$\sum_{k=-\frac{N}{2}+1}^{-1} c_k e^{\frac{i2\pi nk}{N}} = \sum_{j=\frac{N}{2}+1}^{N-1} c_{j-N} e^{\frac{i2\pi n(j-N)}{N}}$$
$$= \sum_{j=\frac{N}{2}+1}^{N-1} c_{j-N} e^{\frac{i2\pi nj}{N}} \cdot e^{\frac{-i2\pi nN}{N}}$$

$e^{(-2\pi ni)} = \cos(2\pi n) - i\sin(2\pi n) = 1$ for any int n.



date

$= \sum_{j=\frac{N}{2}+1}^{N-1} c_{j-N} e^{\frac{i2\pi nj}{N}}$

Define c_j outside range $[-\frac{N}{2}+1, \frac{N}{2}]$ to be periodic, so $c_{j\pm N} = c_j$.

$\sum_{j=\frac{N}{2}+1}^{N-1} c_{j-N} e^{\frac{i2\pi nk}{N}} = \sum_{j=\frac{N}{2}+1}^{N-1} c_j e^{\frac{i2\pi nj}{N}}$

$f_n = \sum_{k=0}^{N/2} c_k e^{\frac{i2\pi nk}{N}} + \sum_{j=\frac{N}{2}+1}^{N-1} c_j e^{\frac{i2\pi nj}{N}}$

$f_n = \sum_{k=0}^{N-1} c_k e^{\frac{i2\pi nk}{N}}$

↳ c_k are N discrete Fourier coeffs for N input pts f_n

- rename to F_k

↳ final notation for IDFT: $f_n = \sum_{k=0}^{N-1} F_k e^{\frac{i2\pi nk}{N}} = \sum_{k=0}^{N-1} F_k W^{nk}$

- $W = e^{\frac{2\pi i}{N}}$ is N^{th} root of unity b/c it satisfies $W^N = e^{2\pi i} = 1$

w/ $W = e^{\frac{2\pi i}{N}}$, then $W^k = e^{\frac{2\pi ik}{N}}$ are also N^{th} roots of unity for $k \in \mathbb{Z}$

↳ e.g. for N = 3, roots of unity have form $W^k = e^{k(\frac{2\pi i}{3})}$

↳ each N^{th} root corresponds to pt on unit circle in complex plane

- visualizing on complex plane:

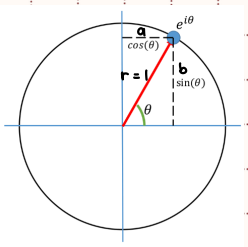
$W^k = e^{\frac{2\pi k}{N} i}$

$= e^{i\theta}$

$= \underbrace{\cos\theta}_{a} + \underbrace{i\sin\theta}_{b}$

$r = \sqrt{\cos^2\theta + \sin^2\theta}$

$= 1$



↳ e.g. N = 3

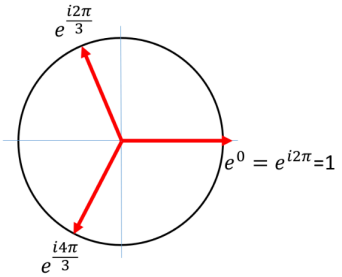
By Euler formula
$e^{i\theta} = \cos(\theta) + i\sin(\theta)$, evaluating
$W = e^{\frac{2\pi ik}{3}}$ for $k = 0, 1, 2, 3$ gives:

$k = 0$ or 3 : $e^0 = e^{i2\pi} = 1 + i0$

$k = 1$: $e^{\frac{2i\pi}{3}} = -\frac{1}{2} + i\frac{\sqrt{3}}{2}$

$k = 2$: $e^{\frac{4i\pi}{3}} = -\frac{1}{2} - i\frac{\sqrt{3}}{2}$

Since these are cube roots of unity, cubing makes the angle (a multiple of) 2π, giving 1.



$W^0 = (e^{\frac{2\pi i}{3}})^0$

$W^1 = (e^{\frac{2\pi i}{3}})^1$

$W^2 = (e^{\frac{2\pi i}{3}})^2$

$W^3 = (e^{\frac{2\pi i}{3}})^3 = e^{2\pi i} = 1 = W^0$

$k = 0$:

$W^0 = e^{0i}$

$= \cos 0 + i\sin 0$

$= 1$

$k = 1$:

$W^1 = e^{\frac{2\pi i}{3}}$

$= \cos(\frac{2\pi}{3}) + i\sin(\frac{2\pi}{3})$

$= -\frac{1}{2} + \frac{\sqrt{3}}{2}i$

$k = 2$:

$W^2 = e^{\frac{4\pi i}{3}}$

$= \cos\left(\frac{4\pi}{3}\right) + i\sin\left(\frac{4\pi}{3}\right)$

$= -\frac{1}{2} - \frac{\sqrt{3}}{2}i$

$k = 3$:

$W^3 = e^{2\pi i}$

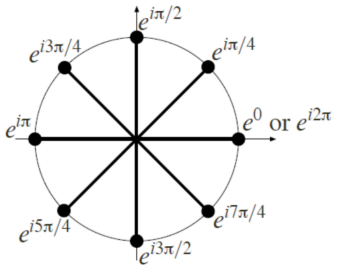
$= \cos 2\pi + i\sin 2\pi$

$= 1$

↳ e.g. $N = 8$

By Euler formula
$e^{i\theta} = \cos(\theta) + i\sin(\theta)$, evaluating $e^{\frac{\pi i k}{4}}$
for $k = 0, 1, \ldots, 8$ gives:

$$e^0 = e^{2i\pi} = 1,$$
$$e^{\frac{i\pi}{4}} = \frac{1}{\sqrt{2}} + i\frac{1}{\sqrt{2}},$$
$$e^{\frac{i\pi}{2}} = i$$
$$e^{\frac{3i\pi}{4}} = \frac{-1}{\sqrt{2}} + i\frac{1}{\sqrt{2}},$$
$$e^{i\pi} = -1$$
$$e^{\frac{5i\pi}{4}} = \frac{-1}{\sqrt{2}} + i\frac{-1}{\sqrt{2}}$$
$$e^{\frac{3i\pi}{2}} = -i$$
$$e^{\frac{7i\pi}{4}} = \frac{1}{\sqrt{2}} + i\frac{-1}{\sqrt{2}}$$



dft and its properties

date 03/08/2025

- to be useful, ops needed are:
  1) convert input time-domain data f_n to freq-domain F_k
  2) convert freq-domain data F_k back to time-domain f_n
     - IDFT: given Fourier coeffs F_k, we have $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$
- for 1), to find F_k need this property of N^{th} roots of unity:

$$\sum_{j=0}^{N-1} W^{jk} W^{-jl} = \sum_{j=0}^{N-1} W^{j(k-l)} = N\delta_{k,l}$$

  - ↳ assume for now that $k, l \in [0, N-1]$
    - $k, l \in \mathbb{Z}$
  - ↳ $\delta_{k,l}$ is Kronecker delta fcn: $\delta_{kl} = \begin{cases} 0, & k \neq l \\ 1, & k = l \end{cases}$
  - ↳ orthogonality identity will allow us to work in reverse dir (i.e. f_n to F_k)
  - ↳ derivation:

Assume k, l are ints in $[0, N-1]$.

Case 1: $k = l$

$$\sum_{j=0}^{N-1} W^{j(0)} = \sum_{j=0}^{N-1} 1 = N$$

Case 2: $k \neq l$

$$x^N - 1 = (x-1)(x^{N-1} + x^{N-2} + \dots + x + 1)$$

$$= (x-1)\left(\sum_{j=0}^{N-1} x^j\right)$$

$$\sum_{j=0}^{N-1} x^j = \frac{x^N - 1}{x-1} \longrightarrow \text{geometric series}$$

Apply to our expr:

$$\sum_{j=0}^{N-1} W^{j(k-l)} = \sum_{j=0}^{N-1} (W^{k-l})^j$$

$$= \frac{(W^{k-l})^N - 1}{W^{k-l} - 1}$$

$$= \frac{(W^N)^{k-l} - 1}{W^{k-l} - 1}$$

$$= \frac{1^{k-l} - 1}{W^{k-l} - 1} \longrightarrow W^N = 1 \text{ is } N^{th} \text{ root of unity}$$

$$= 0$$

- determining F_k coeffs (i.e. DFT):
  - ↳ use orthogonality to isolate each F_k
  - ↳ from IDFT, $f_n = \sum_{j=0}^{N-1} F_j W^{nj}$
  - ↳ to get k^{th} coeff, multiply by W^{-nk} then sum from 0 to N-1:

$$\sum_{n=0}^{N-1} f_n W^{-nk} = \sum_{n=0}^{N-1} \sum_{j=0}^{N-1} F_j W^{nj} W^{-nk}$$

$$= \sum_{j=0}^{N-1} F_j \sum_{n=0}^{N-1} W^{n(j-k)}$$

$$= \sum_{j=0}^{N-1} F_j \cdot N\delta_{j,k} \longleftarrow \text{orthogonality identity}$$

$$= F_k \cdot N$$

  - ↳ DFT is $F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$
    - given data f_n for $n = 0, \dots, N-1$, can find corresponding F_k coeffs for $k = 0, \dots, N-1$
- DFT is invertible
  - ↳ inverse DFT: $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$
  - ↳ DFT: $F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$
- e.g.

date

Given 4 discrete Fourier coefficients F_k, find the corresponding 4 data points f_n.

Let $F = \begin{bmatrix} -2 \\ 2+i \\ -2 \\ 2-i \end{bmatrix}$. What is the vector f?

Equivalently:

If $F_0 = -2$, $F_1 = 2+i$, $F_2 = -2$, and $F_3 = 2-i$, with $N = 4$. Find f_0, f_1, f_2, and f_3.

$F = [-2, 2+i, -2, 2-i]$

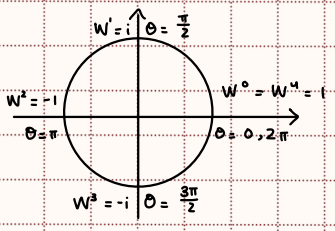
$W = e^{\frac{2\pi i}{N}}$

$= e^{\frac{2\pi i}{4}}$

$= e^{\frac{\pi i}{2}}$

$= \cos\frac{\pi}{2} + i\sin\frac{\pi}{2}$

$= i$



$f_n = \sum_{k=0}^{N-1} F_k W^{nk}$

$n = 0$:

$f_0 = \sum_{k=0}^{3} F_k W^0$

$= \sum_{k=0}^{3} F_k$

$= -2 + 2 + i - 2 + 2 - i$

$= 0$

$n = 1$:

$f_1 = \sum_{k=0}^{3} F_k W^k$

$= (-2)W^0 + (2+i)W^1 + (-2)W^2 + (2-i)W^3$

$= (-2)(1) + (2+i)(i) + (-2)(-1) + (2-i)(-i)$

$= -2 + 2i - 1 + 2 - 2i - 1$

$= -2$

$n = 2$:

$f_2 = \sum_{k=0}^{3} F_k W^{2k}$

$= (-2)W^0 + (2+i)W^2 + (-2)W^4 + (2-i)W^6$

$= (-2)(1) + (2+i)(-1) + (-2)(1) + (2-i)(-1)$

$= -2 - 2 - i - 2 - 2 + i$

$= -8$

$n = 3$:

$f_3 = \sum_{k=0}^{3} F_k W^{3k}$

$= (-2)W^0 + (2+i)W^3 + (-2)W^6 + (2-i)W^9$

$= (-2)(1) + (2+i)(-i) + (-2)(-1) + (2-i)(i)$

$= -2 - 2i + 1 + 2 + 2i + 1$

$= 2$

- e.g. DFT of cosine wave

Consider a set of N data points defined such that $f_n = \cos\left(\frac{2\pi n}{N}\right)$.

Show that $F_1 = F_{N-1} = \frac{1}{2}$; for all other coefficients, $F_k = 0$.

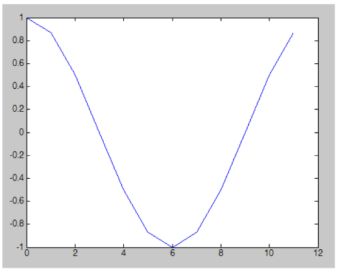
Therefore we can express f_n in our Fourier representation as

$$f_n = \cos\left(\frac{2\pi n}{N}\right) = \sum_{k=0}^{N-1} F_k W^{nk} = \frac{1}{2}W^{n(1)} + \frac{1}{2}W^{n(N-1)}$$

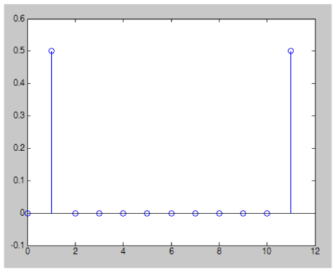
date

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} \cos\left(\frac{2\pi n}{N}\right) W^{-nk}$$
$$= \frac{1}{N}\sum_{n=0}^{N-1} \frac{1}{2}\left(e^{i\frac{2\pi n}{N}} + e^{-i\frac{2\pi n}{N}}\right) W^{-nk}$$
$$= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n} + W^{-n}\right) W^{-nk} \qquad \leftarrow W = e^{\frac{2\pi i}{N}}$$
$$= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{-n(1+k)}\right)$$
$$= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{-n(1+k)} W^{Nn}\right) \qquad \leftarrow W^{Nn} = (W^N)^n = 1^n = 1$$
$$= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{Nn-n-nk}\right)$$
$$= \frac{1}{2N}\sum_{n=0}^{N-1} \left(W^{n(1-k)} + W^{n(N-1-k)}\right)$$
$$= \frac{1}{2N}\left(N\delta_{1,k} + N\delta_{N-1,k}\right)$$
$$= \frac{1}{2}\left(\delta_{1,k} + \delta_{N-1,k}\right)$$

$$F_k = \begin{cases} \frac{1}{2}, & k = 1, N-1 \\ 0, & \text{otherwise} \end{cases}$$



Original Data f_n



Resulting Fourier coefficients F_k

- 2 properties of DFT, as consequences of props of N^{th} roots of unity:

1) sequence $\{F_k\}$ is doubly infinite + periodic
   - ↳ i.e. if we allow $k < 0$ or $k > N-1$, coeffs repeat
   - ↳ proof:

Given F_k for ints $k \in [0, N-1]$, then for ints $k \in (-\infty, +\infty)$, F_k is 1 of existing vals.

Express arbitrary k as $k = mN + p$ for $p \in [0, N-1]$ (i.e. $k \equiv p \pmod N$).

$$W^{-k} = e^{-i\frac{2\pi}{N}(mN+p)}$$
$$= e^{-i2\pi m} \cdot e^{-i\frac{2\pi p}{N}}$$
$$= (\cos(2\pi m) + i\sin(2\pi m)) \cdot (e^{\frac{2\pi i}{N}})^{-p}$$
$$= 1 \cdot W^{-p}$$
$$= W^{-p}$$
$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$$
$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-np}$$
$$= F_p \qquad \leftarrow p \in [0, N-1]$$

2) conjugate symmetry: if data f_n is real, $F_k = \overline{F_{N-k}}$
   - ↳ $|F_k|$ are symmetric abt $k = \frac{N}{2}$
   - ↳ proof:

1) $W^{N-k} = e^{\frac{2\pi i}{N}(N-k)}$
$$= e^{2\pi i} \cdot e^{-\frac{2\pi i k}{N}}$$
$$= W^{-k}$$

2) since $e^{i\theta} = \cos\theta + i\sin\theta$, $e^{-i\theta} = \cos\theta - i\sin\theta$, we observe $\overline{e^{i\theta}} = e^{-i\theta}$
   - → likewise, $\overline{W^j} = W^{-j}$

3) for real x, $\bar{x} = x$ since imaginary part is 0

date

Using above 3 facts:

$$\overline{F_{N-k}} = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-n(N-k)}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n \overline{W^{-n(N-k)}} \quad \leftarrow \text{fact 3 since } \tfrac{1}{N},\ f_n \text{ are real}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-n(-k)} \quad \leftarrow \text{fact 1}$$

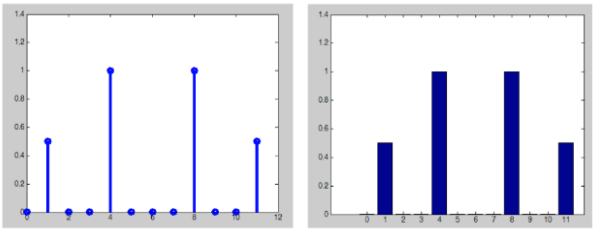
$$= \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk} \quad \leftarrow \text{fact 2}$$

$$= F_k$$

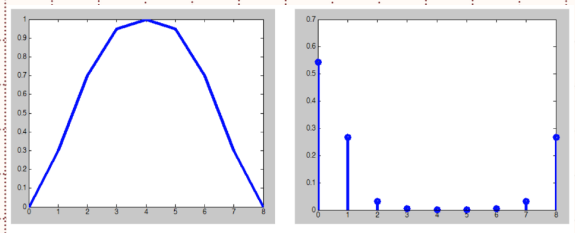
dft properties

date 03/10/2025

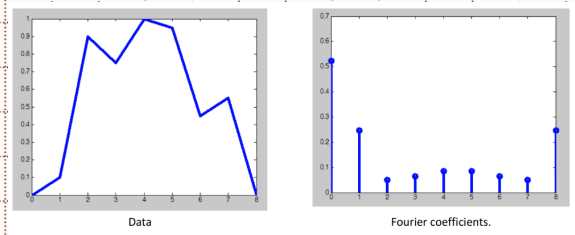
power spectrum visualizes Fourier coeffs by plotting their moduli / magnitudes $|F_k|$
- ↳ expresses freq magnitudes, but ignores phases
- ↳ other sources might use $|F_k|^2$
- ↳ e.g.



- e.g. $\cos\left(\frac{2\pi n}{N}\right)$: **smooth vs rough** bumps in discrete data
  - ↳ smooth:



  - ↳ rough:



- F_0 is always avg of data vals
  - ↳ $F_0 = \frac{1}{N}\sum_{n=0}^{N-1} f_n$
  - ↳ sometimes called direct current (DC)
- smooth bump had 1 dominant freq/wave so 1 coeff pair (F_1, F_8) had large magnitude
- rough bump had more active freqs
  - ↳ main hump still dominates so F_1, F_8 remain largest

Fourier plots are **symmetric** since data are real

- there's **lack of stdization** so various defns of DFT / IDFT pairs
  - ↳ we use following w/ 0-based indexing:
    - $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$
    - $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$
  - ↳ SciPy + others use:
    - $f_n = \frac{1}{N}\sum_{k=0}^{N-1} F_k W^{nk}$
    - $F_k = \sum_{n=0}^{N-1} f_n W^{-nk}$

fast fourier transform

date 03/12/2025

- direct implementation of $F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$ takes $O(N^2)$ complex FP ops
  - ↳ i.e. 2 nested for loops

```
For k = 0 : N − 1                  //iterate over all k unknown coeffs
    F_k = 0                        //initialize coefficient to zero
    For n = 0 : N − 1              //iterate over all n data values
        F_k += f_n W^{−nk}         //increment by scaled data value
    End
    F_k = F_k / N                  //normalize
End
```

- faster Fourier transform uses divide + conquer strategy
  - ↳ steps:
    1) split full DFT into 2 DFTs of $\frac{1}{2}$ length
    2) repeat recursively
    3) finish at base case of individual #s
  - ↳ if $N \neq 2^m$ for some m, pad initial data w/ 0s
  - ↳ can express full DFT w/ DFTs of 2 new arrs of length $\frac{N}{2}$, where $n \in [0, \frac{N}{2}-1]$:
    - $g_n = \frac{1}{2}(f_n + f_{n+\frac{N}{2}})$
    - $h_n = \frac{1}{2}(f_n - f_{n+\frac{N}{2}})W^{-n}$
      → W^{-n} is W-factor / twiddle factor
  - ↳ $F_{even} = G = DFT(g)$
  - ↳ $F_{odd} = H = DFT(h)$

- FFT derivation:

$$F_k = \frac{1}{N}\sum_{n=0}^{N-1} f_n W^{-nk}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} f_n W^{-nk} + \frac{1}{N}\sum_{n=\frac{N}{2}}^{N-1} f_n W^{-nk}$$

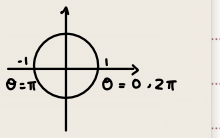
Assume N is even.

Reindex 2nd sum using $m = n - \frac{N}{2}$:

$$F_k = \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} f_n W^{-nk} + \frac{1}{N}\sum_{m=0}^{\frac{N}{2}-1} f_{m+\frac{N}{2}} W^{-k(m+\frac{N}{2})}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} f_n W^{-nk} + \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} f_{n+\frac{N}{2}} W^{-nk} W^{-\frac{kN}{2}}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} (f_n + f_{n+\frac{N}{2}} \cdot W^{-\frac{kN}{2}}) W^{-nk}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} (f_n + f_{n+\frac{N}{2}} \cdot (-1)^k) W^{-nk}$$

← rename m back to n

$W^{-\frac{kN}{2}} = (e^{\frac{2\pi i}{N}})^{-\frac{kN}{2}}$
$= e^{-\pi i k}$
$= (-1)^k$



For $j \in [0, \frac{N}{2}-1]$:

↳ case 1: k is even (+1)

$$F_k = F_{2j} = \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} (f_n + f_{n+\frac{N}{2}}) W^{-2nj}$$

↳ case 2: k is odd (-1)

$$F_k = F_{2j+1}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} (f_n - f_{n+\frac{N}{2}}) W^{-n(2j+1)}$$
$$= \frac{1}{N}\sum_{n=0}^{\frac{N}{2}-1} ((f_n - f_{n+\frac{N}{2}}) W^{-n}) W^{-2nj}$$

twiddle factor

Define 2 new $\frac{1}{2}$-length vectors for $n \in [0, \frac{N}{2}-1]$:

↳ $g_n = \frac{1}{2}(f_n + f_{n+\frac{N}{2}})$

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

Let $M = \frac{N}{2}$. Then:

$$F_{2j} = \frac{2}{N}\sum_{n=0}^{\frac{N}{2}-1} g_n W_N^{-2nj}$$

date

$= \frac{1}{M} \sum_{n=0}^{M-1} g_n W_M^{-nj}$ → DFT(g)

$= G_j$

$F_{2j+1} = \frac{2}{N} \sum_{n=0}^{\frac{N}{2}-1} h_n W_N^{-2nj}$

$= \frac{1}{M} \sum_{n=0}^{M-1} h_n W_M^{-nj}$ → DFT(h)

$= H_j$

↳ we've defined $W_N = e^{\frac{2\pi i}{N}}$, $W_M = e^{\frac{2\pi i}{M}}$ so $W_M = W_N^2$ since $N = 2M$

e.g.

Apply

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

for $n = 0, 1, \ldots, \frac{N}{2} - 1$.

$$f = \begin{bmatrix}2\\2\\1\\3\\2\\2\\1\\3\end{bmatrix} \rightarrow \begin{bmatrix}2\\2\\1\\3\end{bmatrix} = g$$

$$\begin{bmatrix}0\cdot W^0\\0\cdot W^{-1}\\0\cdot W^{-2}\\0\cdot W^{-3}\end{bmatrix} = \begin{bmatrix}0\\0\\0\\0\end{bmatrix} = h$$

$g_0 = \frac{1}{2}(f_0 + f_{0+4}) = \frac{1}{2}(2+2) = 2$

$g_1 = \frac{1}{2}(f_1 + f_5) = \frac{1}{2}(2+2) = 2$

$g_2 = \frac{1}{2}(f_2 + f_6) = \frac{1}{2}(1+1) = 1$

$g_3 = \frac{1}{2}(f_3 + f_7) = \frac{1}{2}(3+3) = 3$

$h_0 = \frac{1}{2}(f_0 - f_{0+4})W^0 = \frac{1}{2}(2-2)W^0 = 0$

$h_1 = \frac{1}{2}(f_1 - f_5)W^{-1} = \frac{1}{2}(2-2)W^0 = 0$

$h_2 = \frac{1}{2}(f_2 - f_6)W^{-2} = \frac{1}{2}(1-1)W^0 = 0$

$h_3 = \frac{1}{2}(f_3 - f_7)W^{-3} = \frac{1}{2}(3-3)W^0 = 0$

DFT of g is $G = \left[2, \frac{1+i}{4}, -\frac{1}{2}, \frac{1-i}{4}\right]$ (e.g. found by definition.).

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

DFT of h is $H = [0, 0, 0, 0]$.

Recover DFT of f by interleaving G & H (even and odd entries, respectively):

$$F = \left[2, 0, \frac{1+i}{4}, 0, -\frac{1}{2}, 0, \frac{1-i}{4}, 0\right]$$

G H

$G_k = \frac{1}{N} \sum_{n=0}^{N-1} g_n W^{-nk}$

$G_0 = \frac{1}{4} \sum_{n=0}^{3} g_n W^{-n\cdot 0}$

$= \frac{1}{4} \sum_{n=0}^{3} g_n (1)$

$= \frac{1}{4}(g_0 + g_1 + g_2 + g_3)$

$= \frac{1}{4}(2+2+1+3)$

$= 2$

$G_1 = \frac{1}{4} \sum_{n=0}^{3} g_n W^{-n}$

$= \frac{1}{4} \sum_{n=0}^{3} (g_0 + g_1 W^{-1} + g_2 W^{-2} + g_3 W^{-3})$ → $W = e^{\frac{2\pi i}{4}} = e^{\frac{\pi i}{2}} = i$

$= \frac{1}{4}(2 + 2(i)^{-1} + 1(i)^{-2} + 3(i)^{-3})$

$= \frac{1}{4}(2 + 2i^3 + i^2 + 3i)$

For G_1, to calc $(i)^{-1}$:

$-1 \bmod 4 = 3$

$(i)^{-1} = i^3$

date

$= \frac{1}{4}(2 - 2i - 1 + 3i)$

$= \frac{1}{4}(i + 1)$

$G_2 = \frac{1}{4}\sum_{n=0}^{3} g_n W^{-2n}$

$= \frac{1}{4}(2 + 2(i)^{-2} + 1(i)^{-4} + 3(i)^{-6})$

$= \frac{1}{4}(2 + 2i^2 + 1 + 3i^2)$

$= \frac{1}{4}(2 - 2 + 1 - 3)$

$= -\frac{1}{2}$

$G_3 = \frac{1}{4}\sum_{n=0}^{3} g_n W^{-3n}$

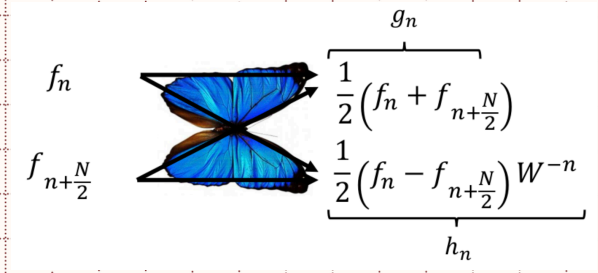
$= \frac{1}{4}(2 + 2(i)^{-3} + 1(i)^{-6} + 3(i)^{-9})$

$= \frac{1}{4}(2 + 2i + i^2 + 3i^3)$

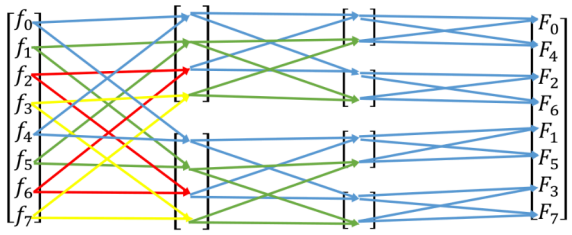
$= \frac{1}{4}(2 + 2i - 1 - 3i)$

$= \frac{1}{4}(1 - i)$

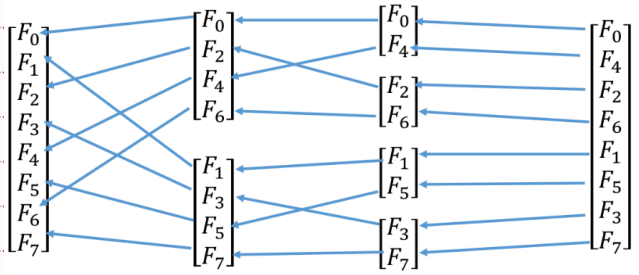
- butterfly op: can think of each step as taking pair of #s + producing 2 outputs



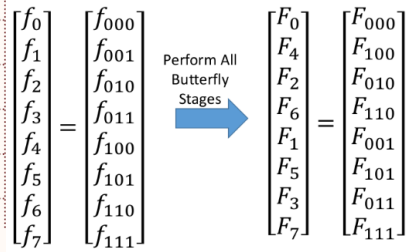
↳ e.g. recursive butterfly FFT algo w/ $N = 8 = 2^3$ (3 recursive stages)



- each step applies even-odd index splitting for result location so output is in "bit-reversed" positions



- unscramble coeffs as final step



date

- FFT algo analysis:
  - ↳ $O(N)$ ops to split arr in 2 at each stage
  - ↳ data has / padded to len $N = 2^m$ so $m = \log_2 N$ stages
  - ↳ total cost $O(N \log_2 N)$
    - cheaper than $O(N^2)$

since DFT & IDFT have similar forms, same core butterfly algo can perform either dir w/ minor changes using generic expr: $f'_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n U^{nk}$

- ↳ fwd DFT: $U = W^{-1}$
  - $f'_k = F_k$
  - $f_n = f_n$
  - $F_k = \frac{1}{N} \sum_{n=0}^{N-1} f_n W^{-nk}$
- ↳ IDFT: $U = W$ + post-multiply by N
  - $f'_k = f_n$
  - $f_n = F_k$
  - $f_n = \sum_{k=0}^{N-1} F_k W^{nk}$

- e.g. complete butterfly

$$\begin{bmatrix}1\\2\\3\\4\\1\\2\\3\\4\end{bmatrix} \xrightarrow[W = e^{\frac{\pi i}{4}}]{N=8} \begin{bmatrix}1\\2\\3\\4\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \xrightarrow[W = e^{\frac{\pi i}{2}}]{N=4} \begin{bmatrix}2\\3\end{bmatrix}, \begin{bmatrix}-1\\i\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix} \xrightarrow[W = e^{\pi i}]{N=2} \begin{bmatrix}2.5\\-0.5\\-0.5+0.5i\\-0.5-0.5i\\0\\0\\0\\0\end{bmatrix} \xrightarrow{\text{Unscramble!}} \begin{bmatrix}2.5\\0\\-0.5+0.5i\\0\\-0.5\\0\\-0.5-0.5i\\0\end{bmatrix}$$

- e.g. detailed FFT

Find the sequence of vectors and the result of applying the butterfly FFT approach to the data:

$$f = \begin{bmatrix}4\\3\\2\\1\\4\\3\\2\\1\end{bmatrix}$$

$f = \begin{bmatrix}4\\3\\2\\1\\4\\3\\2\\1\end{bmatrix} \xrightarrow{N=8} \begin{bmatrix}4\\3\\2\\1\end{bmatrix}, \begin{bmatrix}0\\0\\0\\0\end{bmatrix} \xrightarrow{N=4} \begin{bmatrix}3\\2\end{bmatrix}, \begin{bmatrix}1\\-i\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}0\\0\end{bmatrix} \xrightarrow{N=2} \begin{bmatrix}2.5\\0.5\\0.5-0.5i\\0.5+0.5i\\0\\0\\0\\0\end{bmatrix} \xrightarrow{\text{unscramble}} \begin{bmatrix}2.5\\0\\0.5-0.5i\\0\\0.5\\0\\0.5+0.5i\\0\end{bmatrix}$

Stage 1: $N = 8$, $W = e^{\frac{2\pi i}{8}} = e^{\frac{\pi i}{4}}$

$g = \frac{1}{2}\begin{bmatrix}4\\3\\2\\1\end{bmatrix} + \frac{1}{2}\begin{bmatrix}4\\3\\2\\1\end{bmatrix}$

$= \begin{bmatrix}4\\3\\2\\1\end{bmatrix}$

$h_0 = \frac{1}{2}(4-4)W^0 = 0$

$h_1 = \frac{1}{2}(3-3)W^{-1} = 0$

$h_2 = \frac{1}{2}(2-2)W^{-2} = 0$

$h_3 = \frac{1}{2}(1-1)W^{-3} = 0$

$h = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

Stage 2: $N = 4$, $W = e^{\frac{2\pi i}{4}} = e^{\frac{\pi i}{2}} = i$

↳ top vectors:

$g_0 = \frac{1}{2}(4+2) = 3$

$g_1 = \frac{1}{2}(3+1) = 2$

$h_0 = \frac{1}{2}(4-2)W^0 = 1$

$h_1 = \frac{1}{2}(3-1)W^{-1}$

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

$= -i$

$g_{top} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$

$h_{top} = \begin{bmatrix} 1 \\ -i \end{bmatrix}$

↳ bottom vectors are all 0

Stage 3: $N = 2$, $W = e^{\frac{2\pi i}{2}} = e^{\pi i} = -1$

↳ top pair:

$g_0 = \frac{1}{2}(3+2) = 2.5$

$h_0 = \frac{1}{2}(3-2)W^0 = 0.5$

↳ bottom pair:

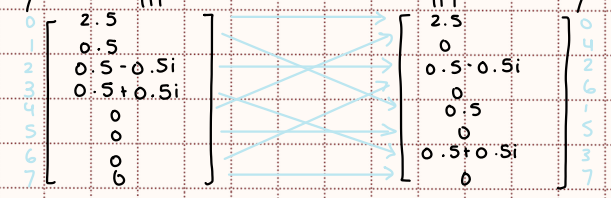
$g_0 = \frac{1}{2}(1-i) = 0.5 - 0.5i$

$h_0 = \frac{1}{2}(1-(-i)) = 0.5 + 0.5i$

↳ rest are 0

Unscramble stage:

Original indices:		Bit reversed indices:	
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7



↳ conjugate symmetry due to input being real

e.g. smaller, complex FFT

Find the sequence of vectors and the final result of applying the butterfly FFT approach to the data:

$$f = \begin{bmatrix} 2+i \\ 3 \\ -2 \\ -2i \end{bmatrix}$$

date

$$f = \begin{bmatrix} 2+i \\ 3 \\ -2 \\ -2i \end{bmatrix} \xrightarrow{N=4} \begin{bmatrix} i/2 \\ 3/2 - i \\ 2 + i/2 \\ 1 - 3/2\,i \end{bmatrix} \xrightarrow{N=2} \begin{bmatrix} 3/4 - i/4 \\ -3/4 + 3i/4 \\ 3/2 - i/2 \\ 1/2 + i \end{bmatrix} \xrightarrow{\text{unscramble}} \begin{bmatrix} 3/4 - i/4 \\ 3/2 - i/2 \\ -3/4 + 3i/4 \\ 1/2 + i \end{bmatrix}$$

Stage 1: $N=4$, $W = e^{\frac{2\pi i}{4}} = e^{\frac{\pi i}{2}} = i$

$g_0 = \frac{1}{2}(2+i-2) = \frac{i}{2}$

$g_1 = \frac{1}{2}(3-2i) = \frac{3}{2} - i$

$h_0 = \frac{1}{2}(2+i+2)W^0 = 2 + \frac{i}{2}$

$h_1 = \frac{1}{2}(3+2i)W^{-1}$

$= \frac{1}{2}(3+2i)(i)^{-1}$

$= \frac{1}{2}(3+2i)(i)^3$

$= 1 - \frac{3}{2}i$

Stage 2: $N=2$, $W = e^{\frac{2\pi i}{2}} = e^{\pi i} = -1$

↳ top:

$g_0 = \frac{1}{2}\left(\frac{i}{2} + \frac{3}{2} - i\right) = \frac{3}{4} - \frac{i}{4}$

$h_0 = \frac{1}{2}\left(\frac{i}{2} - \frac{3}{2} + i\right) = -\frac{3}{4} + \frac{3i}{4}$

↳ bottom:

$g_0 = \frac{1}{2}\left(2 + \frac{i}{2} + 1 - \frac{3}{2}i\right) = \frac{3}{2} - \frac{i}{2}$

$h_0 = \frac{1}{2}\left(2 + \frac{i}{2} - 1 + \frac{3}{2}i\right) = \frac{1}{2} + i$

Unscramble:

Original:		New:	
0	00	00	0
1	01	10	2
2	10	01	1
3	11	11	3

$F = \begin{bmatrix} 3/4 - i/4 \\ 3/2 - i/2 \\ -3/4 + 3i/4 \\ 1/2 + i \end{bmatrix}$

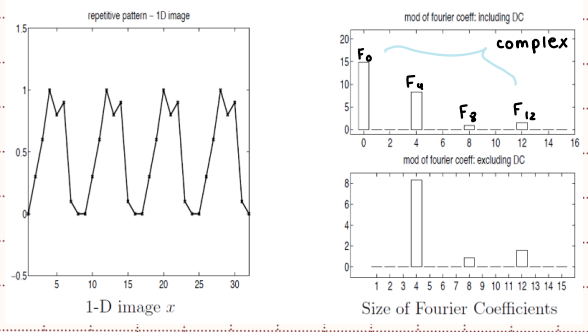
fourier transforms
images & aliasing

date 03/14/2025

IMAGE DATA / COMPRESSION

e.g think of 1D arr of grayscale vals as 1D img
↳ each entry gives pixels' intensity/gray vals

↳ for grayscale imgs, perform DFT on pixels' intensity/gray vals

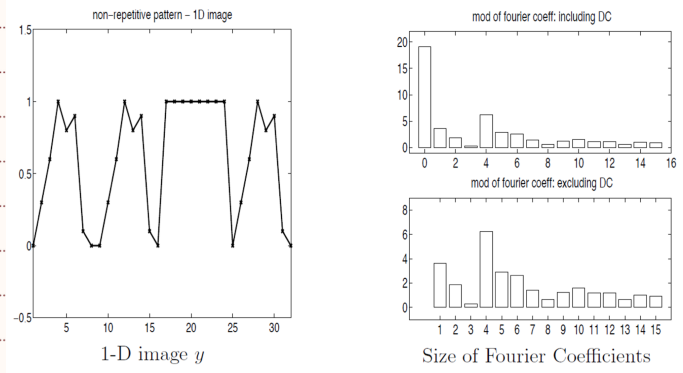


→ 7 real #s

- plot is only first 16 coeffs, since real data implies conjugate symmetry

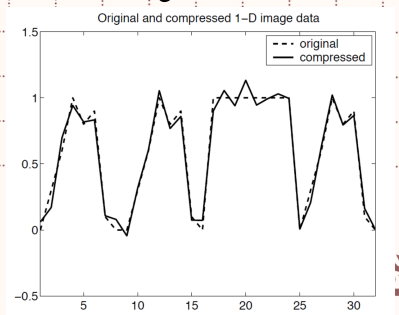
↳ avg is apx 0.5 so $F_0 \simeq 0.5$ (~15 since plot uses Jupyter's convention)
↳ pattern has few coeffs active
- store it cheaply w/ F_0, F_4, F_8, F_{12}

↳ if replace 3rd bump's top w/ flat line, simple repetition is destroyed + many more Fourier coeffs will become active/non-zero



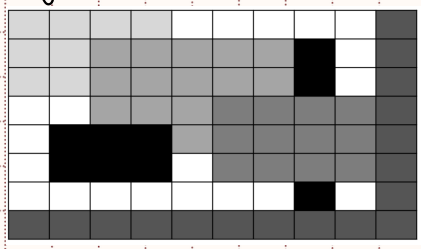
- more expensive
- many coeffs have small moduli so contribute less to img

↳ compression strat:
- create compressed version of img f_n by throwing away small Fourier coeffs
  → $|F_k| < tol$
- to reconstruct img, run IDFT to get modified data $\hat{f}_n$
- discard imaginary parts of $\hat{f}_n$ to ensure it's strictly real
- recovered img has small deviations from og data, but we store fewer Fourier coeffs



date

- for DFT for 2D grayscale img data, think of 2D arr X of per-pixel intensities, of size $M \times N$
  - ↳ $0 \leq X(i,j) \leq 1$
  - ↳ e.g.



  - ↳ apply 2D extension of DFT defn to data: $F_{k,\ell} = \frac{1}{NM} \sum_{n=0}^{N-1} \sum_{j=0}^{M-1} f_{n,j} W_N^{-nk} W_M^{-j\ell}$ (roots of unity)
    - 2 std DFTs combined
    - 2 distinct roots of unity, 1 per dimension:
      - → $W_N = e^{\frac{2\pi i}{N}}$
      - → $W_M = e^{\frac{2\pi i}{M}}$
    - result is 2D arr of Fourier coeffs $F_{k,\ell}$
- 2D FFT computed efficiently using nested 1D FFTs
  1) transform each row using 1D FFTs
  2) transform each col of result
  - ↳ derivation:

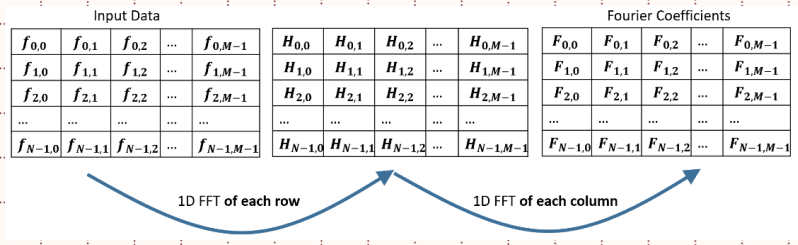
$$F_{k,\ell} = \frac{1}{MN} \sum_{n=0}^{N-1} \sum_{j=0}^{M-1} f_{n,j} W_N^{-nk} W_M^{-j\ell}$$
$$= \frac{1}{N} \sum_{n=0}^{N-1} W_N^{-nk} \left(\frac{1}{M} \sum_{j=0}^{M-1} f_{n,j} W_M^{-j\ell} \right)$$

(1D FFT per row to get H)

$$H_{n,\ell} = \frac{1}{M} \sum_{j=0}^{M-1} f_{n,j} W_M^{-j\ell}$$
$$F_{k,\ell} = \frac{1}{N} \sum_{n=0}^{N-1} H_{n,\ell} W_N^{-nk}$$

(1D FFT per col on H to get F)

  - ↳ visually:



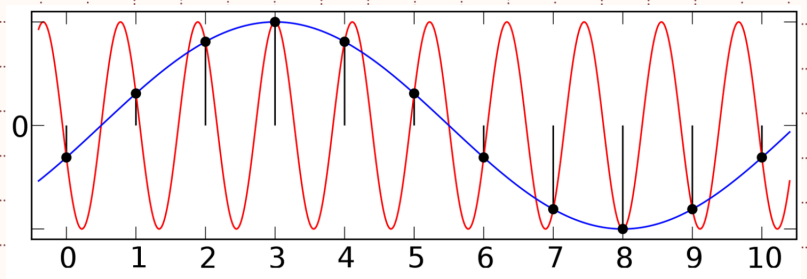
  - ↳ complexity:
    - M 1D FFTs of len N: $M \cdot O(N \log_2 N) = O(MN \log_2 N)$
    - N 1D FFTs of len M: $N \cdot O(M \log_2 M) = O(MN \log_2 M)$
    - total is $O(MN(\log_2 M + \log_2 N))$
- block-based compression: img is often subdivided into smaller blocks (e.g. 16×16 or 8×8 px)
  + compression is performed on each block independently
  - ↳ pixels within block often have similar data so hopefully compress more effectively

ALIASING

- if real input signal contains high freqs but spacing of data pts is inadequate, aliasing can occur

date

↳ e.g. high freq aliases to be lower freq in result



mathematical analysis:

↳ Fourier series of continuous series was $f(t) = \sum_{k=-\infty}^{\infty} C_k e^{\frac{i2\pi kt}{T}}$ for period T

↳ if we sample this true signal at pts $t_n = n\Delta t = n\frac{T}{N}$, then $f_n = f(t_n) = \sum_{k=-\infty}^{\infty} C_k e^{\frac{i2\pi kn}{N}} = \sum_{n=-\infty}^{\infty} C_k W^{nk}$

- exact for arbitrarily high freq
- C_k are true Fourier series coeffs for f since $k \in (-\infty, \infty)$

↳ compare w/ IDFT apx: $f_n = \sum_{k=-\frac{N}{2}+1}^{N/2} F_k W^{nk}$

- same as $k \in [0, N-1]$ by periodicity of F_k
- only have N coeffs F_k
- how do F_k, C_k relate?

DFT gives $F_\ell = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{N/2} f_n W^{-n\ell}$ by defn.

Plug in true f_n expansion in terms of C_k:

$$F_\ell = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{N/2} W^{-n\ell} \sum_{k=-\infty}^{\infty} C_k W^{nk}$$
$$= \sum_{k=-\infty}^{\infty} \frac{C_k}{N} \sum_{n=-\frac{N}{2}+1}^{N/2} W^{n(k-\ell)}$$

$W^{n(k-\ell)}$ hints at orthogonality, but need to adapt identity to handle $k \in (-\infty, \infty)$ instead of only $k \in [0, N-1]$.

New identity: $\sum_{p=-\frac{N}{2}+1}^{N/2} W^{p(k-\ell)} = N(\delta_{k,\ell} + \delta_{k,\ell+N} + \delta_{k,\ell-N} + \delta_{k,\ell+2N} + \delta_{k,\ell-2N} + \dots)$

→ when $\ell = k + jN$ (i.e. $\ell \equiv k \pmod N$), then $W^{p(\ell-k)} = W^{pjN} = (W^N)^{pj} = 1^{pj} = 1$

Using extended orthogonality identity, we have DFT coeffs:

$F_\ell = \sum_{k=-\infty}^{\infty} \frac{C_k}{N} \cdot N(\delta_{k,\ell} + \delta_{k,\ell+N} + \delta_{k,\ell-N} + \dots)$

→ when $k = \ell \to \delta_{\ell,\ell} = 1 \to F_\ell = \sum_{k=\ell} C_k(\delta_{\ell,\ell} + 0 + 0 + \dots) = C_\ell$

$F_\ell = C_\ell + C_{\ell+N} + C_{\ell-N} + C_{\ell+2N} + C_{\ell-2N} + \dots$

→ DFT coeffs are sums of true Fourier series coeffs, including arbitrarily high freq contributions

→ high freqs $C_k \notin [-\frac{N}{2}+1, \frac{N}{2}]$ alias as (i.e. get added to) low freq F_k for $k \in [-\frac{N}{2}+1, \frac{N}{2}]$

→ once signal is discretely sampled, no way to distinguish aliased freqs from real ones

· 2 partial solns:

↳ inc sampling resolution to capture higher freqs

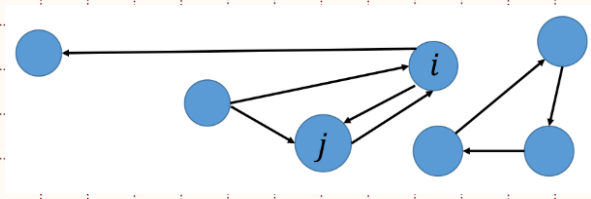
- sampling freq should be 2x higher than largest freq present in signal

↳ filter before sampling (i.e. digitizing) to remove too high freqs

numerical linear algebra & page rank

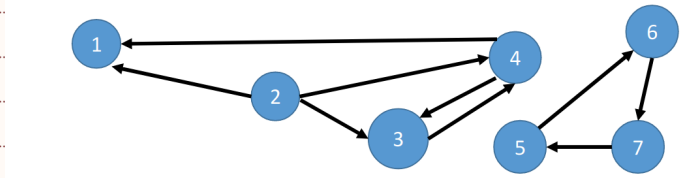
date 03/22/2025

· rough idea of Google pg rank algo: more web pgs that link to a website = more important it is
· link struct btwn pgs provides useful indicator
↳ use directed graph to model web's struct



- nodes rep pgs
- arcs rep links
- deg refers to node's outdeg (i.e. # arcs leaving node)
  → deg(j) = 1

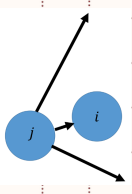
↳ to store directed graph, use adjacency matrix G
- $G_{ij} = \begin{cases} 1 & \text{, if link } j \to i \text{ exists} \\ 0 & \text{, otherwise} \end{cases}$
- deg of node q is sum of entries in col q
- G isn't necessarily symmetric abt diagonal
- e.g.



Question: What is the adjacency matrix of this graph?

	1	2	3	4	5	6	7
1	0	1	0	1	0	0	0
2	0	0	0	0	0	0	0
3	0	1	0	1	0	0	0
4	0	1	1	0	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0

↳ if pg j links to pg i, it's a vote by j that i's important
- outgoing links have equal influence so importance j gives to i is $\frac{1}{\deg(j)}$
  → j gives $\frac{1}{3}$ vote to i



↳ global importance:
- if pg i has many incoming links, probably important
- if pg i has 1 incoming link from pg j (j has many incoming links), then i is fairly important too

date

RANDOM SURFER MODEL

- if user starts at pg + follows links at random from pg to pg for K steps, they'll probably end up on important pgs more often
  - ↳ select new start pg + follow K random links again
  - ↳ repeat process R times
  - ↳ overall importance is estimated as $\text{Rank}(\text{pg } i) = \frac{\text{visits to pg } i}{\text{total visits to all pgs } (R \cdot K)}$
- ranker surfer algo:

$Rank(m) = 0,\ m = 1, \ldots, R$
For $m = 1, \ldots, R$
  $j = m$
  For $k = 1, \ldots, K$
    $Rank(j) = Rank(j) + 1$
    Randomly select outlink l of page j
    $j = l$
  EndFor
EndFor
$Rank(m) = Rank(m)/(K \times R),\ m = 1, \ldots R$

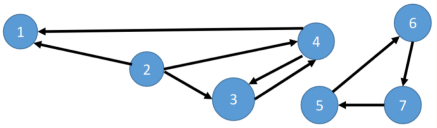
  - ↳ potential issues:
    - # real web pgs is huge so many iterations (i.e. large K, R) needed
    - #steps taken per random surf seq must be large to get representative sample
    - dead end links (i.e. stuck on 1 pg)
    - cycles (i.e. stuck on closed subset of pgs)

MARKOV CHAIN MATRIX

- let P be large matrix of probabilities, where P_{ij} is probability of randomly transitioning from pg j to i
  - ↳ $P_{ij} = \begin{cases} \frac{1}{\deg(j)} & \text{, if link } j \to i \text{ exists} \\ 0 & \text{, otherwise} \end{cases}$
  - ↳ can build P from adjacency matrix G by dividing all entries of each col of G by col sum (i.e. out-deg of node)
    - e.g.

From

To	1	2	3	4	5	6	7
1	0	1/3	0	1/2	0	0	0
2	0	0	0	0	0	0	0
3	0	1/3	0	1/2	0	0	0
4	0	1/3	1	0	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0



$G = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ (rows 1–7)

add: 0 3 1 2 1 1 1

$P = \begin{bmatrix} 0 & 1/3 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1/3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$ (rows 1–7)

- to deal w/ dead ends, teleport to new pg at random
  - ↳ define col vector d st $d_i = \begin{cases} 1 & \text{, if } \deg(i) = 0 \text{ (i.e. col of 0s)} \\ 0 & \text{, otherwise} \end{cases}$

date

↳ define $e = [1, 1, \ldots, 1]^T$ to be col vector of 1s

↳ if R is # pgs, augment P to get $P' = P + \frac{1}{R} ed^T$

- d^T is row vector

↳ e.g.

$$G = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \longrightarrow P = \begin{bmatrix} 0 & 1/3 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix}$$

$$d = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}, \quad \frac{1}{R} = \frac{1}{4}$$

$$P' = P + \frac{1}{R} ed^T$$

$$= \begin{bmatrix} 0 & 1/3 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} [1 \ 0 \ 1 \ 0]$$

$$= \begin{bmatrix} 0 & 1/3 & 0 & 0 \\ 0 & 1/3 & 0 & 1/2 \\ 0 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 1/2 \end{bmatrix} + \frac{1}{4} \begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$

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

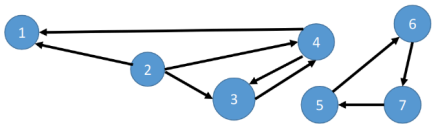
- no dead ends

↳ matrix $\frac{1}{R} ed^T$ is matrix of probabilities st from any dead end pg (i.e. $d_i = 1$), transition to every other pg w/equal probability

- e.g.

From

To	1	2	3	4	5	6	7
1	0	1/3	0	1/2	0	0	0
2	0	0	0	0	0	0	0
3	0	1/3	0	1/2	0	0	0
4	0	1/3	1	0	0	0	0
5	0	0	0	0	0	0	1
6	0	0	0	0	1	0	0
7	0	0	0	0	0	1	0



- to escape closed cycle of pgs, follow links randomly via P' most of time (frac α) + teleport from any pg to any other pg w/some small probability of the time $(1 - \alpha)$

↳ $M = \alpha P' + (1 - \alpha) \frac{1}{R} ee^T$

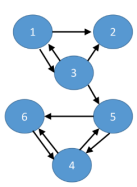
- $\frac{1}{R} ee^T$ matrix:

$$\begin{bmatrix} 1/R & 1/R & \cdots & 1/R \\ 1/R & 1/R & \cdots & 1/R \\ \vdots & \vdots & \ddots & \vdots \\ 1/R & 1/R & \cdots & 1/R \end{bmatrix}$$

→ teleport from 1 pg to another w/equal probability regardless of links

- aka Google matrix
  - → $\alpha \approx 0.85$

- e.g.



$$P = \begin{bmatrix} 0 & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 \\ 0 & 0 & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} \quad d = [0, 1, 0, 0, 0, 0].$$

(Page 2 is a dead end!)

Random Surfing
(but gets stuck in dead ends)

$$P' = P + \frac{1}{6}ed^T = \begin{bmatrix} 0 & \frac{1}{6} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{6} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{6} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{6} & 0 & 0 & \frac{1}{2} & 1 \\ 0 & \frac{1}{6} & \frac{1}{3} & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{6} & 0 & \frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix} M = \begin{bmatrix} \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6} & \frac{1}{6}\alpha+\frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha \\ \frac{1}{3}\alpha+\frac{1}{6} & \frac{1}{6} & \frac{1}{6}\alpha+\frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha \\ \frac{1}{3}\alpha+\frac{1}{6} & \frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha \\ \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{3}\alpha+\frac{1}{6} & \frac{5}{6}\alpha+\frac{1}{6} \\ \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6} & \frac{1}{6}\alpha+\frac{1}{6} & \frac{1}{3}\alpha+\frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6}-\frac{1}{6}\alpha \\ \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha & \frac{1}{3}\alpha+\frac{1}{6} & \frac{1}{3}\alpha+\frac{1}{6} & \frac{1}{6}-\frac{1}{6}\alpha \end{bmatrix}$$

Add Teleportation out of Dead Ends (fills in empty cols)

Add Occasional Random Teleportation to Also Escape Cycles

$$M = \alpha P' + (1-\alpha)\frac{1}{R}ee^T$$

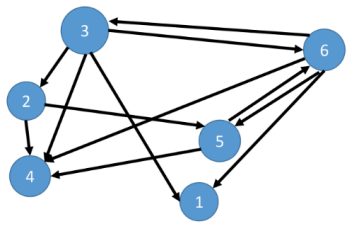
↳ for $\alpha = 0.85$:

$$M = \begin{bmatrix} \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{9}{20} & \frac{7}{8} \\ \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{9}{20} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{9}{20} & \frac{9}{20} & \frac{1}{40} \end{bmatrix}.$$

∘ sum of each col is 1

· e.g.

Construct the google matrix $M = \alpha\left(P + \frac{1}{R}ed^T\right) + (1-\alpha)\frac{1}{R}ee^T$ for the small web shown here, using $\alpha = \frac{1}{2}$, and $R = 6$ pages.



Recall:

$$P_{ij} = \begin{cases} \frac{1}{\deg(j)}, & \text{if link } j \to i \text{ exists} \\ 0, & \text{otherwise} \end{cases}$$

$$d_i = \begin{cases} 1, & \text{if } deg(i) = 0 \\ 0, & \text{otherwise} \end{cases}$$

$e = [1,1,1,\ldots,1]^T$

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

$$\begin{matrix} 0 & 2 & 4 & 0 & 2 & 4 \end{matrix}$$

$$P = \begin{bmatrix} 0 & 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{4} & 0 & \frac{1}{2} & \frac{1}{4} \\ 0 & \frac{1}{2} & 0 & 0 & 0 & \frac{1}{4} \\ 0 & 0 & \frac{1}{4} & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

$$d = [1 \;\; 0 \;\; 0 \;\; 1 \;\; 0 \;\; 0]^T$$

$$e = [1 \;\; 1 \;\; 1 \;\; 1 \;\; 1 \;\; 1]^T$$

$$\frac{1}{R} = \frac{1}{6}, \quad \alpha = \frac{1}{2}$$

$$M = \alpha\left(P + \frac{1}{R}ed^T\right) + (1-\alpha)\frac{1}{R}ee^T$$

$$\alpha\left(P + \frac{1}{R}ed^T\right) = \frac{1}{2}\begin{bmatrix} \frac{1}{6} & 0 & \frac{1}{4} & \frac{1}{6} & 0 & \frac{1}{4} \\ \vdots & 0 & \frac{1}{4} & \vdots & 0 & 0 \\ & 0 & 0 & & 0 & \frac{1}{4} \\ & \frac{1}{2} & \frac{1}{4} & & \frac{1}{2} & \frac{1}{4} \\ & \frac{1}{2} & 0 & & 0 & \frac{1}{4} \\ \frac{1}{6} & 0 & \frac{1}{4} & \frac{1}{6} & \frac{1}{2} & 0 \end{bmatrix} = \begin{bmatrix} \frac{1}{12} & 0 & \frac{1}{8} & \frac{1}{12} & 0 & \frac{1}{8} \\ \vdots & 0 & \frac{1}{8} & \vdots & 0 & 0 \\ & 0 & 0 & & 0 & \frac{1}{8} \\ & \frac{1}{4} & \frac{1}{8} & & \frac{1}{4} & \frac{1}{8} \\ & \frac{1}{4} & 0 & & 0 & \frac{1}{8} \\ \frac{1}{12} & 0 & \frac{1}{8} & \frac{1}{12} & \frac{1}{4} & 0 \end{bmatrix}$$

date

$(1-\alpha)\frac{1}{R}ee^T = \frac{1}{2}\left(\frac{1}{6}\right)\begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix} = \begin{bmatrix} 1/12 & \cdots & 1/12 \\ \vdots & \ddots & \vdots \\ 1/12 & \cdots & 1/12 \end{bmatrix}$

$$M = \begin{bmatrix} 1/6 & 1/12 & 5/24 & 1/6 & 1/12 & 5/24 \\ & 1/12 & 5/24 & & 1/12 & 1/12 \\ & 1/12 & 1/12 & & 1/12 & 5/24 \\ & 1/3 & 5/24 & & 1/3 & 5/24 \\ & 1/3 & 1/12 & & 1/12 & 5/24 \\ 1/6 & 1/12 & 5/24 & 1/6 & 1/3 & 1/12 \end{bmatrix}$$

- useful props of M:
  - ↳ entries of M satisfy $0 \le M_{ij} \le 1$
  - ↳ each col of M sums to 1
    - $\sum_{i=1}^{R} M_{ij} = 1$
- define Markov matrix Q by above 2 props:
  - ↳ $0 \le Q_{ij} \le 1$
  - ↳ $\sum_{i=1}^{R} Q_{ij} = 1$
  - ↳ Google matrix M is Markov matrix
- define probability vector q st $0 \le q_i \le 1$, $\sum_{i=1}^{R} q_i = 1$
  - ↳ e.g. if surfer starts at random pg w/equal probabilities, can be rep by probability vector p where $p_i = \frac{1}{R}$
    - $[\frac{1}{R}, \frac{1}{R}, \ldots, \frac{1}{R}]$
- given probability vector p^0 describing initial state + Markov matrix M describing transition probabilities among pgs, product Mp^0 tells probabilities of surfer being at each pg after 1 transition
  - ↳ $p' = Mp^0$
  - ↳ likewise, $p^{n+1} = Mp^n$
  - ↳ e.g.

$p^0 = [1,0,0,0]^T$. (We're *definitely* starting on page 1.)

If we had a google/transition matrix $M = \begin{bmatrix} 1/3 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 0 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \end{bmatrix}$

Then after one step, what is $p^1 = Mp^0$? And what does it mean?

$$p' = \begin{bmatrix} 1/3 \\ 0 \\ 1/3 \\ 1/3 \end{bmatrix}$$

- apx 33% chance being at pg 1, 3, or 4 after 1 step starting from pg 1

  - ↳ e.g.

$p^0 = \left[\frac{1}{2}, 0, \frac{1}{2}, 0\right]^T$. (We're on page 1 or 3 with probability 0.5 each.)

If we have same matrix $M = \begin{bmatrix} 1/3 & 0 & 0 & 1/2 \\ 0 & 1/2 & 0 & 0 \\ 1/3 & 1/2 & 0 & 0 \\ 1/3 & 0 & 1 & 1/2 \end{bmatrix}$,

then after one step, what is p^1?

$$p' = \begin{bmatrix} 1/6 \\ 0 \\ 1/6 \\ 1/6 + 1/2 \end{bmatrix} = \begin{bmatrix} 1/6 \\ 0 \\ 1/6 \\ 2/3 \end{bmatrix}$$

- if p^n is probability vector, then $p^{n+1} = Mp^n$ is also one (i.e. probability vector is preserved)

date

↳ prove $\sum_i p_i^{n+1} = 1$:

$$\sum_i p_i^{n+1} = \sum_i \sum_j M_{ij} p_j^n$$
$$= \sum_j (p_j^n \sum_i M_{ij})$$
$$= \sum_j (p_j^n \cdot 1)$$
$$= \sum_j p_j^n$$
$$= 1$$

Both M, p^n are probability matrices.

↳ prove $0 \le p_i^{n+1} \le 1$:

- $p_i^{n+1} \ge 0$ b/c it's sums + products of probabilities ≥ 0
- $p_i^{n+1} \le 1$ b/c $\sum_i p_i^{n+1} = 1$ + we just proved $p_i^{n+1} \ge 0$

· pg rank asks w/ what probability does surfer end up at each pg after many steps, starting from $p^0 = \frac{1}{R} e$?

↳ i.e. what's $p^\infty = \lim_{k \to \infty} (M)^k p^0$? (exp, not step index)

- higher probability in p^∞ means greater importance

· pg rank algo summary:

1) given graph of network, compute Google transition matrix
   - $M = \alpha(P + \frac{1}{R} e d^T) + (1-\alpha)\frac{1}{R} e e^T$
2) repeatedly evolve probability vector p^i via $p^{n+1} = Mp^n$ twd steady state, apx random surfer
3) site w/ highest probability of being visited is considered most important

· e.g.

Starting from $p^0 = \frac{1}{R} e$, repeated multiplication by M gives a sequence of probability vectors, eventually settling down.

$$\underset{p^0}{\begin{bmatrix} 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \\ 0.16667 \end{bmatrix}}, \underset{Mp^0}{\begin{bmatrix} 0.09583 \\ 0.16666 \\ 0.11944 \\ 0.26111 \\ 0.16666 \\ 0.19027 \end{bmatrix}}, \underset{M^2p^0}{\begin{bmatrix} 0.08245 \\ 0.12318 \\ 0.08934 \\ 0.28118 \\ 0.19342 \\ 0.23041 \end{bmatrix}}, \underset{M^3p^0}{\begin{bmatrix} 0.06776 \\ 0.10280 \\ 0.07749 \\ 0.32051 \\ 0.18726 \\ 0.24415 \end{bmatrix}}, \ldots, \underset{M^{10}p^0}{\begin{bmatrix} 0.05205 \\ 0.07428 \\ 0.05782 \\ 0.34797 \\ 0.19975 \\ 0.26810 \end{bmatrix}}$$

Page #	Ranking
1	6
2	4
3	5
4	1
5	3
6	2

For earlier example, the pages are ranked as: $4, 6, 5, 2, 3, 1$ based on these final probabilities.

EFFICIENT PAGE RANK

· to make pg rank computationally feasible, use precomputation + sparsity

· ranking vector p^∞ is precomputed once + stored, independent of any specific query

↳ Google finds only subset of pgs matching keywords + ranks them by vals in precomputed p^∞

· in numerical lin alg, often deal w/ 2 kinds of matrices:

↳ dense: most / all entries are non-zero
- store in N x N matrix + manipulate normally

↳ sparse: most entries are 0
- use sparse data struct to save space + time
- prefer algos that avoid destroying sparsity (i.e. fill in 0 entries)

- to efficiently multiply sparse matrix w/ vector, only access non-zero matrix entries
  - ↳ e.g.

$$\begin{pmatrix} 2 & & & 1 & \\ & & & & -2 \\ & & 1 & & \\ & 3 & & & \\ & & & & \\ 1 & & & & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} = \begin{pmatrix} 2\cdot 1 + 1\cdot 4 \\ -2\cdot 5 \\ 1\cdot 3 \\ 3\cdot 2 \\ 0 \\ 1\cdot 1 + 1\cdot 5 \end{pmatrix} = \begin{pmatrix} 6 \\ -10 \\ 3 \\ 6 \\ 0 \\ 6 \end{pmatrix}$$

    - only 7 multiplications in total (vs naive 30)
- although Google matrix M was fully dense, use lin alg manipulations to perform main iteration $p^{n+1} = Mp^n$ w/o ever creating/storing M
  - ↳ $M = \alpha(P + \frac{1}{R} e d^T) + \frac{1-\alpha}{R} e e^T$
    - sparse b/c not all pgs linked tgt
    - dense in dead end cols
    - fully dense
  - ↳ consider computing $Mp^n = \alpha P p^n + \frac{\alpha}{R} e d^T p^n + \frac{1-\alpha}{R} e e^T p^n$ (1) (2) (3)
    - (1) is sparse matrix/vector multiply
    - (3) involves $ee^T p^n = e(e^T p^n)$
      - → $e^T p^n$ is dot product, which results in scalar
      - → $e^T p^n = [1, \ldots, 1] \begin{bmatrix} \\ \\ \\ \end{bmatrix} = 1$
      - → result is 1 b/c p^n is probability vector
      - → simply use $\frac{1-\alpha}{R} e$ for (3)
    - (2) is similar w/ $\frac{\alpha}{R}(d^T p^n)e$ ($d^T p^n$: scalar)
    - p^{n+1} is sum of 3 vectors w/no dense matrix/vector multiplication
- given above efficient/sparse iteration, loop until max change in probability vector per step is small (i.e. < tol)
  - ↳ pg rank algo:

Page Rank Algorithm

$\mathbf{p}^0 = \mathbf{e}/R$
For $k = 1, \ldots,$ until converged
  $\mathbf{p}^k = M\mathbf{p}^{k-1}$
  If $\max_i |[\mathbf{p}^k]_i - [\mathbf{p}^{k-1}]_i| < tol$ then quit (7.7.1)
EndFor

- pg rank can be tweaked to incorporate other elmts
  - ↳ replace std teleportation $\frac{1-\alpha}{R} e e^T$ w/ $(1-\alpha) v e^T$
    - special probability vector v places extra weight on some sites

EIGENVALUES & EIGENVECTORS

- eigenvalue λ + corresponding eigenvector $\vec{x}$ of matrix Q satisfy $Q\vec{x} = \lambda\vec{x} = \lambda I\vec{x}$
  - ↳ rearranging gives $(\lambda I - Q)\vec{x} = 0$
    - $\lambda I - Q$ must be singular for $\lambda, \vec{x}$ to be pair since we want $\vec{x} \neq \vec{0}$
- singular matrix A satisfies $\det(A) = 0$

date

- to find eigenvals λ of Q, solve characteristic polynomial: $\det(\lambda I - Q) = 0$
  - ↳ e.g.

Example

Find the eigenvalues/eigenvectors of $\begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix}$

$\det(\lambda I - Q) = 0$

$\det(\lambda I - Q) = \det \begin{bmatrix} \lambda - 2 & 2 \\ 5 & \lambda + 1 \end{bmatrix}$

$0 = (\lambda - 2)(\lambda + 1) - 10$

$0 = \lambda^2 + \lambda - 2\lambda - 2 - 10$

$0 = \lambda^2 - \lambda - 12$

$0 = (\lambda - 4)(\lambda + 3)$

$\lambda_1 = 4, \ \lambda_2 = -3$

To find corresponding eigenvectors, plug λ back in $Q\vec{x} = \lambda\vec{x}$:

$\lambda_1 = 4$:

$\begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4x_1 \\ 4x_2 \end{bmatrix}$

↳ row 1: $2x_1 + 2x_2 = 4x_1$

$x_1 = x_2$

↳ row 2: $5x_1 - x_2 = 4x_1$

$x_1 = x_2$

Any vector $\vec{u}_1 = c_1 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$ for arbitrary non-zero scalar c_1 is eigenvector for $\lambda_1 = 4$.

$\lambda_2 = -3$:

$\begin{bmatrix} 2 & 2 \\ 5 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -3x_1 \\ -3x_2 \end{bmatrix}$

↳ row 1: $2x_1 + 2x_2 = -3x_1$

$5x_1 = -2x_2$

$-\frac{5}{2}x_1 = x_2$

↳ row 2: $5x_1 - x_2 = -3x_2$

$5x_1 = -2x_2$

$-\frac{5}{2}x_1 = x_2$

$\vec{u}_2 = c_2 \begin{bmatrix} 1 \\ -5/2 \end{bmatrix} = c_2 \begin{bmatrix} 2 \\ -5 \end{bmatrix}$

Any vector $\vec{u}_2 = c_2 \begin{bmatrix} 2 \\ -5 \end{bmatrix}$ for arbitrary non-zero scalar c_2 is eigenvector for $\lambda_2 = -3$.

- eigenvals aren't always real
  - ↳ e.g. 2 eigenvals of $\begin{bmatrix} 2 & -1 \\ 1 & 2 \end{bmatrix}$ are $2 \pm i$

page rank & gaussian elimination

date 03/26/2025

PAGE RANK CONVERGENCE

- pg rank process converges to specific eigenvector of Markov matrix M
  - ↳

$$M = \begin{bmatrix} \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{37}{120} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{9}{20} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{1}{40} & \frac{9}{20} & \frac{7}{8} \\ \frac{1}{40} & \frac{1}{6} & \frac{37}{120} & \frac{9}{20} & \frac{1}{40} & \frac{1}{40} \\ \frac{1}{40} & \frac{1}{6} & \frac{1}{40} & \frac{9}{20} & \frac{9}{20} & \frac{1}{40} \end{bmatrix}$$

  - ↳ e.g.

With the earlier example 10 iterations gave:

$$M^{10}p^0 = [0.05205, 0.07428, 0.05782, 0.34797, 0.19975, 0.26810]^T$$

The eigenvector of M corresponding to an eigenvalue of 1 is (approximately):

$$[0.05170, 0.07367, 0.05741, 0.34870, 0.19990, 0.26859]^T$$

- to show pg rank converges, need following props & defns involving Markov matrices:
  1) thm 7.5: every Markov matrix Q has 1 as eigenval
     - eigenvals of Q, Q^T are equal since $\det(Q) = \det(Q^T)$
       → eigenvectors can differ
     - $Q^T e = e$ b/c since cols of Q sum to 1, then rows of Q^T sum to 1
       → e.g.

$$Q^T e = \begin{bmatrix} \frac{1}{4} & \frac{1}{8} & 0 \\ \frac{1}{2} & \frac{7}{8} & 0 \\ \frac{1}{4} & 0 & 1 \end{bmatrix}^T \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{8} & \frac{7}{8} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

     - since $Q^T e = (1)e$, then $\lambda = 1$ is eigenval of Q^T w/ eigenvector e
     - then, 1 is also eigenval of Q
  2) thm 7.6: every eigenval of Q satisfies $|\lambda| \leq 1$ so 1 is largest eigenval
     - proof that $|\lambda| \leq 1$ for Q^T (so also for Q):

Let $\lambda, \vec{x}$ be eigenval/vector pair for Q^T.

$Q^T \vec{x} = \lambda \vec{x}$

Let k be index of largest magnitude entry of $\vec{x}$. This means $\forall j, |x_j| \leq |x_k|$.

$(Q^T \vec{x})_k = \sum_{j=1}^{n} Q_{jk} x_j = \lambda x_k$

$|\lambda x_k| = |\lambda||x_k| = \left|\sum_{j=1}^{n} Q_{jk} x_j\right|$

$\leq \sum_{j=1}^{n} Q_{jk} |x_j|$ ← Δ inequality + Q's entries ≥ 0

$\leq \sum_{j=1}^{n} Q_{jk} |x_k|$ ← $|x_j| \leq |x_k|$

$\leq |x_k|$ ← since cols of Q sum up to 1

$\sum_{j=1}^{n} Q_{jk} x_j$ is dot product of k^{th} col of Q w/ $\vec{x}$. Note that k^{th} col of Q is k^{th} row of Q^T.

- so, $|\lambda||x_k| \leq |x_k|$, $|\lambda| \leq 1$ for Q^T as well as Q since they share same eigenvals

date

3) defn 7.7: Q is +ve Markov matrix if $Q_{ij} \geq 0 \ \forall i,j$

4) thm 7.8: if Q is +ve Markov matrix $\Rightarrow$ there's only 1 linearly independent eigenvector of Q w/ $|\lambda| = 1$

- if Q is +ve Markov, then $Q\vec{x} = (1)\vec{x}$ for some $x \neq 0$
  - → if $Q\vec{y} = \vec{y}$ for some $\vec{y} \neq 0$, then $\vec{y} = c\vec{x}$ for some scalar c (i.e. $\vec{y}$ is multiple of $\vec{x}$)
- i.e. eigenvector w/ $\lambda = 1$ is unique

· proof of pg rank convergence, where if M is +ve Markov matrix, then pg rank converges to unique probability vector $\vec{p}^{\infty}$ for some initial probability vector $\vec{p}^{0}$:

Let $\vec{x}_\ell$ be corresponding eigenvector of λ_ℓ, $\forall \ell$ Assume we can write $\vec{p}^{0}$ as linear combo of eigenvectors $\vec{x}_\ell$.

↳ $\vec{p}^{0} = \sum_{\ell} c_\ell \vec{x}_\ell$ for scalar c_ℓ

Assume eigenvals are in sorted order

↳ $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \dots$

Pg rank computes:

$$(M^k)\vec{p}^{0} = (M^k)\sum_{\ell=1}^{n} c_\ell \vec{x}_\ell$$
$$= \sum_{\ell=1}^{n} (M^k) c_\ell \vec{x}_\ell$$
$$= \sum_{\ell=1}^{n} \lambda_\ell^k c_\ell \vec{x}_\ell \quad \leftarrow \text{since } \vec{x}_\ell \text{ is eigenvector of M so } M\vec{x}_\ell = \lambda\vec{x}_\ell$$
$$= c_1\vec{x}_1 + \sum_{\ell=2}^{n} \lambda_\ell^k c_\ell \vec{x}_\ell$$

Thm 7.8 said that $|\lambda_\ell| < 1$ for $\ell > 1$ since $\lambda_1 = 1$ is unique Hence, $\lim_{k\to\infty} \lambda_\ell^k = 0$ for $\ell > 1$. So, $\vec{p}^{\infty} = \lim_{k\to\infty} (M^k)\vec{p}^{0} = c_1\vec{x}_1$ since other components scale twd 0.

If we start w/ diff probability vector $\vec{q}^{0} = \sum b_\ell \vec{x}_\ell$, then $\vec{q}^{\infty} = b_1\vec{x}_1$. Since $\vec{q}^{\infty}$, $\vec{p}^{\infty}$ are probability vectors, both sum to 1.

↳ $\sum_{i=1}^{n} b_1 x_i = \sum_{i=1}^{n} c_1 x_i = 1$

$b_1(\sum_{i=1}^{n} x_i) = c_1(\sum_{i=1}^{n} x_i)$

$b_1 = c_1$

We see $\vec{p}^{\infty} = \vec{q}^{\infty}$ so pg rank converges to unique vector.

· #iterations required for pg rank to converge to final vector $\vec{p}^{\infty}$ depends on size of 2nd largest eigenval $|\lambda_2|$

↳ $\vec{p}^{k} = (M^k)\vec{p}^{0} = c_1\vec{x}_1 + \sum_{\ell=2}^{n} c_\ell(\lambda_\ell)^k \vec{x}_\ell$

↳ λ_2 dictates slowest rate at which unwanted components of $\vec{p}^{0}$ are shrinking

· for Google matrix, $|\lambda| \simeq \alpha$, where α dictates balance btwn following real links + teleporting randomly

↳ e.g.

e.g., if $\alpha = 0.85$, then $|\lambda_2|^{114} \approx |0.85|^{114} \approx 10^{-8}$. What does this say?

After 114 iterations, any vector components of $\mathbf{p}^0$ not corresponding to the eigenvalue $|\lambda_1|$ will be scaled down by about $\sim 10^{-8}$ (or smaller!)

The resulting vector $\mathbf{p}^{114}$ is likely to be a good approximation of the dominant eigenvector, $\mathbf{x}_1$.

↳ small α implies faster convergence, but $\alpha = 0$ implies only random teleportation + ignores web's link struct completely

- α trades off accuracy for efficiency

date

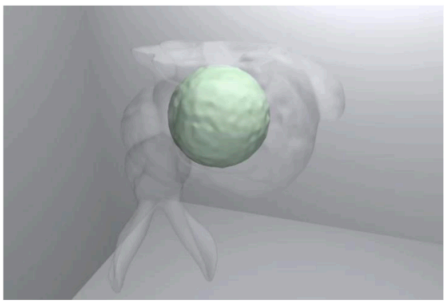
GAUSSIAN ELIMINATION

many practical problems rely on solving systems of linear eqns of form $A\vec{x} = \vec{b}$
- ↳ A is matrix
- ↳ $\vec{b}$ is RHS col vector
- ↳ $\vec{x}$ is col vector of unknowns

· e.g. animating fluids

Computing one frame of animation requires solving a linear system with > **one million unknowns**.

- i.e. matrix A has dimensions $> 1,000,000 \times 1,000,000$.



Must be done once per frame; animations are usually played back at 30 frames / second.

- e.g. for 10 seconds of video, must solve 300 linear systems with size $1,000,000^2$ each.

So: We need methods to solve linear systems **efficiently** and **accurately**.

· Gaussian elimination involves:
- ↳ eliminating vars via row ops until 1 remains
- ↳ back-subbing to recover val of all other vars
- ↳ apply combos of:
  - ∘ multiply row by constant
  - ∘ swap rows
  - ∘ add multiple of 1 row to another

· our process to solve $A\vec{x} = \vec{b}$:

1) factor matrix A into $A = LU$, where L, U are triangular
   - ∘ $L = \begin{bmatrix} & 0 \\ * & \end{bmatrix}$ is lower triangular
   - ∘ $R = \begin{bmatrix} & * \\ 0 & \end{bmatrix}$ is upper triangular
2) solve $L\vec{z} = \vec{b}$ for intermediate vector $\vec{z}$
3) solve $U\vec{x} = \vec{z}$ for $\vec{x}$

· e.g.

Solve $\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 2 \\ 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 4 \\ 2 \end{bmatrix}$ for the vector $\vec{x}$.

$$\left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 1 & -2 & 2 & 4 \\ 1 & 2 & -1 & 2 \end{array}\right] \xrightarrow[R_3 = R_3 - R_1]{R_2 = R_2 - R_1} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & -3 & 1 & 4 \\ 0 & 1 & -2 & 2 \end{array}\right] \xrightarrow{R_3 = R_3 - (-\frac{1}{3}R_2)} \left[\begin{array}{ccc|c} 1 & 1 & 1 & 0 \\ 0 & -3 & 1 & 4 \\ 0 & 0 & -5/3 & 10/3 \end{array}\right]$$

System is now upper triangular. Use back-sub to find $\vec{x}$.

date

$-\frac{5}{3}x_3 = \frac{10}{3}$

$x_3 = -2$

$-3x_2 + x_3 = 4$

$-3x_2 - 2 = 4$

$x_2 = -2$

$x_1 + x_2 + x_3 = 0$

$x_1 - 2 - 2 = 0$

$x_1 = 4$

$\rightarrow \vec{x} = \begin{bmatrix} 4 \\ -2 \\ -2 \end{bmatrix}$

In matrix factorization $A = LU$, L is multiplicative factors + U is upper triangular matrix. To get L, start w/ identity matrix, then record multipliers used in row ops.

↳ $R_2 = R_2 - (1)R_1 \rightarrow L_{21}$

↳ $R_3 = R_3 - (1)R_1 \rightarrow L_{31}$

↳ $R_3 = R_3 - (-\frac{1}{3})R_2 \rightarrow L_{32}$

$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -1/3 & 1 \end{bmatrix}$, $U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 0 & -5/3 \end{bmatrix}$

Verify by multiplication. Diagonal entries of L are always 1s. We need to store below diagonal. To save space, store back into A itself:

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -2 & 2 \\ 1 & 2 & -1 \end{bmatrix} \xrightarrow{R_2 = R_2 - 1R_1} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -3 & 1 \\ 1 & 2 & -1 \end{bmatrix} \xrightarrow{R_3 = R_3 - 1R_1} \begin{bmatrix} 1 & 1 & 1 \\ 1 & -3 & 1 \\ 1 & 1 & -2 \end{bmatrix} \xrightarrow{R_3 = R_3 - (-\frac{1}{3})R_2}$

$\begin{bmatrix} 1 & 1 & 1 \\ 1 & -3 & 1 \\ 1 & -1/3 & -5/3 \end{bmatrix}$

U components

L components

LU FACTORIZATION

date 03/27/2025

- LU factorization/decomp algo:

```
For k = 1, .., n                                  //Iterate over all rows.
    For i = k + 1, ..., n                         //Iterate over each row i beneath row k.
        mult := a_ik / a_kk   ← pivot             //Determine row i's multiplicative factor.
        a_ik := mult                              //Store this factor (instead of a zero).
        For j = k + 1, ..., n                     //Iterate over all (non-zero) columns in the row.
            a_ij := a_ij − mult * a_kj            //Subtract the scaled row data.
        EndFor
    EndFor
EndFor                                            //Note: Resulting factors are stored
                                                  back //into A matrix for sake of space.
```

- given factorization $A=LU$, can quickly solve $A\vec{x}=\vec{b}$ b/c it's same as solving $LU\vec{x}=\vec{b} \rightarrow L(U\vec{x})=\vec{b}$
  - ↳ define $\vec{z}=U\vec{x}$, then do 2 separate solves:
    - first solve $L\vec{z}=\vec{b}$ for $\vec{z}$
      → fwd solve
    - then solve $U\vec{x}=\vec{z}$ for $\vec{x}$
      → bwd solve
  - ↳ L, U are both triangular so they're more efficient to solve
- bwd solve $U\vec{x}=\vec{z}$ is same as back-sub step from std Gaussian elimination
  - ↳ e.g.:

e.g. Our example last time required back-substitution on:

$$\underbrace{\begin{bmatrix}1 & 1 & 1\\ 0 & -3 & 1\\ 0 & 0 & -5/3\end{bmatrix}}_{U}\underbrace{\begin{bmatrix}x_0\\ x_1\\ x_2\end{bmatrix}}_{\vec{x}}=\underbrace{\begin{bmatrix}0\\ 4\\ 10/3\end{bmatrix}}_{\vec{z}}$$

$$-\frac{5}{3}x_2=\frac{10}{3}$$
$$x_2=-2$$
$$-3x_1+x_2=4$$
$$-3x_1-2=4$$
$$x_1=-2$$
$$\vec{x}=\begin{bmatrix}4\\ -2\\ -2\end{bmatrix}$$
$$x_0+x_1+x_2=0$$
$$x_0-2-2=0$$
$$x_0=4$$

- fwd solve $L\vec{z}=\vec{b}$ gives us same RHS as we had after row ops
  - ↳ e.g.

$$\underbrace{\begin{bmatrix}1 & 0 & 0\\ 1 & 1 & 0\\ 1 & -1/3 & 1\end{bmatrix}}_{L}\underbrace{\begin{bmatrix}z_0\\ z_1\\ z_2\end{bmatrix}}_{\vec{z}}=\underbrace{\begin{bmatrix}0\\ 4\\ 2\end{bmatrix}}_{\vec{b}}$$

Solving it gives $\begin{bmatrix}0\\ 4\\ 10/3\end{bmatrix}$, exactly the RHS we had after row operations on the augmented system in basic Gaussian elimination.

- triangular solves algos:

date

$Lz = b$ is called the forward solve (or forward substitution).

e.g., $L = \begin{bmatrix} 1 & 0 & 0 \\ X & 1 & 0 \\ X & X & 1 \end{bmatrix}$

For $i = 1, ..., n$
  $z_i := b_i$
  For $j = 1, ..., i-1$
    $z_i := z_i - l_{ij} * z_j$
  EndFor
EndFor

$Ux = z$ is called the back(ward) solve (or back(ward) substitution).

e.g. $U = \begin{bmatrix} X & X & X \\ 0 & X & X \\ 0 & 0 & X \end{bmatrix}$

For $i = n, ..., 1$
  $x_i := z_i$
  For $j = i+1, ..., n$
    $x_i := x_i - u_{ij} * x_j$
  EndFor
  $x_i := x_i / u_{ii}$
EndFor

↳ e.g. fwd solve

$l_{31} z_1 + l_{32} z_2 + z_3 = b_3$

$z_3 = b_3 - l_{32} z_2 - l_{31} z_1$

· RHS vector $\vec{b}$ isn't needed to factor A into A = LU

↳ factoring work can be reused for diff $\vec{b}$'s + only have to redo cheaper fwd/bwd solves

° e.g.

e.g., Let $A = LU$ with

$$L = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -1/3 & 1 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 0 & -5/3 \end{bmatrix}$$

(i.e. Same factorization from earlier example.)

Solve $Ax = b$ for $\begin{bmatrix} 6 \\ 3 \\ 2 \end{bmatrix}$, by first solving $Lz = b$ and then $Ux = z$.

For $L\vec{z} = \vec{b}$:

$$\begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & -1/3 & 1 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} = \begin{bmatrix} 6 \\ 3 \\ 2 \end{bmatrix}$$

$z_1 = 6$

$z_1 + z_2 = 3$

$6 + z_2 = 3$

$z_2 = -3$

$z_1 - \frac{1}{3} z_2 + z_3 = 2$

$6 - \frac{1}{3}(-3) + z_3 = 2$

$z_3 = -5$

$$\vec{z} = \begin{bmatrix} 6 \\ -3 \\ -5 \end{bmatrix}$$

For $U\vec{x} = \vec{z}$:

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & -3 & 1 \\ 0 & 0 & -5/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -3 \\ -5 \end{bmatrix}$$

$-\frac{5}{3} x_3 = -5$

$x_3 = 3$

$-3x_2 + x_3 = -3$

$-3x_2 + 3 = -3$

$x_2 = 2$

$x_1 + x_2 + x_3 = 6$

$x_1 + 2 + 3 = 6$

$x_1 = 1$

$\therefore \vec{x} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$

e.g. diff RHS application

e.g. Polynomial fitting of $y = p(x)$.

If you change only the y-values being interpolated, you can factorize the Vandermonde matrix once and reuse it.

$$V = \begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-1} \\ & & \cdots & \\ 1 & x_n & \cdots & x_n^{n-1} \end{bmatrix}, \quad \vec{c} = \begin{bmatrix} c_1 \\ \vdots \\ c_n \end{bmatrix} \quad \text{and} \quad \vec{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

- if diagonal entry a_{kk} is 0 or near 0 can lead to numerical instability (i.e. FP error)
  - ↳ near 0 makes multiplicative factor $\frac{a_{ik}}{a_{kk}}$ really large in magnitude
  - ↳

For $k = 1, .., n$
  For $i = k+1, ..., n$
    $mult := a_{ik}/a_{kk}$ ← Divide by (exactly or nearly) zero?
    $a_{ik} := mult$
    For $j = k+1, ..., n$
      $a_{ij} := a_{ij} - mult * a_{kj}$
    EndFor
  EndFor
EndFor

  - ↳ strat: find row w/ largest magnitude in curr col beneath curr row + swap those rows if larger than curr entry
    - e.g.

$$\begin{bmatrix} 0 & 2 & 3 \\ 1 & 4 & 2 \\ -3 & 2 & -1 \end{bmatrix} \xrightarrow{\text{after pivoting}} \begin{bmatrix} -3 & 2 & -1 \\ 1 & 4 & 2 \\ 0 & 2 & 3 \end{bmatrix}$$

- in order for factorization view to account for row swaps, find modified factorization of A st $PA = LU$, where P is permutation matrix
  - ↳ P is permuted (i.e. row-swapped) version of identity matrix I
  - ↳ e.g.

e.g. to swap rows 2 and 3 of a 3×3 matrix, swap rows 2 and 3 of I.

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

Multiplying with a matrix performs the same swap:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{P} \underbrace{\begin{bmatrix} 2 & 3 & 1 \\ 1 & 2 & 2 \\ 5 & 4 & -1 \end{bmatrix}}_{A} = \underbrace{\begin{bmatrix} 2 & 3 & 1 \\ 5 & 4 & -1 \\ 1 & 2 & 2 \end{bmatrix}}_{PA}$$

- given $PA = LU$, multiplying $A\vec{x} = \vec{b}$ by P gives $PA\vec{x} = P\vec{b} = \vec{b}'$
  - ↳ $LU\vec{x} = P\vec{b} = \vec{b}'$
    - solve by first permuting entries of $\vec{b}$ by P to form $\vec{b}'$
    - then, fwd + bwd-sub finds $\vec{x}$
- to find P during factorization, whenever we swap pair of rows during LU factorization, swap corresponding rows of P (including already stored factors)
  - ↳ e.g.

$$\begin{bmatrix} 1 & 4 & 5 \\ -2 & 3 & 3 \\ 3 & 0 & 6 \end{bmatrix} x = \begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix}$$

date

Set $P = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ to start.

$\begin{bmatrix} 1 & 4 & 5 \\ -2 & 3 & 3 \\ 3 & 0 & 6 \end{bmatrix}$ swap R_1, R_3, $P = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ $\begin{bmatrix} 3 & 0 & 6 \\ -2 & 3 & 3 \\ 1 & 4 & 5 \end{bmatrix}$ $R_2 = R_2 - (-\frac{2}{3})R_1$, $R_3 = R_3 - (\frac{1}{3})R_1$ $\begin{bmatrix} 3 & 0 & 6 \\ -2/3 & 3 & 7 \\ 1/3 & 4 & 3 \end{bmatrix}$ swap R_2, R_3, $P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ $\begin{bmatrix} 3 & 0 & 6 \\ 1/3 & 4 & 3 \\ -2/3 & 3 & 7 \end{bmatrix}$

$R_3 = R_3 - (\frac{3}{4})R_2$ $\begin{bmatrix} 3 & 0 & 6 \\ 1/3 & 4 & 3 \\ -2/3 & 3/4 & 19/4 \end{bmatrix}$

$L = \begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -2/3 & 3/4 & 1 \end{bmatrix}$ $U = \begin{bmatrix} 3 & 0 & 6 \\ 0 & 4 & 3 \\ 0 & 0 & 19/4 \end{bmatrix}$ $P = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$

Using $PA = LU$ factorization, solve $A\vec{x} = \vec{b}$ for $\vec{x}$ w/ $\vec{b} = \begin{bmatrix} 4 \\ 1 \\ -3 \end{bmatrix}$.

1) Compute $\vec{b}' = P\vec{b}$.

$$\vec{b}' = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 4 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} -3 \\ 4 \\ 1 \end{bmatrix}$$

2) Solve $L\vec{z} = \vec{b}'$ (fwd solve).

$$\begin{bmatrix} 1 & 0 & 0 \\ 1/3 & 1 & 0 \\ -2/3 & 3/4 & 1 \end{bmatrix} \vec{z} = \begin{bmatrix} -3 \\ 4 \\ 1 \end{bmatrix} \longrightarrow \vec{z} = \begin{bmatrix} -3 \\ 5 \\ -19/4 \end{bmatrix}$$

3) Solve $U\vec{x} = \vec{z}$ (bwd solve).

$$\begin{bmatrix} 3 & 0 & 6 \\ 0 & 4 & 3 \\ 0 & 0 & 19/4 \end{bmatrix} \vec{x} = \begin{bmatrix} -3 \\ 5 \\ -19/4 \end{bmatrix} \longrightarrow \vec{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}$$

- Gaussian elimination w/ row pivoting summary:
  1) perform LU factorization on A to find P, L, U
  2) compute $\vec{b}' = P\vec{b}$
  3) solve $L\vec{z} = \vec{b}'$ for $\vec{z}$ (fwd solve)
  4) solve $U\vec{x} = \vec{z}$ for $\vec{x}$ (bwd solve)

more LU, norms, & conditioning

date 03/29/2025

COST FOR SOLVING LINEAR SYSTEMS

- to know asymptotic cost of solving system of size $n \times n$, apx as # adds + subtracts + multiplies + divides
  - ↳ measuring unit is FLOP (FP ops)

Careful: FLOPS is FP ops per sec

- cost of LU factorization:

For $k = 1, ..., n$
  For $i = k+1, ..., n$
    $mult := a_{ik}/a_{kk}$ — 1 FLOPs (division) outside the innermost loop.
    $a_{ik} := mult$
    For $j = k+1, ..., n$
      $a_{ij} := a_{ij} - mult * a_{kj}$ — 2 FLOPs (1 subtraction, 1 multiply) in the innermost loop.
    EndFor
  EndFor
EndFor

Summing over all the loops we get:

$$\sum_{k=1}^{n} \sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 2\right) = \frac{2n^3}{3} + O(n^2)$$

The above requires using the following sum identities ...

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

and

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

↳ FLOPs count:

$$\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (1 + \sum_{j=k+1}^{n} 2)$$
$$= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (1 + 2(n - (k+1) + 1))$$
$$= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (1 + 2(n-k))$$
$$= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (2n - 2k + 1)$$
$$= \sum_{k=1}^{n-1} (2n-2k+1)(n-k)$$
$$= \sum_{k=1}^{n-1} (2k^2 + k(-4n-1) + 2n^2 + 2)$$
$$= \frac{2n(n+1)(2n+1)}{6} + (-4n-1)\frac{n(n+1)}{2} + (2n^2+n)n$$
$$= \frac{(n^2+n)(2n+1)}{3} + \frac{(-4n^2-n)(n+1)}{2} + 2n^3 + n^2$$
$$= \frac{2n^3}{3} + n^2 + \frac{n}{3} - 2n^3 - \frac{5n^2}{2} - \frac{n}{2} + 2n^3 + n^2$$
$$= \frac{2n^3}{3} - \frac{n^2}{2} - \frac{n}{6} \text{ FLOPs}$$
$$= \frac{2n^3}{3} + O(n^2) \text{ FLOPs}$$

↳ total cost is $\frac{2n^3}{3} + O(n^2)$

- cost of triangular solve:

Show that the total FLOP count of backward substitution ($Lx = z$) is: $n^2 + O(n)$ FLOPs.

For $i = n, ..., 1$
  $x_i := z_i$
  For $j = i+1, ..., n$
    $x_i := x_i - u_{ij} * x_j$ — 2 FLOPs
  EndFor
  $x_i := x_i / u_{ii}$ — 1 FLOP
EndFor

We previously used these identities:

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

and

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

↳ FLOPs count:

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

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

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

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

$= n + 2n^2 - (n^2 + n)$

$= n + 2n^2 - n^2 - n$

$= n^2$ FLOPs

↳ 1 triangular solve (bwd/fwd) costs $n^2 + O(n)$

- cost of solving linear systems:
  - ↳ LU factorization costs apx $\frac{2n^3}{3} + O(n^2)$ FLOPs
  - ↳ each triangular solve costs $n^2 + O(n)$ so total is $2n^2 + O(n)$ FLOPs
  - ↳ factorization cost dominates when n is large + scales worse
  - ↳ given existing factorization, solving for new RHS's is $O(n^2)$, which is cheap

ROW SUBTRACTION VIA MATRICES

- zeroing sub-diagonal entry of col by row subtraction can be written as applying matrix M st $MA^{old} = A^{new}$
  - ↳ A^{old} is og matrix
  - ↳ A^{new} is matrix after subtracting specific row
  - ↳ M is identity matrix w/ 0 entry replaced by -ve of multiplicative factor
  - ↳ e.g.

e.g. The operation ...

$$\underset{(2^{\text{nd}}\text{ row})}{R_2} := \underset{(2^{\text{nd}}\text{ row})}{R_2} - \frac{a_{2,1}}{a_{1,1}}\underset{(1^{\text{st}}\text{ row})}{R_1}$$

can be written as a matrix:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - ↳ e.g.

Example:

$$\underbrace{\begin{bmatrix} 1 & 0 & 0 \\ -3/2 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{M}\underbrace{\begin{bmatrix} 2 & 3 & 4 \\ 3 & 4 & -1 \\ -2 & 2 & 1 \end{bmatrix}}_{A^{old}} = \underbrace{\begin{bmatrix} 2 & 3 & 4 \\ 0 & -1/2 & -7 \\ -2 & 2 & 1 \end{bmatrix}}_{A^{new}}$$

- whole process of factorization viewed as seq of matrix left-multiplications applied to A
  - ↳ matrix in end is U
    - e.g. for 3×3 case, $M^{(3)}M^{(2)}M^{(1)}A = U$
      - → $A = (M^{(3)}M^{(2)}M^{(1)})^{-1}U$
      - $A = \underbrace{(M^{(1)})^{-1}(M^{(2)})^{-1}(M^{(3)})^{-1}}_{L}U$

inverse of $M^{(i)}$ is same matrix w/ off-diagonal entry negated

  - ↳ e.g.

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

date

- in summary, row reduction can be rep as $M^{(3)} M^{(2)} M^{(1)} A = L^{-1} A = U$
  - ↳ interleaving permutation matrices $P^{(k)}$ before each $M^{(k)}$ leads to $PA = LU$ factorization
- alt for solving $A\vec{x} = \vec{b}$ is invert A to get A^{-1}, then multiply $A^{-1}\vec{b}$ to get $\vec{x}$
  - ↳ more expensive + incurs more FP error
  - ↳ most numerical algos avoid computing A^{-1}

NORMS AND CONDITIONING

- **norms** are measurements of size/magnitude for vectors/matrices
  conditioning describes how output of fcn/op/matrix changes due to change in input
- common choices for vector norms for $\vec{x} = [x_1, x_2, \ldots, x_n]^T$:
  - ↳ 1-norm: $\|\vec{x}\|_1 = \sum_{i=1}^{n} |x_i|$
    - aka taxicab/manhatten norm
  - ↳ 2-norm: $\|\vec{x}\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
    - aka Euclidean norm
  - ↳ ∞-norm: $\|\vec{x}\|_\infty = \max_i |x_i|$
    - aka max norm
- above are called **p-norms** for $p = 1, 2, \infty$ + can be written as $\|x\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$
  - ↳ only holds in limit for ∞ case
- **key props** of norms:
  - ↳ if norm is 0, then vector must be 0-vector
    - $\|\vec{x}\| = 0 \rightarrow x_i = 0 \ \forall i$
  - ↳ norm of scaled vector must satisfy $\|\alpha\vec{x}\| = |\alpha| \cdot \|\vec{x}\|$ for any scalar α
  - ↳ triangle inequality holds
    - $\|\vec{x} + \vec{y}\| \leq \|\vec{x}\| + \|\vec{y}\|$
- **matrix norms** are often defined/induced using p-norms of vectors
  - ↳ $\|A\| = \max_{\|\vec{x}\| \neq 0} \frac{\|A\vec{x}\|}{\|\vec{x}\|}$
    - a bit tricky b/c can't try out all possible $\vec{x}$ to determine this
  - ↳ simpler equiv defns:
    - $\|A\|_1 = \max_j \sum_{i=1}^{n} |A_{ij}|$ (max absolute col sum)
    - $\|A\|_\infty = \max_i \sum_{j=1}^{n} |A_{ij}|$ (max absolute row sum)
  - ↳ matrix 2-norm (aka spectral norm) is $\|A\|_2 = \max_{\|\vec{x}\| \neq 0} \frac{\|A\vec{x}\|_2}{\|\vec{x}\|_2}$
    - if λ_i are eigenvals of $A^T A$, then $\|A\|_2 = \max_i \sqrt{|\lambda_i|}$
- **matrix norm props**:
  - ↳ $\|A\| = 0 \leftrightarrow A_{ij} = 0 \ \forall i, j$
  - ↳ $\|\alpha A\| = |\alpha| \cdot \|A\|$ for scalar α
  - ↳ $\|A + B\| \leq \|A\| + \|B\|$
  - ↳ $\|A\vec{x}\| \leq \|A\| + \|\vec{x}\|$

date

↳ $\|AB\| \leq \|A\| \cdot \|B\|$

↳ $\|I\| = 1$

· another matrix norm not induced by std vector norm is Frobenius norm

↳ $\|A\|_F = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij}^2}$

linear algebra conditioning

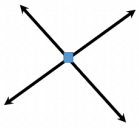
date 04/06/2025

conditioning describes how output of fcn/op/problem changes due to changes in input
↳ independent of algo/numerical method used

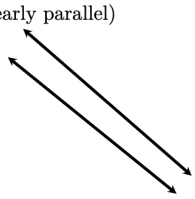
- for linear syster $A\vec{x} = \vec{b}$, we ask:
  ↳ how much does perturbation of $\vec{b}$ cause soln $\vec{x}$ to change?
  ↳ how much does perturbation of A cause soln $\vec{x}$ to change?
- for given perturbation, system is:
  ↳ well-conditioned if $\vec{x}$ changes little
  ↳ ill-conditioned if $\vec{x}$ changes lots
    ° small errors can be radically magnified
- e.g.

Application: Find the intersection between two lines.

Well-conditioned case:
(Nearly perpendicular)



Ill-conditioned case:
(Nearly parallel)



How much does a small angle change modify the intersection point?

- e.g. perturbing $\vec{b}$

Consider the system:

$$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} x = \begin{bmatrix} 4 \\ 7.999 \end{bmatrix} \longrightarrow \vec{x} = \begin{bmatrix} 1.99\ldots911 \\ 1.00\ldots044 \end{bmatrix}$$

And the similar system (perturbed b):

$$\begin{bmatrix} 1 & 2 \\ 2 & 3.999 \end{bmatrix} x = \begin{bmatrix} 4.001 \\ 7.998 \end{bmatrix} \longrightarrow \hat{\vec{x}} = \begin{bmatrix} -3.9990\ldots0178 \\ 4.00\ldots089 \end{bmatrix}$$

How do the solutions differ?

Tiny change in b; huge difference in solution!

- conditioning when perturbing $\vec{b}$:

Given eqn $A\vec{x} = \vec{b}$, perturb $\vec{b}$ by $\Delta\vec{b}$. This gives $A(\vec{x} + \Delta\vec{x}) = \vec{b} + \Delta\vec{b}$. Subtracting off $A\vec{x} = \vec{b}$ gives $A\Delta\vec{x} = \Delta\vec{b} \rightarrow \Delta\vec{x} = A^{-1} \cdot \Delta\vec{b}$.

We want to find rltive change in $\vec{x}$, $\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|}$, given rltive change in b, $\frac{\|\Delta\vec{b}\|}{\|\vec{b}\|}$. Apply norm rules to $A\vec{x} = \vec{b}$:

$\|\vec{b}\| = \|A\vec{x}\|$

$\|\vec{b}\| \leq \|A\| \cdot \|\vec{x}\|$

$\frac{1}{\|\vec{x}\|} \leq \frac{\|A\|}{\|\vec{b}\|}$ (1)

From $\Delta\vec{x} = A^{-1} \cdot \Delta\vec{b}$:

$\|\Delta\vec{x}\| \leq \|A^{-1}\| \cdot \|\Delta\vec{b}\|$ (2)

Multiply (1), (2):

$\|\Delta\vec{x}\| \cdot \frac{1}{\|\vec{x}\|} \leq (\|A^{-1}\| \cdot \|\Delta\vec{b}\|)\left(\frac{\|A\|}{\|\vec{b}\|}\right)$

$\frac{\|\Delta\vec{x}\|}{\|\vec{x}\|} \leq \underbrace{(\|A^{-1}\| \cdot \|A\|)}_{\kappa} \frac{\|\Delta\vec{b}\|}{\|\vec{b}\|}$

date

$\kappa = \|A^{-1}\| \cdot \|A\|$

↳ condition # κ bounds r.l.tive change in $\vec{x}$ due to r.l.tive change in $\vec{b}$

· e.g. perturbing A

Consider another similar system (perturbing A):

$$\begin{bmatrix} 1.001 & 2.001 \\ 2.001 & 3.998 \end{bmatrix} x = \begin{bmatrix} 4.001 \\ 7.998 \end{bmatrix} \longrightarrow \vec{x} = \begin{bmatrix} 3.9940089\ldots \\ 0.00149775\ldots \end{bmatrix}$$

Again, compare the solutions.

Tiny change in A; huge difference in solution!

Notice A is close to the singular matrix, $A = \begin{bmatrix} 1 & 2 \\ 2 & 4 \end{bmatrix}$ $\longrightarrow$ $\det(A) = 0$

We'll now try to characterize when these effects are likely to occur.

· conditioning when perturbing A:

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

$A\vec{x} + A\vec{\Delta x} + \Delta A\vec{x} + \Delta A\vec{\Delta x} = \vec{b}$

Subtract $A\vec{x} = \vec{b}$, then rearrange:

$A\vec{\Delta x} = -(\Delta A)(\vec{x} + \vec{\Delta x})$

$\vec{\Delta x} = -A^{-1}(\Delta A)(\vec{x} + \vec{\Delta x})$

$\|\vec{\Delta x}\| \leq \|A^{-1}\| \cdot \|\Delta A\| \cdot \|\vec{x} + \vec{\Delta x}\|$

Multiply by $\frac{\|A\|}{\|A\|} = 1$:

$\|\vec{\Delta x}\| \leq \|A\| \cdot \|A^{-1}\| \cdot \frac{\|\Delta A\|}{\|A\|} \cdot \|\vec{x} + \vec{\Delta x}\|$

$\frac{\|\vec{\Delta x}\|}{\|\vec{x} + \vec{\Delta x}\|} \leq \underbrace{(\|A\| \cdot \|A^{-1}\|)}_{\kappa} \cdot \frac{\|\Delta A\|}{\|A\|}$

↳ again, condition # κ bounds r.l.tive change in $\vec{x}$

· condition # of matrix A is denoted $\kappa(A) = \|A\| \cdot \|A^{-1}\|$

↳ $\kappa \approx 1 \rightarrow$ A is well-conditioned

↳ $\kappa \gg 1 \rightarrow$ A is ill-conditioned

↳ $\kappa(A)$ provides upper bound on r.l.tive change in $\vec{x}$ due to:

- r.l.tive change in $\vec{b}$
  $\rightarrow \frac{\|\vec{\Delta x}\|}{\|\vec{x}\|} \leq \kappa(A) \frac{\|\vec{\Delta b}\|}{\|\vec{b}\|}$
- r.l.tive change in A
  $\rightarrow \frac{\|\vec{\Delta x}\|}{\|\vec{x} + \vec{\Delta x}\|} \leq \kappa(A) \frac{\|\Delta A\|}{\|A\|}$

↳ diff norms will give diff κ

- specify norm w/ subscript
  → e.g. $\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2$
- if unspecified, always assume 2-norm

· norms differ from 1 another by only constant factor

↳ $C_1\|\vec{x}\|_a \leq \|\vec{x}\|_b \leq C_2\|\vec{x}\|_a$ for constants C_1, C_2 + norms $\|\cdot\|_a, \|\cdot\|_b$

· condition #s help bound accuracy of numerical soln, but we need exact soln $\vec{x}$ for comparison

↳ chicken + egg problem

· as proxy for error, often use residual, which is how much computed soln fails to

satisfy og problem
↳ $\vec{r} = \vec{b} - A(\vec{x}_{apx})$

- relating residual to error:
  - ↳ assume $\vec{x}_{apx} = \vec{x} + \vec{\Delta x}$
  - ↳ $\vec{r} = \vec{b} - A(\vec{x}_{apx})$
    $\vec{r} = \vec{b} - A(\vec{x} + \vec{\Delta x})$
    $A(\vec{x} + \vec{\Delta x}) = \vec{b} - \vec{r}$
  - ↳ $\vec{r}$ looks like perturbation of $\vec{b}$
  - ↳ using $\vec{r} = \vec{\Delta b}$:
    $\frac{\|\vec{\Delta x}\|}{\|\vec{x}\|} \leq \kappa(A) \frac{\|\vec{r}\|}{\|\vec{b}\|}$
  - ↳ soln's rltive error is bounded by κ times rltive size of $\vec{r}$ wrt RHS $\vec{b}$
    - i.e. if we know A's condition #, computed residual indicates how large error might be at worst
    - if $\kappa \approx 1$, small residual indicates small rltive error
    - if $\kappa >> 1$, residual could be small while error's large
- many alt algos for solving $A\vec{x} = \vec{b}$ are iterative ↑ size of residual dictates when to stop
  - ↳ e.g.

```
Do
    Improve estimate x_cur of x;
    Recompute r = b-Ax_cur;
While (norm(r) > tolerance)
```

- in FP, Gaussian elimination w/pivoting on $A\vec{x} = \vec{b}$ is quite stable + accurate
  - ↳ numerical result $\hat{\vec{x}}$ gives exact soln to "nearby" problem (i.e. perturbed A)
    - $(A + \Delta A)\hat{\vec{x}} = \vec{b}$
      → $\|\Delta A\| = \|A\| \epsilon_{machine}$
    - applying bound:
      $\frac{\|\vec{x} - \hat{\vec{x}}\|}{\|\hat{\vec{x}}\|} \leq \kappa(A) \frac{(\|A\| \epsilon_{machine})}{\|A\|}$
      $\frac{\|\vec{x} - \hat{\vec{x}}\|}{\|\hat{\vec{x}}\|} \leq \kappa(A) \epsilon_{machine}$
      → e.g. if $\kappa(A) = 10^{10}$, $\epsilon_{machine} = 10^{-16}$, then rltive error in $\hat{\vec{x}}$ is around 10^{-6} or roughly 6 accurate digits

- e.g.

Find the condition numbers $\kappa_1(A)$ and $\kappa_\infty(A)$ for the matrix

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

using the fact that

$$A^{-1} = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 2/3 & -1/3 \\ 0 & -1/3 & 2/3 \end{bmatrix}.$$

$$\kappa(A) = ||A|| \cdot ||A^{-1}||$$

$$||A||_1 = \max_j \sum_{i=1}^{n} |A_{ij}|$$

$$||A||_\infty = \max_i \sum_{j=1}^{n} |A_{ij}|$$

$\kappa_1(A) = \|A\|_1 \cdot \|A^{-1}\|_1$

$\|A\|_1 = \max_j \sum_{i=1}^{n} |A_{ij}|$ → max absolute col sum

$= \max\{(3+0+0), (0+2+1), (0+1+2)\}$

$= 3$

date

$\|A^{-1}\|_1 = \max\{(\frac{1}{3}+0+0), (0+\frac{2}{3}+\frac{1}{3}), (0+\frac{1}{3}+\frac{2}{3})\}$

$= 1$

$\kappa_1(A) = 3 \cdot 1$

$= 3$

$\kappa_\infty(A) = \|A\|_\infty \cdot \|A^{-1}\|_\infty$

$\|A\|_\infty = \max_i \sum_{j=1}^{n} |A_{ij}|$ ⟶ max absolute row sum

$= \max\{(3+0+0), (0+2+1), (0+1+2)\}$

$= 3$

$\|A^{-1}\|_\infty = \max\{(\frac{1}{3}+0+0), (0+\frac{2}{3}+\frac{1}{3}), (0+\frac{1}{3}+\frac{2}{3})\}$

$= 1$

$\kappa_\infty(A) = 3 \cdot 1$

$= 3$

· e.g.

What is $\kappa_2(A)$, if we know the eigenvalues λ_i of A^TA are $1, 9, 9$?

Reminder: $||A||_2 = \sqrt{\text{max magnitude eigenvalue of} A^TA}$.

$\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2$

$\|A\|_2 = \max_i \sqrt{|\lambda_i|}$

$= \sqrt{|9|}$

$= 3$

For $\|A^{-1}\|_2$, need eigenvals of $A^{-T}A^{-1} = (A^{-1})^T A^{-1} = (AA^T)^{-1}$. Since A is symmetric, $AA^T = A^TA$, so they have same eigenvals. To find eigenvals of $(AA^T)^{-1}$, invert eigenvals of $AA^T = A^TA$, giving $1, \frac{1}{9}, \frac{1}{9}$.

$\|A^{-1}\|_2 = \max_i \sqrt{|\lambda_i|}$

$= 1$

$\kappa_2(A) = 3 \cdot 1$

$= 3$

· e.g. using props of norms:

↳ ❶ Why is it true that $\kappa(\alpha A) = \kappa(A)$?

$\kappa(\alpha A) = \|\alpha A\| \cdot \|(\alpha A)^{-1}\|$

$= \|\alpha A\| \cdot \|\frac{1}{\alpha} A^{-1}\|$

$= |\alpha| \, \|A\| \cdot |\frac{1}{\alpha}| \, \|A^{-1}\|$

$= \|A\| \cdot \|A^{-1}\|$

$= \kappa(A)$

↳ ❷ Why is it true that $\kappa(A) \geq 1$?

$\kappa(A) = \|A\| \cdot \|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1$

This gives $\kappa(A) \geq 1$.

· e.g.

date

Are the following matrices (relatively) ill-conditioned or well-conditioned? How do you know?

1. $\begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix}$
2. $\begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$
3. $\begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

$||A|| = 0 \leftrightarrow A_{ij} = 0 \; \forall i, j.$
$||\alpha A|| = |\alpha| \cdot ||A||$ for scalar α
$||A + B|| \leq ||A|| + ||B||$
$||A\boldsymbol{x}|| \leq ||A|| \cdot ||\boldsymbol{x}||$
$||AB|| \leq ||A|| \cdot ||B||$
$||I|| = 1$

1) $A = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix}$

$A^T A = \begin{bmatrix} 10^{20} & 0 \\ 0 & 10^{20} \end{bmatrix}$

$\lambda_1, \lambda_2 = 10^{20}$

$\|A\|_2 = \max \sqrt{|\lambda_i|}$

$= 10^{10}$

$A^{-1} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

$(A^{-1})^T A^{-1} = \begin{bmatrix} 10^{-20} & 0 \\ 0 & 10^{-20} \end{bmatrix}$

$\lambda_1, \lambda_2 = 10^{-20}$

$\|A^{-1}\|_2 = \max \sqrt{|\lambda_i|}$

$= 10^{-10}$

$\kappa_2(A) = \|A\|_2 \cdot \|A^{-1}\|_2$

$= 10^{10} \cdot 10^{-10}$

$= 1$

↳ well-conditioned

2) $A = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

$A^T A = \begin{bmatrix} 10^{20} & 0 \\ 0 & 10^{-20} \end{bmatrix}$

$\lambda_1 = 10^{20}, \; \lambda_2 = 10^{-20}$

$\|A\|_2 = \sqrt{10^{20}}$

$= 10^{10}$

$A^{-1} = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{10} \end{bmatrix}$

$(A^{-1})^T A^{-1} = \begin{bmatrix} 10^{-20} & 0 \\ 0 & 10^{20} \end{bmatrix}$

$\lambda_1 = 10^{-20}, \; \lambda_2 = 10^{20}$

$\|A^{-1}\|_2 = \sqrt{10^{20}}$

$= 10^{10}$

$\kappa_2(A) = 10^{10} \cdot 10^{10}$

$= 10^{20}$

↳ ill-conditioned

3) $A = \begin{bmatrix} 10^{-10} & 0 \\ 0 & 10^{-10} \end{bmatrix}$

$A^T A = \begin{bmatrix} 10^{-20} & 0 \\ 0 & 10^{-20} \end{bmatrix}$

$\lambda_1, \lambda_2 = 10^{-20}$

date

$\|A\|_2 = 10^{-10}$

$A^{-1} = \begin{bmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{bmatrix}$

$(A^{-1})^T A^{-1} = \begin{bmatrix} 10^{20} & 0 \\ 0 & 10^{20} \end{bmatrix}$

$\lambda_1, \lambda_2 = 10^{20}$

$\|A^{-1}\|_2 = 10^{10}$

$\kappa_2(A) = 10^{-10} \cdot 10^{10}$

$= 1$

↳ well-conditioned