CS 357 – Numerical Methods 1 Review

Numerical methods

Method = Math + Complexity + Accuracy

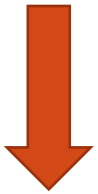
Real life problems

Models

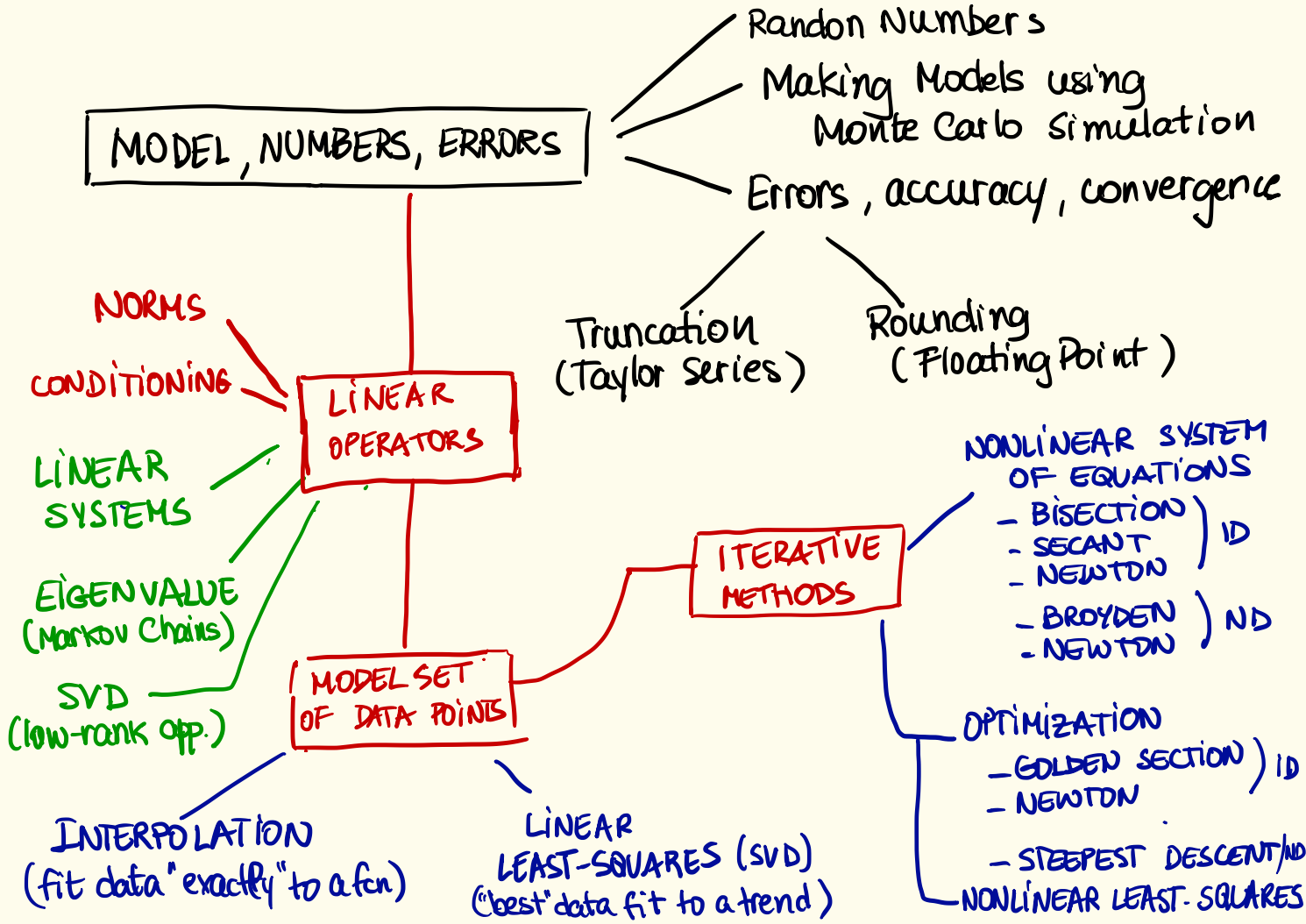
Mathematical formulation

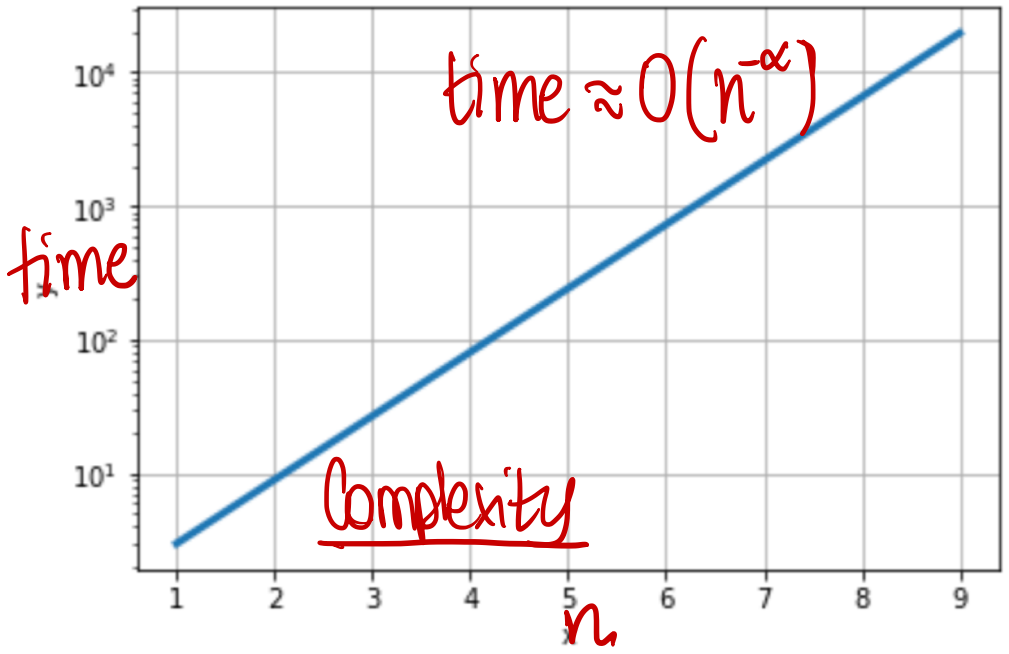
+

Numerical Experiments



Simulation of physical reality


MODEL, NUMBERS, ERRORS
Random Numbers
Making Models using Monte Carlo Simulation
Errors, accuracy, convergence
Truncation (Taylor Series)
Rounding (Floating Point)
NORMS
CONDITIONING
LINEAR OPERATORS
LINEAR SYSTEMS
EIGENVALUE (Markov Chains)
SVD (low-rank opp.)
MODEL SET OF DATA POINTS
INTERPOLATION (fit data "exactly" to a fcn)
LINEAR LEAST-SQUARES (SVD) ("best" data fit to a trend)
ITERATIVE METHODS
NONLINEAR SYSTEM OF EQUATIONS
- BISECTION
- SECANT
- NEWTON
1D
- BROYDEN
- NEWTON
ND
OPTIMIZATION
- GOLDEN SECTION
- NEWTON
1D
- STEEPEST DESCENT/ND
NONLINEAR LEAST-SQUARES

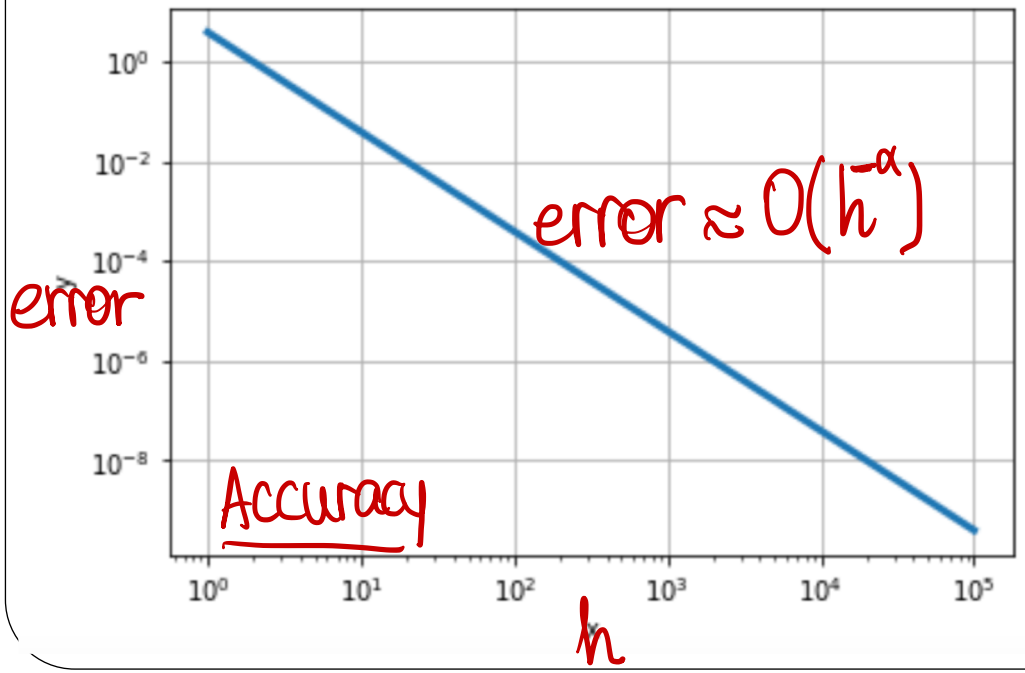


→ Exponential function

$y = e^{\alpha x}$

$\log y = \log(e^{\alpha x})$

$\bar{y} = \alpha x$



→ Power function

$y = x^{\alpha}$

$\log y = \log(x^{\alpha}) = \alpha \log(x)$

$\bar{y} = \alpha \bar{x}$

Let's talk about plots…

- Power functions: $y = a\,x^b$

  $\log y = \log\left(a\,x^b\right) = \log(a) + \log\left(\,x^b\right) = \log(a) + \mathrm{b}\,\log(x)$

  $\bar{y} = \bar{a} + b\,\bar{x}$

- Exponential functions: $y = a\,b^x$

  $\log y = \log(a\,b^x) = \log(a) + \log(\,b^x) = \log(a) + \;\mathrm{x}\,\log(b)$

  $\bar{y} = \bar{a} + \bar{b}\,x$

- Log functions: $y = a\log(b\,x)$

  $y = a\log(b) + a\log(x)$

  $y = \bar{b} + a\,\bar{x}$

Random numbers

- Pseudo-random numbers
  - Numbers and sequences appear random, but they are in fact reproducible
  - Good for algorithm development and debugging
- Desired properties for a good random number generator
  - Random pattern
  - Long period
  - Efficiency
  - Repeatability
  - Portability

Monte Carlo Methods

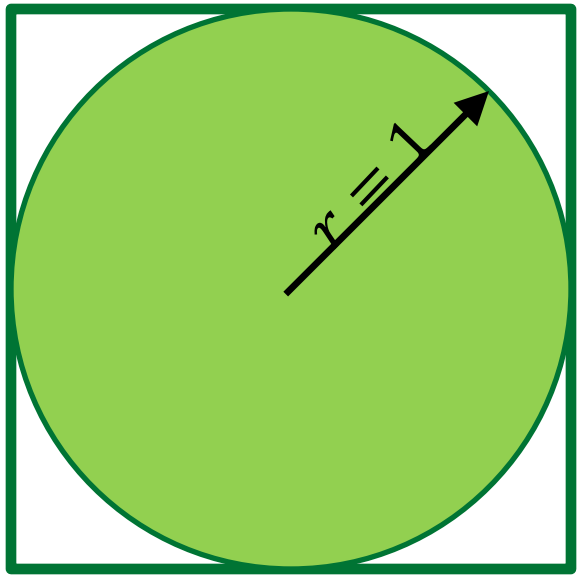
- Algorithms that compute approximations of desired quantities based on repeated randomized sampling

- Typically used to model:
  - Nondeterministic problems
  - Complicated deterministic problems, specially in high dimensions

- Asymptotic behavior is $O\left(\frac{1}{\sqrt{n}}\right)$ when $n \to \infty$, where n is the number of samples

$O(n^{-0.5})$

Using Monte Carlo to approximate integrals

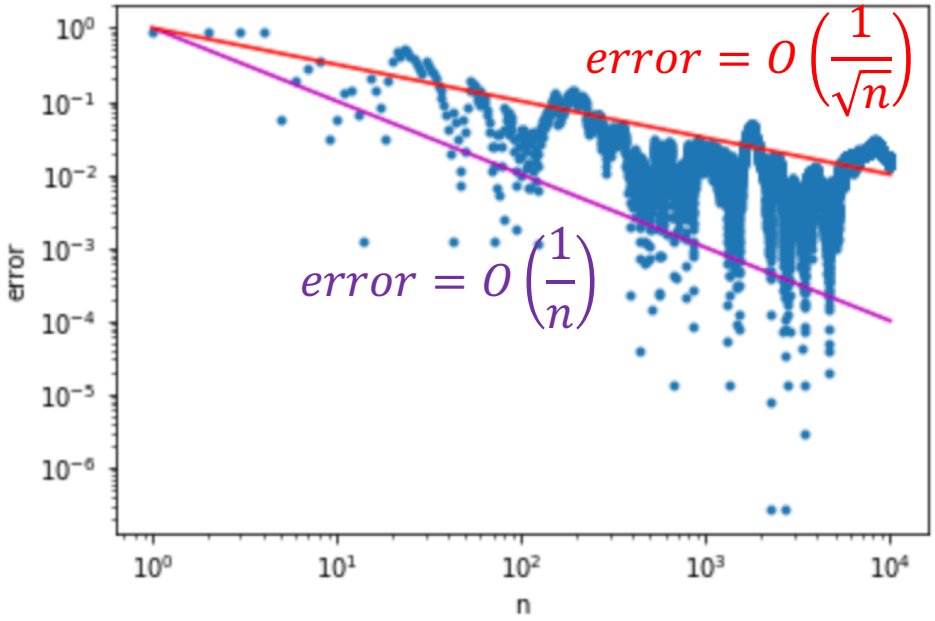


We sample points uniformly inside the domain $D = [x_o, x_1] \times [y_o, y_1]$

Nc: number of points inside circle

Ns: number of points inside square = total number of sample points = n

$$Ac = As\left(\frac{Nc}{n}\right)$$



- CONS: Slow convergence rate
- PROS: Efficiency does not degrade when increasing the dimension of the problem

Error in Numerical Methods

Absolute error: $|x - \bar{x}|$

Relative error: $\dfrac{|x - \bar{x}|}{|x|}$

Accurate to *n* significant digits means that you can trust a total of *n* digits. *Accurate digits* is a measure of relative error.

Relative error: $error = \dfrac{|x_{exact} - x_{approx}|}{|x_{exact}|} \leq 10^{-n+1}$

n is the number of accurate significant digits

3 sig. figures $\longrightarrow$ error $\leq 10^{-3+1}$ $\longrightarrow$ error $\leq 10^{-2}$ (1%)

Rates of convergence or growth

- $error = O(n^{-\alpha})$ or $complexity = O(n^{\alpha})$

  Power function – straight line in a log-log plot

  Algebraic convergence/growth

- $error = O(e^{-\alpha n})$ or $complexity = O(e^{\alpha n})$

  Exponential function – straight line in a linear-log plot

  Exponential convergence/growth

Taylor Series

The Taylor Series approximation about point x_o is given by:

$$f(x) = f(x_o) + f'(x_o)(x - x_o) + \frac{f''(x_o)}{2!}(x - x_o)^2 + \frac{f'''(0)}{3!}(x - x_o)^3 + \cdots$$

$$f(x) = \sum_{i=0}^{\infty} \frac{f^{(i)}(x_o)}{i!}(x - x_o)^i$$

The Taylor Series approximation of degree n, $\hat{f}(x)$, is given by:

$$\hat{f}(x) = \sum_{i=0}^{n} \frac{f^{(i)}(x_o)}{i!}(x - x_o)^i$$

truncated function

$$h = (x - x_o)$$

And the Taylor error of degree n due the truncation is: $O(h^{n+1})$

$$R = f(x) - \hat{f}(x) = \sum_{i=n+1}^{\infty} \frac{f^{(i)}(x_o)\, h^i}{i!} \leq \frac{f^{(n+1)}(\xi)}{(n+1)!} h^{(n+1)}$$

Taylor series for e^x about $x_0 = 0$ is

$$e^x \approx 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

1) Approximate e^x at $x = 0.2$ using Taylor expansion of degree 2

$$1 + \frac{0.2}{1!} + \frac{(0.2)^2}{2!} = 1.22$$

2) Approximate $\frac{d}{dx}(e^x)$ at $x = 0.1$ using Taylor expansion of degree 3

$$1 + \frac{2x}{2!} + \frac{3x^2}{3!} = 1 + 0.1 + \frac{(0.1)^2}{2} = 1.105$$

3) If we want to use the first 4 terms of the expansion, what is the order of the error of the Taylor approximation?

$h = x - x_0$ error $\approx O(h^{n+1})$

(A) 2 (B) 3 (C) 4 (D) 5 (E) 6

Normalized Floating-point numbers

Normalized floating point numbers are expressed as

$$x = \pm q \times 2^m = \pm 1.b_1 b_2 b_3 \ldots b_n \times 2^m = \pm 1.f \times 2^m$$

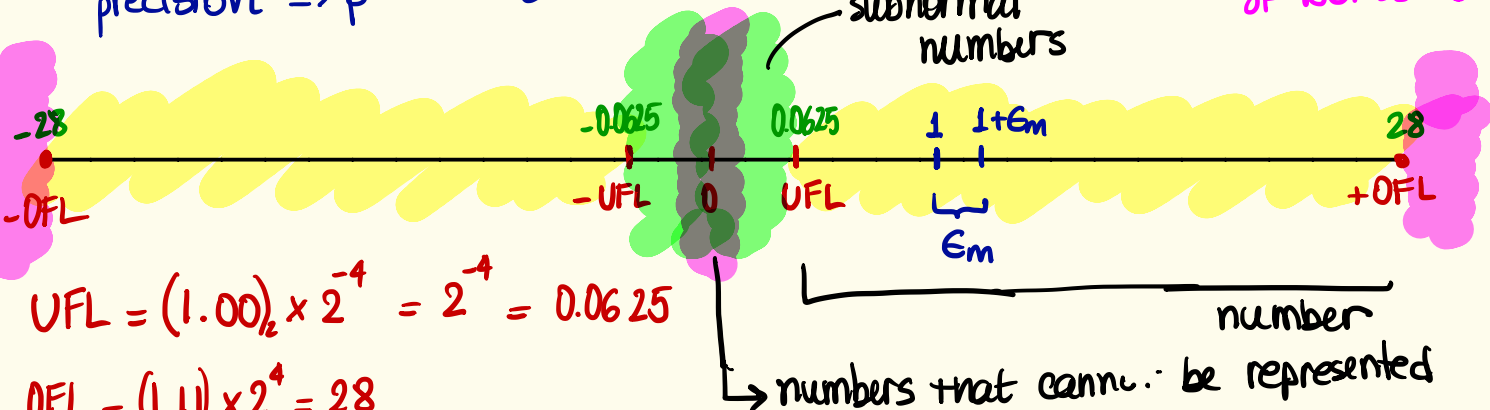
where f is the fractional part of the significand, m is the exponent and $b_i \in \{0,1\}$.

- Exponent range: $[L, U]$
- Precision: $p = n + 1$
- Smallest positive normalized FP number: UFL = 2^L
- Largest positive normalized FP number: OFL = $2^{U+1}(1 - 2^{-p})$
- Machine epsilon (ϵ_m): is defined as the distance (gap) between 1 and the next largest floating point number.
- Noticeable gap around zero, present in any floating system, due to normalization
  - ✓ The smallest possible significand is $1.00 \ldots 0$
  - ✓ The smallest possible exponent is L
- Relax the requirement of normalization, and allow the leading digit to be zero, only when the exponent is at its minimum ($m = L$)
- Computations with subnormal numbers are often slow.

$x = \pm 1.b_1 b_2 \times 2^m \qquad m \in [-4, 4] \qquad b_i \in \{0, 1\}$

ONLY FINITE REPRESENTATION OF NUMBERS!

precision $\Rightarrow p = n+1 = 3$



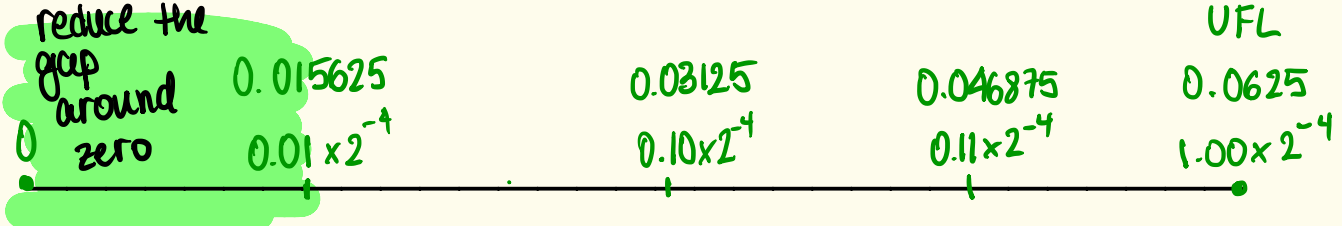
$UFL = (1.00)_2 \times 2^{-4} = 2^{-4} = 0.0625$

$OFL = (1.11)_2 \times 2^4 = 28$

Machine epsilon: gap between 1 and next largest fp number

$\epsilon_m = 1.00 \times 2^0 - 1.01 \times 2^0 = 0.01 \times 2^0 = 0.25$

Subnormal numbers → let the leading digit $\pm 0.b_1 b_2$ be zero if $m = L$



What is machine epsilon double precision ?

$\pm 1.b_1 b_2 b_3 \cdots b_{52} \times 2^{m}$

1 bit

52 bits in the fraction

11 bits

$1.000\cdots0 \times 2^0 = 1$

$1.000\cdots1 \times 2^0$ = next fp number after 1

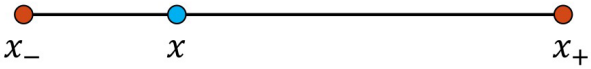
$\epsilon_m = 0.000\cdots1 \times 2^0 \Longrightarrow \boxed{\epsilon_m = 2^{-52}}$

2^{-52}

Rounding

The process of replacing x by a nearby machine number $\tilde{x}$ is called rounding, and the error involved is called **roundoff error.**

$$x = \pm 1.b_1 b_2 b_3 \ldots b_n \ldots \times 2^m$$



round to nearest

$fl(x)$ round down

"exact" number

$fl(x)$ round up

Relative error due to rounding:

$$e_r = \frac{|fl(x) - x|}{|x|} \leq \epsilon_m$$

- single precision: $e_r \leq 2^{-23} \approx 10^{-7}$ (7 decimal accurate digits)
- double precision: $e_r \leq 2^{-52} \approx 10^{-16}$ (16 decimal accurate digits)

Absolute error:

$$e_a = |fl(x) - x| \leq \epsilon_m |x|$$

Loss of Significance

Results from floating point arithmetic used by computers, and the numbers of significant digits is substantially reduced.

Consider the decimal representation for π using 20 significant figures

```
a = 3.1415926535897932385;
```

Consider the decimal representation for π using 10 significant figures

```
b = 3.141592654;
```

Now we perform the following calculations:

```
a - 3.1415926530000000000 (* calculation gives 9 significant digits *)
```

$5.89793239 \times 10^{-10}$

```
b - 3.141592653 (* calculation gives 1 significant digit *)
```

$1. \times 10^{-9}$

$$f(x) = \sqrt{x^2 + 1} - 1 = \frac{x^2}{\sqrt{x^2 + 1} - 1}$$

Using five-decimal-digit arithmetic:

$$f(10^{-3}) = \sqrt{(10^{-3})^2 + 1} - 1 = 0$$

$$f(10^{-3}) = \frac{(10^{-3})^2}{\sqrt{(10^{-3})^2 + 1} - 1} = \frac{10^{-6}}{2}$$

Norms

What's a norm?

- A generalization of 'absolute value' to vectors.
- $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}_0^+$, returns a 'magnitude' of the input vector
- In symbols: Often written $\|\boldsymbol{x}\|$.

Define norm.

A function $\|\boldsymbol{x}\| : \mathbb{R}^n \to \mathbb{R}_0^+$ is called a norm if and only if

1. $\|\boldsymbol{x}\| > 0 \Leftrightarrow \boldsymbol{x} \neq \boldsymbol{0}$.
2. $\|\gamma \boldsymbol{x}\| = |\gamma| \, \|\boldsymbol{x}\|$ for all scalars γ.
3. Obeys triangle inequality $\|\boldsymbol{x} + \boldsymbol{y}\| \leqslant \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$

Properties of Matrix Norms

Matrix norms inherit the vector norm properties:

1. $\|A\| > 0 \Leftrightarrow A \neq \mathbf{0}$.
2. $\|\gamma A\| = |\gamma| \, \|A\|$ for all scalars γ.
3. Obeys triangle inequality $\|A + B\| \leqslant \|A\| + \|B\|$

But also some more properties that stem from our definition:

1. $\|A\boldsymbol{x}\| \leqslant \|A\| \, \|\boldsymbol{x}\|$
2. $\|AB\| \leqslant \|A\| \, \|B\|$ (easy consequence)

Both of these are called submultiplicativity of the matrix norm.

Demo “Matrix Norms”

Vectors $\underset{\sim}{v}$: $(n \times 1)$

$$\|\underset{\sim}{v}\|_p = \left(|v_1|^p + |v_2|^p + \cdots + |v_n|^p \right)^{1/p} \qquad p \geq 1$$

$$\|\underset{\sim}{v}\|_1 = |v_1| + |v_2| + \cdots + |v_n|$$

$$\|\underset{\sim}{v}\|_2 = \sqrt{v_1^2 + v_2^2 \cdots + v_n^2}$$

$$\|\underset{\sim}{v}\|_\infty = \left(|v_1|^\infty + |v_2|^\infty + \cdots + |v_n|^\infty \right)^{1/\infty} = \max_i |v_i|$$

Note that we take the absolute value of the components!

Matrices $\underset{\approx}{A}$:

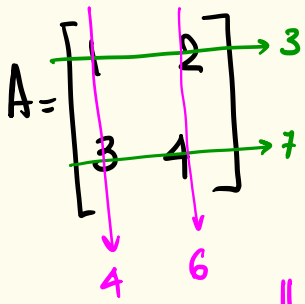
$$\|A\| = \max_{\|x\|=1} \|\underset{\approx}{A}\underset{\sim}{x}\| \quad \text{or} \quad \|A\| = \max_{x} \frac{\|Ax\|}{\|x\|}$$

$\|A\|_1$ = max absolute column sum

$\|A\|_\infty$ = max absolute row sum

$\|A\|_2$ = max singular value

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$



$\|A\|_1 = 6$ $\quad \|A\|_\infty = 7$

$$B = \begin{bmatrix} -1 & 2 \\ 3 & -4 \end{bmatrix}$$

$\|A\|_1 = \|B\|_1$

$\|A\|_\infty = \|B\|_\infty$

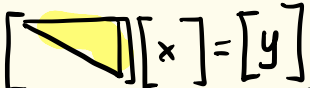
Linear System of Equations

- Solve $\underline{\underline{A}}\,\underline{x} = \underline{b}$
- Factorize $\underline{\underline{A}}$: LU factorization $\Rightarrow \underline{\underline{A}} = \underline{\underline{L}}\,\underline{\underline{U}}$ $(O(n^3))$

$LUx = b \Rightarrow \underbrace{Ly}_{y} = b$

Forward substitution
$O(n^2)$

$Ux = y$



backward substitution
$O(n^2)$

- Solve $\underline{\underline{A}}\,\underline{x}_i = \underline{b}_i$ for different RHS $(i = 1, \dots, k)$ $\Rightarrow$ cost $\cong O(n^3 + 2kn^2)$ and <u>NOT</u> $O(kn^3)$

- Partial Pivoting : avoids division by zero

  Find largest entry in the column (absolute value) and swap it with top row

Partial Pivoting

2 points

Consider LU factorization on the following matrix:

$$A = \begin{bmatrix} 5 & 3 & 2 & 9 \\ -10 & 2 & 1 & 5 \\ 0 & 3 & 3 & 3 \\ 1 & 2 & 1 & 2 \end{bmatrix}$$

If partial pivoting is employed during elimination of the first column, what matrix value should be chosen as the pivot?

Answer*

Let's assume that when solving the system of equations $\boldsymbol{K}\,\boldsymbol{U} = \boldsymbol{F}$, we observe the following:

- When $\boldsymbol{K}$ has dimensions (100,100), computing the LU factorization takes about 1 second and each solve (forward + backward substitution) takes about 0.01 seconds.

Estimate the total time it will take to find the solution $\boldsymbol{U}$ corresponding to 10 different right-hand sides $\boldsymbol{F}$ when the matrix has dimensions (1000,1000)?

LU factorization : $(100)^3$ — 1 sec
$(1000)^3$ — x
$\rightarrow x = \left(\frac{1000}{100}\right)^3 = 10^3$ seconds

One Solve : $(100)^2$ — 0.01 sec
$(1000)^2$ — y
$\rightarrow y = \left(\frac{1000}{100}\right)^2 0.01 = 10^2(0.01) = 1$ sec

total time = 10^3 seconds + $10(1\text{sec}) \approx 10^3$ seconds

Condition number

Small condition numbers mean not a lot of error amplification. Small condition numbers are good!

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \|\boldsymbol{A}^{-1}\| \, \|\boldsymbol{A}\| \, \frac{\|\Delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|} = cond(\boldsymbol{A}) \, \frac{\|\Delta \boldsymbol{b}\|}{\|\boldsymbol{b}\|} \qquad \frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq \|\boldsymbol{A}^{-1}\| \, \|\boldsymbol{A}\| \, \frac{\|\boldsymbol{E}\|}{\|\boldsymbol{A}\|}$$

When solving linear system of equations, the residual is $\boldsymbol{r} = \boldsymbol{b} - \boldsymbol{A}\,\hat{\boldsymbol{x}}$, where $\hat{\boldsymbol{x}} = (\boldsymbol{x} + \Delta \boldsymbol{x})$ is the solution of the perturbed problem. For well-conditioned matrices, small relative residual implies small relative error.

$$\frac{\|\Delta \boldsymbol{x}\|}{\|\boldsymbol{x}\|} \leq cond(\boldsymbol{A}) \, \frac{\|\boldsymbol{r}\|}{\|\boldsymbol{A}\|\|\boldsymbol{x}\|}$$

Gaussian elimination with partial pivoting (where c is small) yields **small relative residual regardless of conditioning of the system**

$$\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{A}\|\|\hat{\boldsymbol{x}}\|} \leq c \, \epsilon_m$$

Rule of thumb: If the entries in $\boldsymbol{A}$ and $\boldsymbol{b}$ are accurate to S decimal digits, then a condition number $\mathbf{10}^{\boldsymbol{W}}$ reduces W digits in the accuracy of the computed solution (i.e., solution will have $(\mathrm{S} - W)$ accurate digits.

When solving a system of linear equations via LU with partial pivoting, which of the following is guaranteed to be small?

A) Relative residual: $\frac{\|\boldsymbol{r}\|}{\|\boldsymbol{A}\|\|\boldsymbol{x}\|}$

B) Relative error: $\frac{\|\Delta \boldsymbol{x}\ \|}{\|\boldsymbol{x}\|}$

C) Neither one of them

D) Both of them

Changing the Right-Hand Side (RHS)

1 point

You performed an experiment using an expensive piece of scientific equipment, then used the resulting $\mathbf{b}$ to solve the linear system $A\mathbf{x} = \mathbf{b}$. Later, you realized that due to a calibration mistake, there may be some error associated with your calculated $\mathbf{b}$.

Your lab is out of funding and can't afford to run another experiment. Worried about the accuracy of your computed result (say $\hat{\mathbf{x}}$), you calculated:

$$\frac{\|\Delta\mathbf{b}\|}{\|\mathbf{b}\|} = 10^{-3} \qquad \text{where} \qquad \Delta\mathbf{b} = \hat{\mathbf{b}} - \mathbf{b}.$$

The condition number of your matrix (A), i.e. $\text{cond}(A)$, is 100.

Calculate an upper bound on the relative error e in the true solution given by:

$$e = \frac{\|\Delta\mathbf{x}\|}{\|\mathbf{x}\|} \qquad \text{where} \qquad \Delta\mathbf{x} = \hat{\mathbf{x}} - \mathbf{x}$$

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta b\|}{\|b\|} = 100\,(10^{-3}) = 0.1$$

$$\boxed{\frac{\|\Delta x\|}{\|x\|} \leq 0.1}$$

Iclicker question

Matrix Conditioning: Accurate digits

1 point

Let's say we want to solve the following linear system:

$$Ax = b$$

Assuming you are working with IEEE double precision floating point numbers, how many digits of accuracy will your answer have if $\kappa(A) = 1000$?

A) 3
B) 10
C) 13
D) 16
E) 32

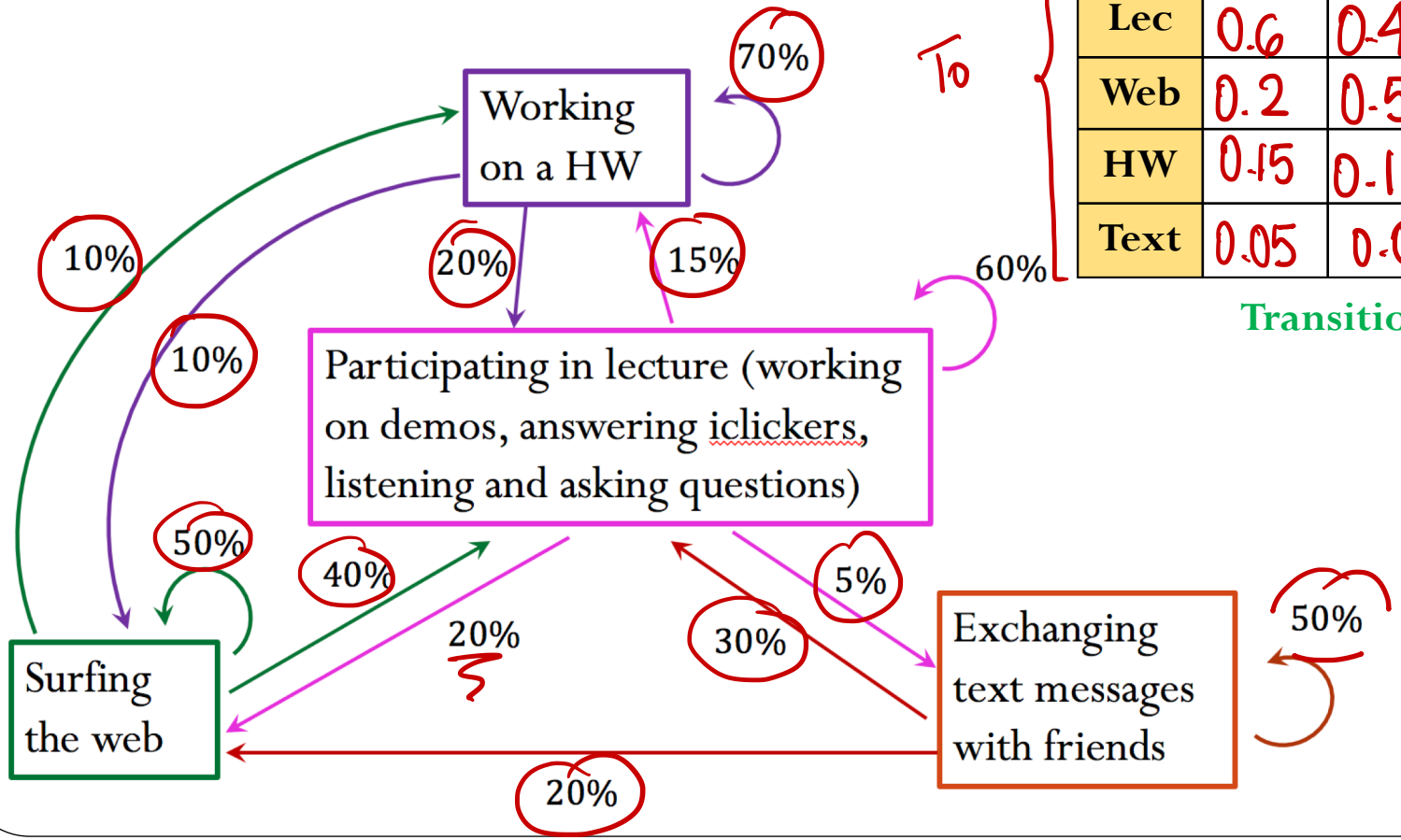
IEEE double precision → input has 16 accurate digits

accuracy is decreased by w, where 10^w is the condition number

output x → (16 - w) accurate digits = 13

Markov chain: only the most recent state matters to determine the probability of the next state. $\boldsymbol{x}_n = \boldsymbol{A}\,\boldsymbol{x}_{n-1}$

Transition (Markov or stochastic) matrix: used to describe the transitions of a Markov chain. Each of its entries is a non-negative real number representing a probability.



From

To

	Lect	Web	HW	Text
Lec	0.6	0.4	0.2	0.3
Web	0.2	0.5	0.1	0.2
HW	0.15	0.1	0.7	0.0
Text	0.05	0.0	0.0	0.5

Transition matrix

Sparse Storage

$$A = \begin{bmatrix} 0 & 0 & 1.3 \\ -1.5 & 0.2 & 0 \\ 5 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0.3 & 3 \end{bmatrix}$$

(columns: 0 1 2; rows: 0 1 2 3 4)

COO Format:

row = [0 1 1 2 4 4] nnz integers

col = [2 0 1 0 1 2] nnz integers

data = [1.3 -1.5 0.2 5 0.3 3] nnz doubles

CSR Format:

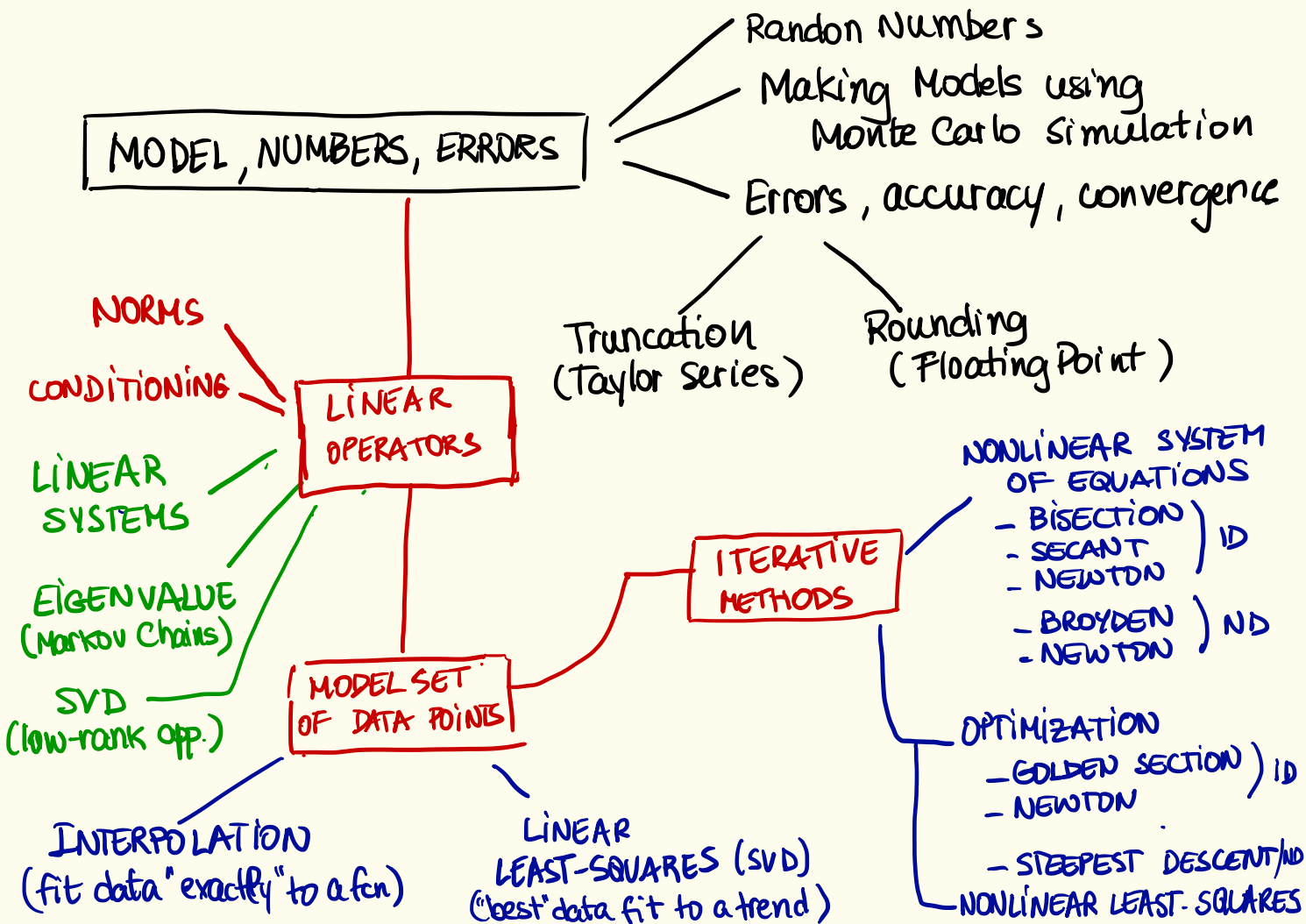
rowptr = [0 1 3 4 4 6] (nrow+1) integers ("zero" row; nnz)

col = [2 0 1 0 1 2] (nnz) integers

data = [1.3 -1.5 0.2 5 0.3 3] (nnz) doubles

indices: 0 1 2 3 4 4 5

row 0 | row 1 | row 2 | row 3 | row 4


MODEL, NUMBERS, ERRORS
Random Numbers
Making Models using Monte Carlo Simulation
Errors, accuracy, convergence
Truncation (Taylor Series)
Rounding (Floating Point)
NORMS
CONDITIONING
LINEAR OPERATORS
LINEAR SYSTEMS
EIGENVALUE (Markov Chains)
SVD (low-rank opp.)
MODEL SET OF DATA POINTS
ITERATIVE METHODS
NONLINEAR SYSTEM OF EQUATIONS
- BISECTION
- SECANT
- NEWTON
1D
- BROYDEN
- NEWTON
ND
OPTIMIZATION
- GOLDEN SECTION
- NEWTON
1D
- STEEPEST DESCENT/ND
NONLINEAR LEAST-SQUARES
INTERPOLATION (fit data "exactly" to a fcn)
LINEAR LEAST-SQUARES (SVD) ("best" data fit to a trend)

Eigenvalue Problems

- $\underline{\underline{A}}$ is $n \times n$ matrix, $\underset{\sim}{x} \neq 0$ is eigenvector of $\underline{\underline{A}}$, λ is eigenvalue of $\underline{\underline{A}}$

$$\underline{\underline{A}}\,\underset{\sim}{x} = \lambda \underset{\sim}{x}$$

- $\underline{\underline{A}}$ is diagonalizable if it has (n) linearly independent eigenvectors.

$$\underline{\underline{A}} = \underline{\underline{U}}\,\underline{\underline{D}}\,\underline{\underline{U}}^{-1} \qquad \underline{\underline{U}} = \begin{bmatrix} | & | & & | \\ \underset{\sim}{u}_1 & \underset{\sim}{u}_2 & \cdots & \underset{\sim}{u}_n \\ | & | & & | \end{bmatrix} \qquad \underline{\underline{D}} = \begin{bmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n \end{bmatrix}$$

- **Power iteration:** → finds the largest eigenvalue (λ_1)

$$\underset{\sim}{x}_{k+1} = \underline{\underline{A}}\,\underset{\sim}{x}_k$$

$$\underset{\sim}{x}_k = \lambda_1^k \left[\alpha_1 \underset{\sim}{u}_1 + \underbrace{\alpha_2 \left(\frac{\lambda_2}{\lambda_1}\right)^k \underset{\sim}{u}_2}_{e_k} + \cdots + \alpha_n \left(\frac{\lambda_n}{\lambda_1}\right)^k \underset{\sim}{u}_n \right]$$

$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$

usually normalize

$$\underset{\sim}{x}_k = \frac{\underline{\underline{A}}\,\underset{\sim}{x}_{k-1}}{\| \underline{\underline{A}}\,\underset{\sim}{x}_{k-1} \|}$$

$$\lim_{k \to \infty} \frac{\underset{\sim}{x}_k}{\lambda_1^k} = \alpha_1 \underset{\sim}{u}_1$$ → $\underset{\sim}{x}_k$ converges to a multiple of the first eigenvector $\underset{\sim}{u}_1$!

Once algorithm converges to eigenvector $\underset{\sim}{x}$ then the eigenvalue is evaluated using **Rayleigh coefficient**:

$$\lambda = \frac{\underset{\sim}{x}^T \underline{\underline{A}}\,\underset{\sim}{x}}{\underset{\sim}{x}^T \underset{\sim}{x}}$$

- Convergence:

$$\frac{e_{k+1}}{e_k} \approx \frac{|\lambda_2|}{|\lambda_1|}$$ → linear convergence

- Cost per iteration: $O(n^2)$ → matrix-vect multiply

Note that ...
Convergence is faster when
$\frac{\lambda_2}{\lambda_1}$ is small or
$\frac{\lambda_1}{\lambda_2}$ is large

- **Inverse Power Iteration** ⟶ finds the smallest eigenvalue (λ_n)

$\underset{\sim}{x}_{k+1} = \underset{\approx}{A}^{-1} \underset{\sim}{x}_k$ ⟶ x_0 = initial guess

solve $\underset{\approx}{A}\, \underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k$ for $\underset{\sim}{x}_{k+1}$

Factorize $PA = LU$ ⟶ $\underset{\approx}{L}\,\underset{\approx}{U}\,\underset{\sim}{x}_{k+1} = \underset{\approx}{P}\,\underset{\sim}{x}_k$

(done only once!)

solve $\underset{\approx}{L}\,\underset{\sim}{y} = \underset{\approx}{P}\,\underset{\sim}{x}_k$

solve $\underset{\approx}{U}\,\underset{\sim}{x}_{k+1} = \underset{\sim}{y}$

} • Cost per iteration $O(n^2)$

- Convergence:

$$\frac{e_{k+1}}{e_k} \approx \frac{|\lambda_n|}{|\lambda_{n-1}|}$$

⟶ linear convergence

convergence is faster when this ratio is small

- **Shifted-Inverse Power Iteration** → finds eigenvalue close to σ

$\underset{\sim}{x}_{k+1} = (\underline{\underline{A}} - \sigma \underline{\underline{I}})^{-1} \underset{\sim}{x}_k$ → x_0 = initial guess

solve $(\underline{\underline{A}} - \sigma \underline{\underline{I}}) \underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k$ $\quad \underline{\underline{B}} = \underline{\underline{A}} - \sigma \underline{\underline{I}}$

Factorize $PB = LU$ (done only once!) → $\underline{\underline{L}}\,\underline{\underline{U}}\,\underset{\sim}{x}_{k+1} = \underline{\underline{P}}\,\underset{\sim}{x}_k$

solve $\underline{\underline{L}}\,y = \underline{\underline{P}}\,\underset{\sim}{x}_k$

solve $\underline{\underline{U}}\,\underset{\sim}{x}_{k+1} = y$

} • Cost per iteration $O(n^2)$

λ_{c1} → eigenvalue closest to σ

λ_{c2} → next eigenvalue closest to σ

Convergence: $\frac{e_{k+1}}{e_k} \approx \frac{|\lambda_{c1} - \sigma|}{|\lambda_{c2} - \sigma|}$ → linear convergence

Convergence is faster when this rate is small

- Rayleigh Quotient Iteration → finds eigenvalue close to σ

$\underset{\sim}{x}_{k+1} = (\underline{\underline{A}} - \sigma \underline{\underline{I}})^{-1} \underset{\sim}{x}_k$ ⟶ x_0 = initial guess

$\underline{\underline{B}}_k = \underline{\underline{A}} - \sigma_k \underline{\underline{I}}$

Factorize $PB_k = LU$ } $O(n^3)$

solve $\underline{\underline{L}}\, y = \underline{\underline{P}}\, \underset{\sim}{x}_k$ } $O(n^2)$

solve $\underline{\underline{U}}\, \underset{\sim}{x}_{k+1} = y$

update $\sigma_k = \dfrac{\underset{\sim}{x}_{k+1}^T \underline{\underline{A}}\, \underset{\sim}{x}_{k+1}}{\underset{\sim}{x}_{k+1}^T \underset{\sim}{x}_{k+1}}$

cost per iteration: $O(n^3)$

At least quadratic convergence

Compare cost of all methods.

Singular Value Decomposition

$$A = U \Sigma V^T$$

$m \times n \quad m \times m \quad m \times n \quad n \times n$

→ complexity: $O(n^3)$

$$A = \begin{bmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_n \\ & & \emptyset & \end{bmatrix} \begin{bmatrix} - & v_1^T & - \\ & \vdots & \\ - & v_n^T & - \end{bmatrix}$$

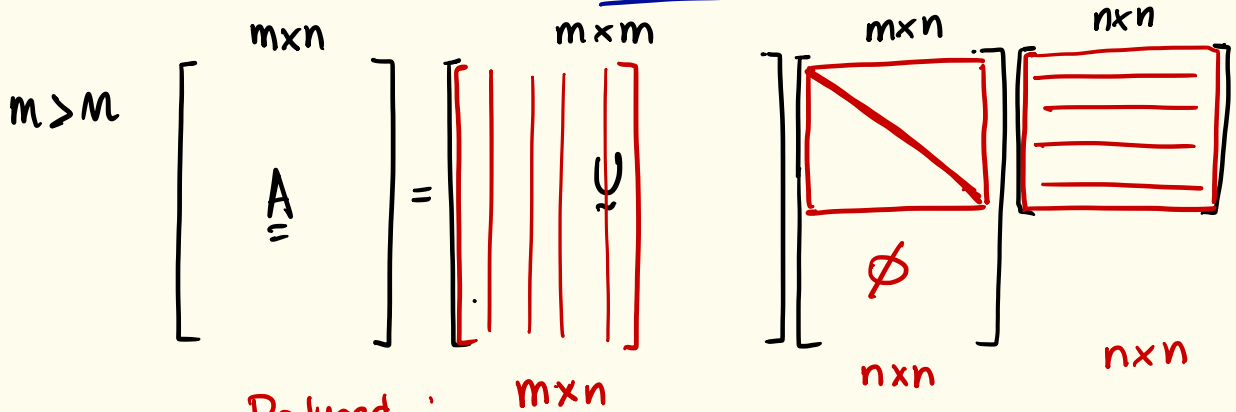
$m > n$

$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$

- Columns of V are eigenvectors of A^TA (right singular vectors)
- Columns of U are eigenvectors of AA^T (left singular vectors)
- σ_i^2 are eigenvalues of A^TA (singular values)

Note: A^TA → positive definite → positive eigenvalues (singular values are always positive!)

"Reduced" SVD


m > n
m×n
A
=
m×m
Ψ
m×n
Σ
n×n
n×n
Reduced : m×n
K = min(m,n) ⟶ (m×k) (k×k) (k×n)
Reduced : m×m m×m m×n
n > m
m×n
=
m×m
m×n
Σ
n×n

Low-rank approximations

$$\underset{\approx}{A} = \sigma_1 \underset{\sim}{u}_1 \underset{\sim}{v}_1^T + \sigma_2 \underset{\sim}{u}_2 \underset{\sim}{v}_2^T + \cdots + \sigma_n \underset{\sim}{u}_n \underset{\sim}{v}_n^T \qquad (m > n)$$

n singular values

$\underset{\approx}{A}$: $m \times n$ $\quad \underset{\sim}{u}_1$: $m \times 1$ $\quad \underset{\sim}{v}_1^T$: $1 \times n$

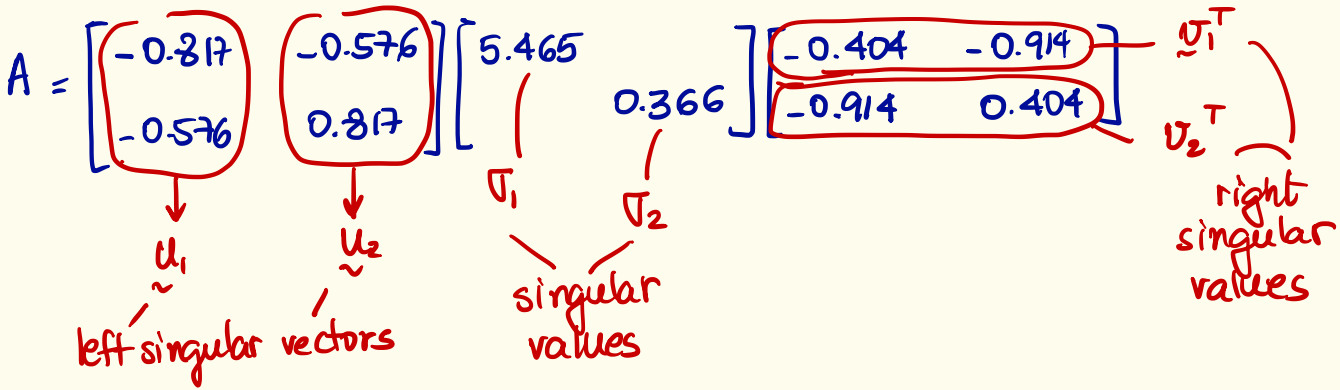
Best rank-k approximation for $m \times n$ matrix $\underset{\approx}{A}$ is solution of:

$$\min_{\underset{\approx}{A}_k} \| \underset{\approx}{A} - \underset{\approx}{A}_k \|_p$$

$$\text{s.t.} \quad \text{rank}(A_k) \leq k$$

when using $p=2$, the best rank-k approximation is:

$$\underset{\approx}{A}_k = \sigma_1 \underset{\sim}{u}_1 \underset{\sim}{v}_1^T + \sigma_2 \underset{\sim}{u}_2 \underset{\sim}{v}_2^T + \cdots + \sigma_k \underset{\sim}{u}_k \underset{\sim}{v}_k^T$$



(1) What is $\|A\|_2 = \max \sigma_i = 5.465$

(2) What is $\text{cond}_2(A) = \|A\|_2 \|A^{-1}\| = \frac{\sigma_{max}}{\sigma_{min}} = \frac{5.465}{0.366} = 14.9$

(3) Best rank-1 approximation

$$\underline{\underline{A}}_1 = \sigma_1 \underset{\sim}{u}_1 \underset{\sim}{v}_1^T = 5.465 \begin{bmatrix} -0.817 \\ -0.576 \end{bmatrix} \begin{bmatrix} -0.404 & -0.914 \end{bmatrix}$$

$$\text{error} \approx \|\underline{\underline{A}} - \underline{\underline{A}}_1\|_2 = \sigma_2$$

Linear Least-Squares

Given m data points : $(t_1, y_1), (t_2, y_2), \dots, (t_m, y_m)$

Find the n coefficients ($n < m$) of the function $f(t)$ that best fits the data

Eg. $f(t) = x_0 + x_1 t + x_2 t^2$

$$\begin{bmatrix} 1 & t_1 & t_1^2 \\ 1 & t_2 & t_2^2 \\ 1 & t_3 & t_3^2 \\ \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_m \end{bmatrix} \Longrightarrow \underset{=}{A}\,\underset{\sim}{x} \cong \underset{\sim}{b}$$

→ more equations than unknowns !

OVERDETERMINED SYSTEM

$$\Downarrow$$

$$\min_x \|\underset{=}{A}\underset{\sim}{x} - \underset{\sim}{b}\|_2^2$$

→ solution is unique if $\text{rank}(A) = n$

① $\underset{=}{A}^T\underset{=}{A}\,\underset{\sim}{x} = \underset{=}{A}^T\underset{\sim}{b}$ (Normal equations) → Cost : $O(n^3)$ → conditioning issues

② Use SVD : $\underset{\sim}{x} = \underset{=}{V}\,\underset{=}{\Sigma}^{+}\,\underset{=}{U}^T\underset{\sim}{b}$ where $\Sigma^{+} = \begin{cases} 1/\sigma_i & , \sigma_i \neq 0 \\ 0 & , \sigma_i = 0 \end{cases}$ → Cost: $O(n^3)$ (but more expensive)

① $\underline{\underline{A}}^T\underline{\underline{A}}\,\underline{x} = \underline{\underline{A}}^T\underline{b}$ (Normal equations)

$A_{m\times n} \longrightarrow A^TA_{n\times n}$ $\quad m>n$

→ Cost of factorization: $O(n^3)$

→ Cost of solving: $O(n^2)$

→ matrix can be very ill-conditioned

→ solution exists if rank(A) = n

② Use SVD: $\underline{x} = \underline{\underline{V}}\,\underline{\underline{\Sigma}}^+\underline{\underline{U}}^T\underline{b}$ where $\Sigma^+ = \begin{cases} 1/\sigma_i, & \sigma_i \neq 0 \\ 0, & \sigma_i = 0 \end{cases}$

→ Cost of factorization: $O(n^3)$

→ Cost of solving: $O(mn)$

$z = \underset{(m\times n)^T}{U^T}\underset{m\times 1}{b} \rightarrow O(mn)$

$y = \Sigma^T z \rightarrow O(n)$

$Vy \rightarrow O(n^2)$

→ solution always exist

- unique if rank(A) = n $\longrightarrow$ $x = V\Sigma^{-1}U^Tb$
- rank(A) < n $\longrightarrow$ $x = V\Sigma^+U^Tb$

Interpolation

Given m data points : $(t_1, y_1), (t_2, y_2), \ldots, (t_m, y_m)$

Find the function $f(t)$ – interpolant – such that

$$f(t_i) = y_i \quad , \quad i = 0, \ldots, m-1$$

$$f(t) = \sum_{j=0}^{n-1} x_j \phi_j(t)$$ ϕ_j : basis functions (n linearly independent basis functions)

Generalized Vandermonde matrix :

$$\begin{bmatrix} \phi_0(t_0) & \phi_1(t_0) & \cdots & \phi_{n-1}(t_0) \\ \phi_0(t_1) & \phi_1(t_1) & \cdots & \phi_{n-1}(t_1) \\ \vdots & & & \\ \phi_0(t_{m-1}) & \phi_1(t_{m-1}) & \cdots & \phi_{n-1}(t_{m-1}) \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{m-1} \end{bmatrix}$$

$m \times n$ $\quad n \times 1$ $\quad m \times 1$

$$\Rightarrow \underset{\sim}{A}\,\underset{\sim}{x} = \underset{\sim}{b}$$

$m = n \rightarrow$ unique interpolant

Interpolation with monomials

$$p_{n-1}(t) = \sum_{j=0}^{n-1} x_j t^j \longrightarrow \text{polynomial of degree } (n-1)$$

in matrix form, this yields the Vandermonde Matrix

If the points t_i are equally spaced on an interval of length h, and the "true function" we are trying to interpolate is sufficiently smooth, then the error of the interpolant of degree $=(n-1)$ is:

$$\text{error} = O(h^n) = O(h^{\text{degree}+1})$$

Nonlinear Equation (1D)

Solve $f(x) = 0$ for $f: \mathbb{R} \to \mathbb{R}$ (Root Finding)

1) Bisection method:
- no need for derivatives
- one function evaluation per iteration
- linear convergence
- $h_{k+1} = 0.5\, h_k$

2) Newton's method:
- $x_{k+1} = x_k + f(x_k)/f'(x_k)$
- need to evaluate one function value and one derivative for each iteration
- quadratic convergence

3) Secant method:
- need to initial guesses (x_0, x_1)
- $\hat{f}'(x_k) = (f(x_k) - f(x_{k-1}))/(x_k - x_{k-1})$ → derivative approx.
- $x_{k+1} = x_k - f(x_k)/\hat{f}'(x_k)$
- no need for derivatives
- one function evaluation per iteration
- superlinear convergence

System Nonlinear Equations

Solve $\underset{\sim}{f}(\underset{\sim}{x}) = \underset{\sim}{0}$ for $f: \mathbb{R}^n \to \mathbb{R}^n$

1) Newton's Method
- $\underset{\sim}{x}_0$: initial guess
- solve $\underline{\underline{J}}(x_k)\, \underset{\sim}{s}_k = -f(x_k)$
- $\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + \underset{\sim}{s}_k$
- typically quadratic convergence
- local convergence
- main cost: computing $\underline{\underline{J}}(\underset{\sim}{x}_k)$ and solve

$$\left[\underline{\underline{J}}(\underset{\sim}{x}_k)\right]_{ij} = \frac{\partial f_i}{\partial x_j}$$

$$\underline{\underline{J}}(\underset{\sim}{x}_k) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

2) Secant method (eg. Broyden)
→ use a approximation for the Jacobian that satisfies

$$\hat{J}\,(\underset{\sim}{x}_{k+1} - \underset{\sim}{x}_k) = \underset{\sim}{f}(\underset{\sim}{x}_{k+1}) - \underset{\sim}{f}(\underset{\sim}{x}_k)$$

3) Finite Difference Method

Approximate partial derivatives as:

$$\frac{\partial f_i}{\partial x_j} \approx \frac{f_i(\underset{\sim}{x} + h\underset{\sim}{\delta}_j) - f_i(\underset{\sim}{x})}{h}$$

$$(\delta_j)_k = \begin{cases} 0, & k \neq j \\ 1, & k = j \end{cases}$$

$$\underset{\sim}{\delta}_j = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \end{bmatrix} \leftarrow j^{th} \text{ component}$$

Optimization 1D

$$\begin{cases} \min_x f(x) \longrightarrow \text{objective (cost) function} \\ \text{s.t. } g(x)=0 \longrightarrow \text{equality constraint} \\ \quad\; h(x)\leq 0 \longrightarrow \text{inequality constraint} \end{cases}$$

solution x^* is the minimizer.

Unconstrained optimization

First order necessary condition: $f'(x)=0$ → if $f'(x^*)=0$ → x^* is critical point

Second order sufficient condition: if $f''(x^*)>0$ → x^* is a minimizer

1) Golden Section Search

- guaranteed to converge to global minimum for unimodal functions
- no need for derivatives
- only one function evaluation per iteration
- $h_{k+1} = 0.618\, h_k$
- linear convergence

2) Newton's Method

- $x_{k+1} = x_k - f'(x_k)/f''(x_k)$
- typically quadratic convergence
- local convergence
- need to evaluate $f'(x_k)$ and $f''(x_k)$ for each iteration
- may fail to converge (or converge to a max or saddle, since it does not check sign of $f''(x)$)

Optimization ND

Unconstraind Optimization : $\min_{\underset{\sim}{x}} f(\underset{\sim}{x})$ $\quad f: \mathbb{R}^n \to \mathbb{R}$

First order necessary condition: $\nabla f(\underset{\sim}{x}) = \underset{\sim}{0}$ → if $\nabla f(\underset{\sim}{x}^*) = \underset{\sim}{0}$ → $\underset{\sim}{x}^*$ is critical point

Second order sufficient condition : $H_f(x) = \nabla^2 f(x)$

if $H_f(x^*)$ positive definite → minimum

$H_f(x^*)$ negative definite → maximum

$H_f(x^*)$ indefinite → saddle

1) Steepest Descent

$\underset{\sim}{x}_0$: initial guess

$\underset{\sim}{s}_k = -\nabla f(\underset{\sim}{x}_k)$

$\alpha_k^* = \underset{\alpha}{\operatorname{argmin}} \left(f(\underset{\sim}{x}_k + \alpha \underset{\sim}{s}_k) \right)$

$\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + \alpha_k^* \underset{\sim}{s}_k$

- linear convergence

2) Newton's Method

$\underset{\sim}{x}_0$: initial guess

solve $H_f(\underset{\sim}{x}_k)\, \underset{\sim}{s}_k = -\nabla f(\underset{\sim}{x}_k)$

$\underset{\sim}{x}_{k+1} = \underset{\sim}{x}_k + \underset{\sim}{s}_k$

- typical quadratic convergence
- need Hessian ☹
- local convergence
- high computational cost