Numerical Solutions

Goal is to find $\underline{x}$ such that $\underline{A}\underline{x} = \underline{b}$

If $\det(\underline{A}) \neq 0$, $\underline{A}^{-1}$ exists, $\Rightarrow \underline{x} = \underline{A}^{-1}\underline{b}$

You **never** actually find $\underline{A}^{-1}$.

2 methods to solve $\underline{A}\underline{x} = \underline{b}$

1) Iteration Methods: Use matrix-vector products & projection to iterate a sequence of vectors $\underline{y}$ that converge to the solution $\underline{x}$

GMRES, BiCGSTAB, QMR, etc.

Most are Krylov Subspace methods.

"Iterative Methods for Sparse Linear Systems" by Yousef Saad.

2) Decomposition Methods: Given $\underline{A}$ find $\underline{B}$ & $\underline{C}$ such that $\underline{A} = \underline{C}\,\underline{B}$ then solve $\underline{C}\underline{y} = \underline{b}$ & $\underline{B}\underline{x} = \underline{y}$

$$\underline{y} = \underline{C}^{-1}\underline{b} \Rightarrow \underline{B}\underline{x} = \underline{y} = \underline{C}^{-1}\underline{b} \Rightarrow \underline{x} = \underline{B}^{-1}\underline{C}^{-1}\underline{b}$$

$$\Rightarrow \underline{C}\,\underline{B}\,\underline{x} = \underline{b} \Rightarrow \underline{A}\underline{x} = \underline{b}$$

LU, QR, SVD, etc.

These are "exact" methods

In practice they are not exact due to finite precision

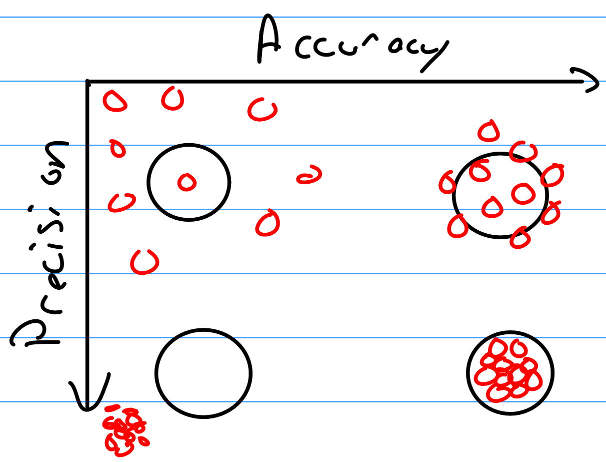
1) How computers store information
2) finite precision
3) How errors in $\underline{A}$ & $\underline{b}$ influence the errors in $\underline{x}$.

Accuracy vs. Precision

Let the true answer be "truth".
We have approximate solutions

Accuracy: How close are the approximate solutions to truth

Precision: How close are the approximations to each other.



Numerical Errors come from:

- Choice of the model
- Numerical Approximations
- Data representation
- Implementation
- Fluctuations in Hardware

Truth = Approximation + Error

$x^* = x + \varepsilon$

$\Rightarrow \varepsilon = x^* - x$ ← not usually a good measure

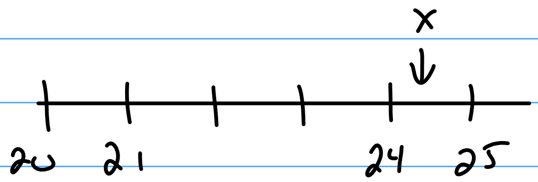
Relative error: $e_{rel} = \dfrac{x^* - x}{x^*}$

Note: In iterative methods you get

$\underline{y}_1, \underline{y}_2, \underline{y}_3, \ldots, \underline{y}_n \to \underline{x}$

$\underline{r}_i = \underline{A}\,\underline{y}_i - \underline{b}$ = residual (not error!)

Significant Figures



$24 < x < 25$

$x = \cancel{24.25}?$

$\cancel{24.225}?$

$24.4?$

Sig figs tell how much useful information you have: Typically the # of certain digits plus one.

ex.) 53800 ← 5 sig figs

5.38×10^4 ← 3 sig figs

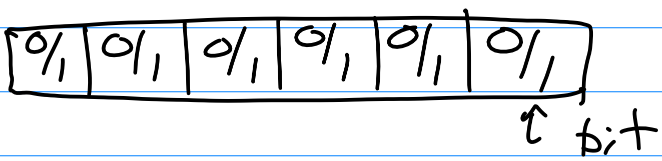
5.380×10^4 ← 4 sig figs

Trailing zeros count, leading zeros do not.

0.01234 ← 4 sig figs

$\underbrace{1.234} \times 10^{-2}$

Computers store numbers in binary: 0 & 1



finite # of bits = finite precision

32-bit : 7 sig figs (single)

64-bit : 15 sig figs (double)

$$\frac{3.25x}{1.96x} = 1.65|816\ldots$$

$$\frac{3.259}{1.960} = 1.6627$$

$$\frac{3.250}{1.969} = 1.65$$

Typically round or chop

1.65932 to 2 sig figs

rounded = 1.66

chop = 1.65

Round-off errors

Data in a computer must have a type

Integer: ..., −1, 0, 1, 2, ...

float/double/long double: Decimals

1.39, π (up to a #)

characters: 'a', 'b', etc.

Consider $\frac{7}{2} = 3.5$

let 2 & 7 be integers & the result must also be an integer

$\Rightarrow \frac{7}{2} = 3 \Rightarrow$ round-off error of 0.5

In double precision you can represent the difference of numbers larger than

$$2^{-52} \sim 2.22 \times 10^{-16}$$

In double precision

$$1.0 \times 10^{0} = 1.0 \times 10^{0} + 10^{-20}$$

$$1. = 1. + 10^{-20}$$

In Matlab: eps gives the machine precision

This also why when comparing floating point numbers do <u>not</u> use equals.

$a \cancel{==} b \Rightarrow$ use $\underbrace{\text{abs}(a-b) < \text{tol}}$

Round off error example

3 sig fig w/ chopping

$\underbrace{0.990}$ + $0.00\underbrace{440}$ + $0.00\underbrace{490}$ = 0.9993 (exact)

$(0.990 + 0.00440) + 0.00490$
$0.994 + 0.00490 = 0.998$

$0.990 + (0.00440 + 0.00490)$
$0.990 + 0.00930 = 0.999$

Frequent in division & subtraction

You can modify the algorithm to minimize

roots of $x^2 + bx + 1 = 0$ b is large

$r = (b^2 - 4)^{1/2}$ $x_1 = \frac{b+r}{2}$ $x_2 = \frac{b-r}{2}$

If b is large $b \approx r$

$b = 110$ $r = (110^2 - 4)^{1/2} \approx 109.|9818$

Truth: $x_1 = 109.99...$
$x_2 = 0.00909166...$

Solve on 3 sig fig machine + chopping

$b = 110 \qquad r = 109$

$x_1 = \frac{b+r}{2.00} = \frac{110+109}{2.00} = \frac{219}{2.00} = 109$ ← 3 digits of accuracy

↑ Actually $109.\cancel{5}$

$x_2 = \frac{b-r}{2} = \frac{110-109}{2.00} = \frac{1.00}{2.00} = 0.500$ ← no accurate digits

but, $x_2 = \frac{b-r}{2} = \frac{b-r}{2}\,\frac{b+r}{b+r} = \frac{b^2-r^2}{2(b+r)} = \frac{4}{2(b+r)} = \frac{2}{b+r}$

$\Rightarrow x_2 = \frac{2.00}{110+109} = \frac{2.00}{219} = 0.00913$ vs. 0.00909

↑ one digit of accuracy

Why do we care? Solutions to $A\underline{x} = \underline{b}$ involve $+, -, *, \div$

Repeated applications of math operations introduce errors that accumulate

A **stable algorithm** does not break down due to these errors

Condition Number

The condition number of a matrix $\underline{A}$ tells us how errors in $\underline{b}$ influence errors in $\underline{x}$ when solving $\underline{A}\,\underline{x} = \underline{b}$

Instead of $\underline{A}\,\underline{x} = \underline{b}$ you solve

$$\underline{A}\,(\underline{x} + \underbrace{\underline{\Delta x}}_{\uparrow}) = (\underline{b} + \underbrace{\underline{\Delta b}}_{\uparrow})$$

error induced by $\underline{A}$ error in $\underline{b}$

Goal: Relate $\dfrac{\|\underline{\Delta x}\|}{\|\underline{x}\|} \;\alpha\; \dfrac{\|\underline{\Delta b}\|}{\|\underline{b}\|}$

$$\cancel{\underline{A}\,\underline{x}} + \underline{A}\,\underline{\Delta x} = \cancel{\underline{b}} + \underline{\Delta b} \implies \underline{A}\,\underline{\Delta x} = \underline{\Delta b}$$

$$\|\underline{\Delta b}\| = \|\underline{A}\,\underline{\Delta x}\| \leq \|\underline{A}\|\,\|\underline{\Delta x}\|$$

Also, $\underline{\Delta x} = \underline{A}^{-1}\underline{\Delta b} \implies \|\underline{\Delta x}\| = \|\underline{A}^{-1}\underline{\Delta b}\| \leq \|\underline{A}^{-1}\|\,\|\underline{\Delta b}\|$

$$\underline{b} = \underline{A}\,\underline{x} \implies \|\underline{b}\| = \|\underline{A}\,\underline{x}\| \leq \|\underline{A}\|\,\|\underline{x}\|$$

$$\frac{\|\underline{\Delta x}\|}{\|\underline{x}\|} \leq \|\underline{A}^{-1}\|\,\|\underline{\Delta b}\|\,\frac{\|\underline{A}\|}{\|\underline{b}\|}$$

$$\frac{\|\underline{\Delta x}\|}{\|\underline{x}\|} \leq \left(\|\underline{A}^{-1}\|\,\|\underline{A}\|\right)\frac{\|\underline{\Delta b}\|}{\|\underline{b}\|}$$

Define the condition number of $\underline{A}$ as

$$\kappa(\underline{A}) = \|\underline{A}\| \, \|\underline{A}^{-1}\|$$

$\kappa(\underline{A})$ indicates how errors grow

ex.) let $\frac{\|\Delta b\|}{\|b\|} \sim 10^{-16}$ w/ $\kappa(\underline{A}) \sim 10^{10}$

$$\frac{\|\Delta x\|}{\|\underline{x}\|} \leq \kappa(\underline{A}) \frac{\|\Delta b\|}{\|b\|} \sim 10^{-6}$$

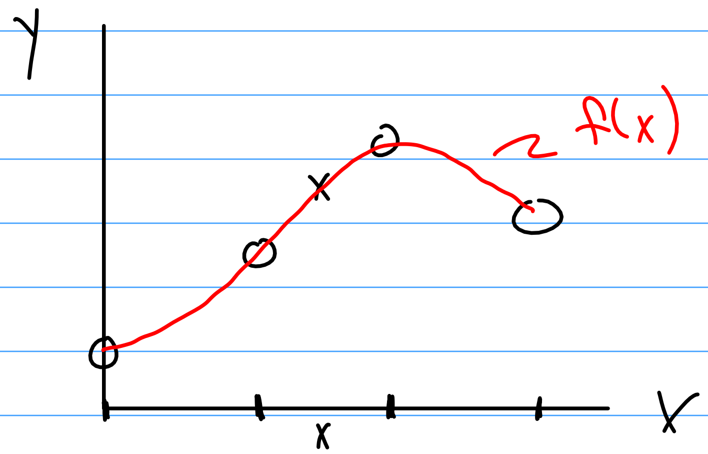
In matlab: cond, rcond, condest

Also why use backslash: $x = A \backslash b$, not

$x = inv(A) * b$

Interpolation

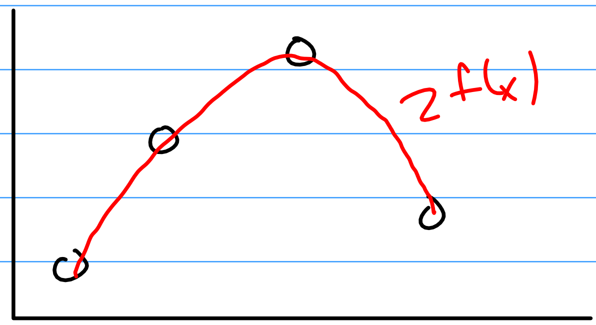
Given a data set, determine values not in the data set.



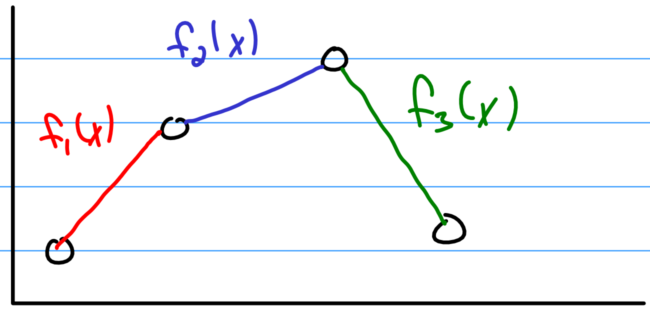
Goal: Get f(x)

Two main types:

1) Global interpolants: Use all of the data to construct one f(x) over the entire range



2) Piecewise Interpolation: One equation between data points



Polynomial Interpolation (Global)

let n-data points $(x_1, y_1) \rightarrow (x_n, y_n)$ be given w/ $x_i \neq x_j$ if $i \neq j$

Construct an $n-1$ polynomial:

$$f(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-2} x^{n-2} + a_{n-1} x^{n-1}$$

$a_0 \rightarrow a_{n-1}$ unknowns

Require that $f(x_1) = y_1$, $f(x_2) = y_2$, etc.

$$f(x_1) = a_0 + a_1 x_1 + a_2 x_1^2 + \cdots + a_{n-1} x_1^{n-1} = y_1$$
$$f(x_2) = a_0 + a_1 x_2 + a_2 x_2^2 + \cdots + a_{n-1} x_2^{n-1} = y_2$$
$$\vdots$$
$$f(x_n) = a_0 + a_1 \underset{\text{known}}{\underset{\uparrow}{x_n}} + \cdots + \underset{\text{unknown}}{\underset{\uparrow}{a_{n-1}}} x_n^{n-1} = y_n$$

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ \vdots & \vdots & & & \\ 1 & x_n & \cdots & & x_n^{n-1} \end{bmatrix}}_{\underline{V}} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix}}_{\underline{a}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}}_{\underline{y}}$$

If $x_i \neq x_j$ if $i \neq j$, $\det(\underline{V}) \neq 0$

$\Rightarrow \underline{a} = \underline{V}^{-1} \underline{y}$ exists!

but, $\kappa(\underline{V})$ is huge!

$\underline{V}$ is called the Vandermonde matrix

ex.) let $(300, 1)$, $(400, 1.1)$, $(500, 1.3)$ be given

$$\underbrace{\begin{bmatrix} 1 & 300 & 90000 \\ 1 & 400 & 16000 \\ 1 & 500 & 25000 \end{bmatrix}}_{\kappa(\underline{V}) \sim 6 \times 10^6} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.1 \\ 1.3 \end{bmatrix}$$

It turns out that while the polynomial interpolant is unique, the method to compute it is not.

Next time: Lagrange Interpolation

$$f(x) = y_1 L_1(x) + y_2 L_2(x) + \dots + y_n L_n(x)$$

$L_i(x)$ = Lagrange function / polynomial