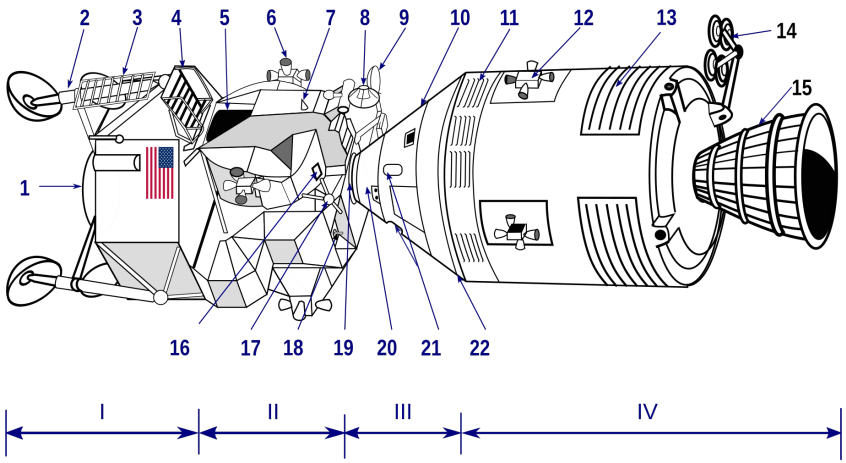
Welcome to
ASEN 3502
Computational Methods

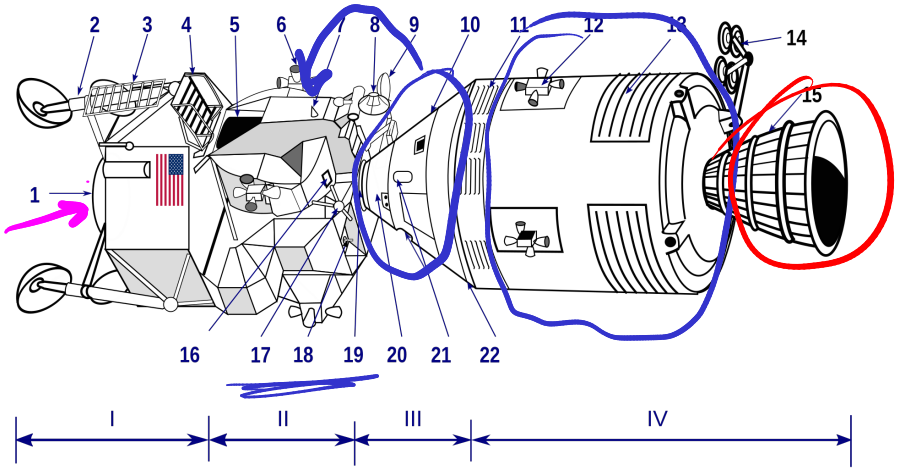
Apollo 13



I–II: lunar module (descent, ascent stages); III: command module; IV: service module.

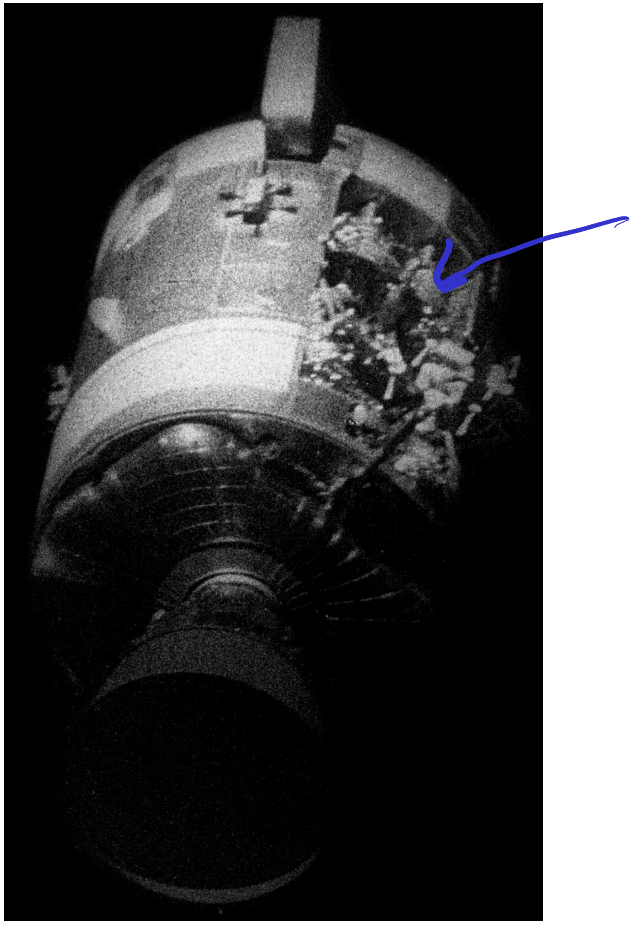
Popadius, Wikimedia Commons, CC0.

Apollo 13



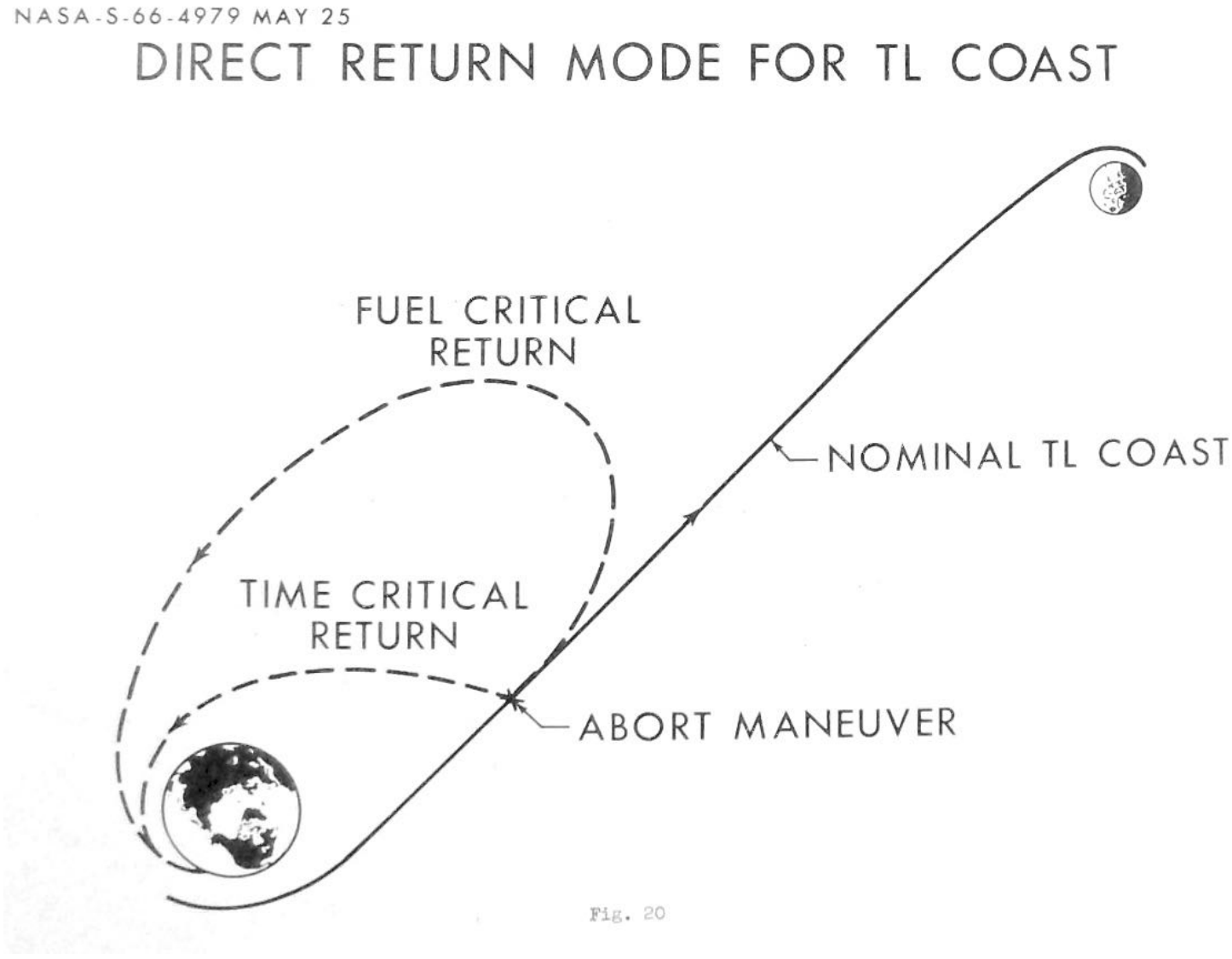
I–II: lunar module (descent, ascent stages); III: command module; IV: service module.

Popadius, Wikimedia Commons, CC0.

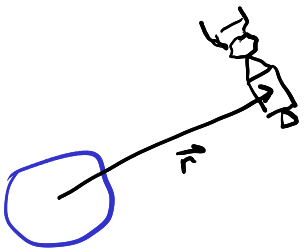


The service module after separation, with one panel blown off. NASA (scan by Kipp Teague), public domain.

Direct return

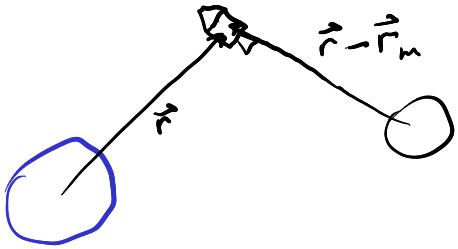


R. L. Berry, *Apollo Lunar Landing Mission Symposium*, NASA MSC, June 1966. Public domain, via Wikimedia Commons.



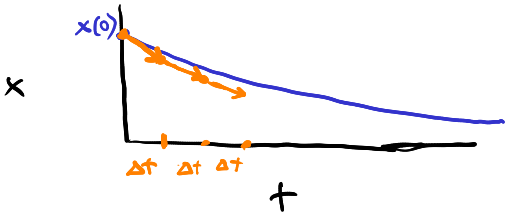
$$\ddot{\vec{r}} = -\frac{\mu_e}{r^2}\frac{\vec{r}}{r} \rightarrow \text{elliptical hyperbolic}$$

$$\mu_e = G m_e$$



$$\ddot{\vec{r}} = -\frac{\mu_e}{r^2}\frac{\vec{r}}{r} - \frac{\mu_m}{\|\vec{r}-\vec{r}_m\|^2}\frac{\vec{r}-\vec{r}_m}{\|\vec{r}-\vec{r}_m\|}$$

no easy analytical solution



$$\dot{x} = -ax$$

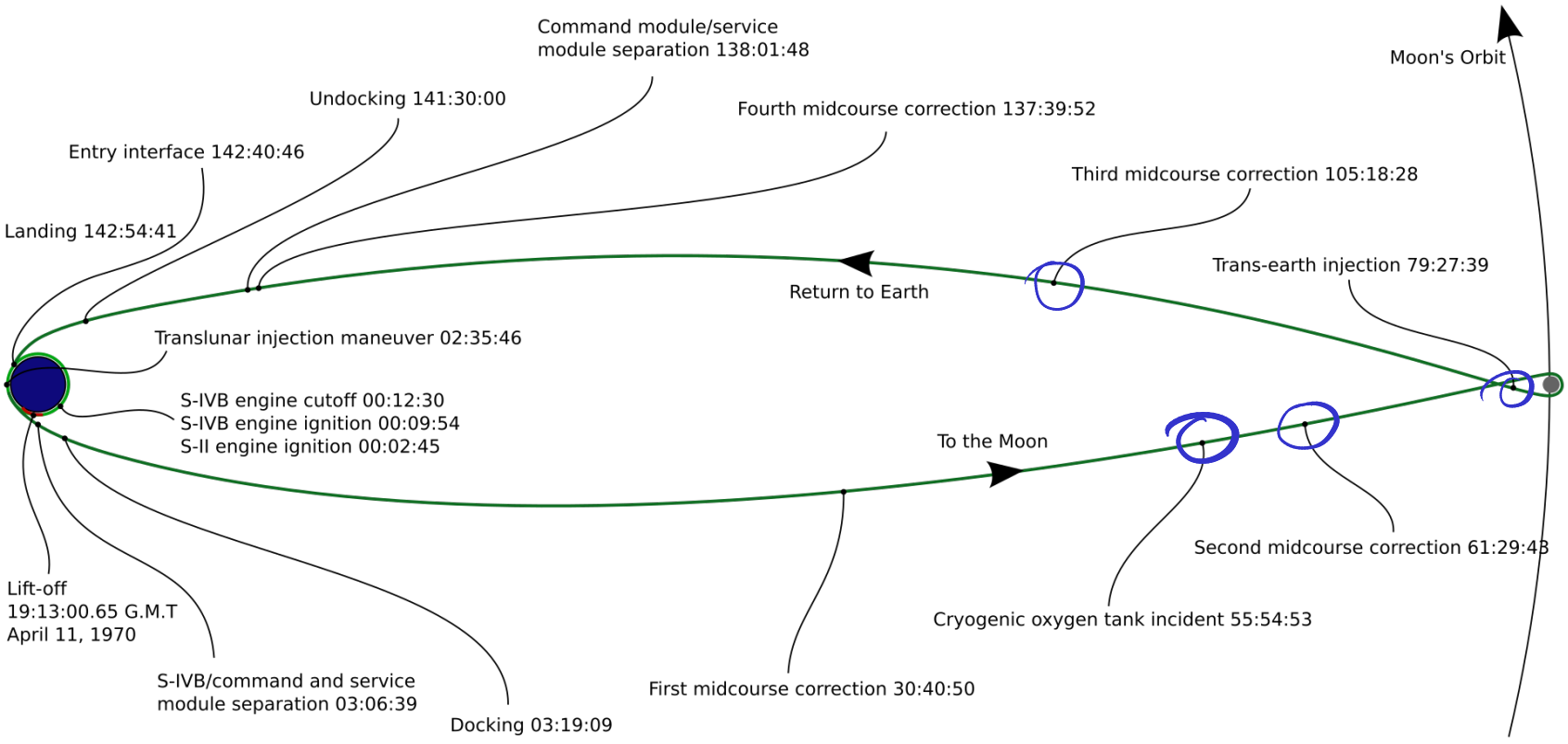
Euler's Method

Error = $O(\Delta t)$

Runge-Kutta

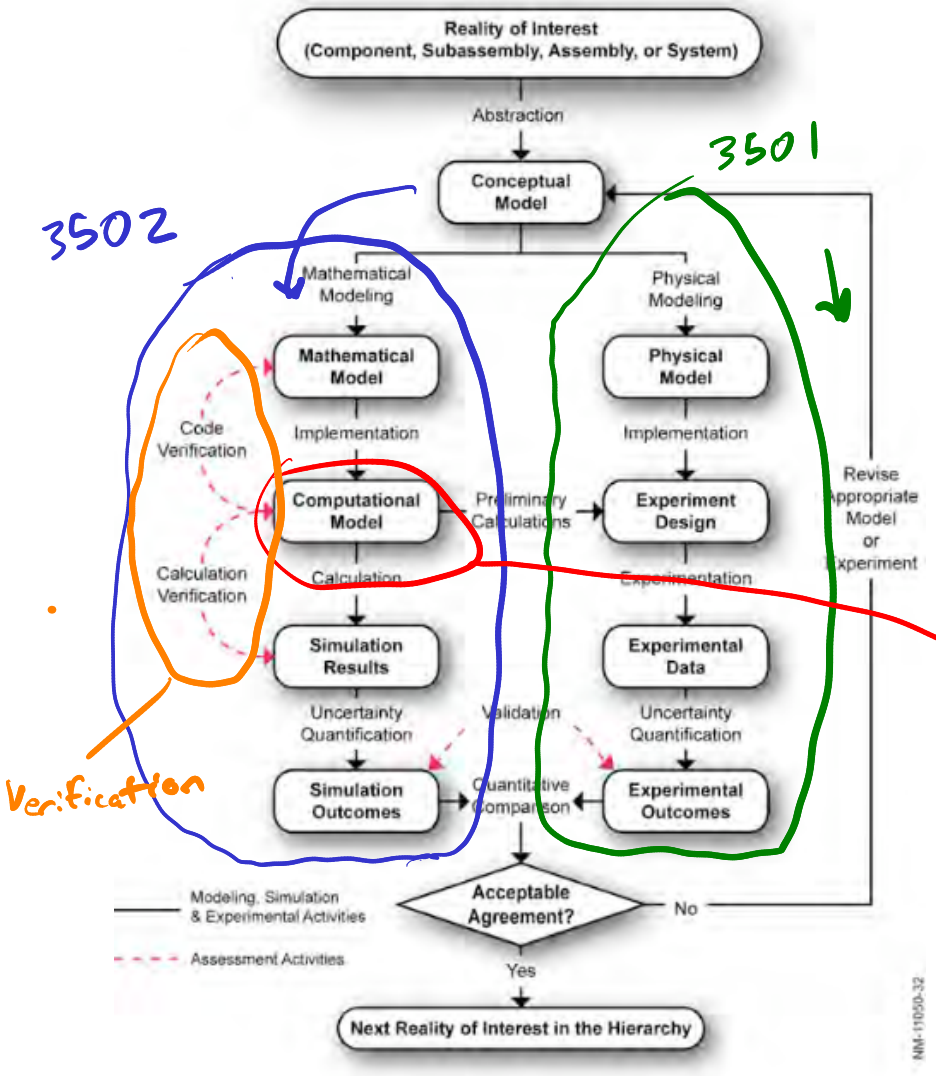
Error = $O(\Delta t^4)$

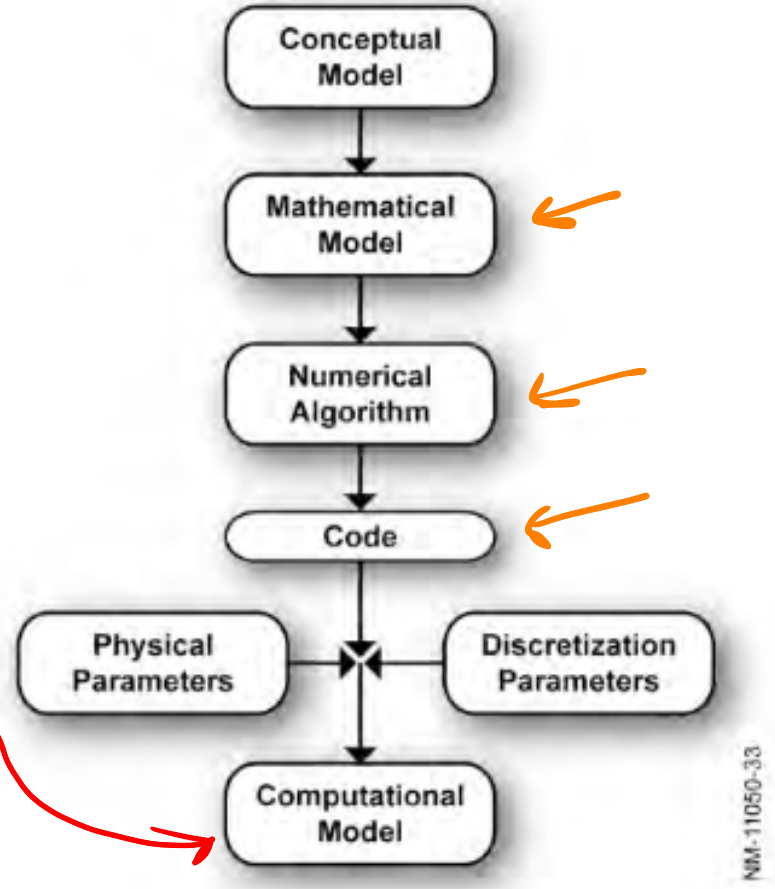
What Apollo 13 actually flew



AndrewBuck, Wikimedia Commons, CC BY-SA 4.0. Source for figures: Apollo 13 Mission Report, p. 3-2.

ASME V&V 10





Your code is not done until you verify it with tests

ASME V&V 10, *Guide for Verification and Validation in Computational Solid Mechanics.*

asen3502.com

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn
- It may take multiple ways of explaining a concept before a student understands

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn
- It may take multiple ways of explaining a concept before a student understands
- We must work together to help you find the way to understand

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn
- It may take multiple ways of explaining a concept before a student understands
- We must work together to help you find the way to understand
- Taking notes in class and doing the homework yourselves are critical steps to understanding

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn
- It may take multiple ways of explaining a concept before a student understands
- We must work together to help you find the way to understand
- Taking notes in class and doing the homework yourselves are critical steps to understanding
- I am trying to bring you to the point of understanding, not just give you the answer

Our Assumptions

- The material is challenging, and can take multiple engagements to understand fully
- A student who comes to my office or asks a question wants to learn
- It may take multiple ways of explaining a concept before a student understands
- We must work together to help you find the way to understand
- Taking notes in class and doing the homework yourselves are critical steps to understanding
- I am trying to bring you to the point of understanding, not just give you the answer
- We will have fun this semester

8/26

Today

- Linear Systems of Equations
- Gaussian Elim.
- Partial Pivoting

$$2x_1 + x_2 = 4$$
$$-x_1 + x_2 = 1$$

Unkowns: $x_1, \dots x_n$

Linear Eqn.: $\sum_{j=1}^{n} a_j x_j = b$

no nonlinear functions of x_i: x^2, e^x, $\log(x)$, $\sin(x)$ etc.

System of m linear equations $\sum_{j=1}^{n} a_{ij} x_j = b_i \quad i = 1 \dots m$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & & \\ a_{m1} & & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_m \end{bmatrix}$$

Book: $[A]\{x\} = \{b\}$

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

for now $m = n$

HW0 $\Rightarrow A \cdot x$

Lab $Ax = b$ (known, known, unknowns)

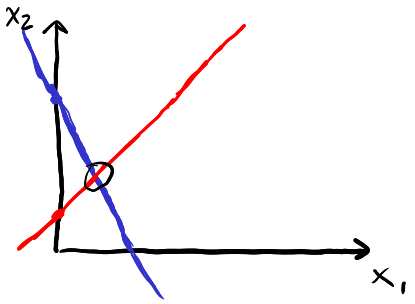
$2x_1 + x_2 = 4$

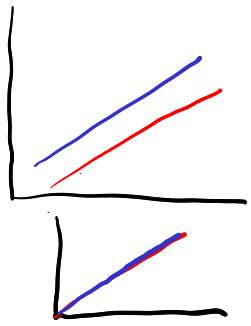
$x_2 = -2x_1 + 4$

$-x_1 + x_2 = 1$

$x_2 = x_1 + 1$

$x_1 = 1, \; x_2 = 2$





Example:

$$\begin{bmatrix} 3 & -1 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 6 \end{bmatrix}$$

$3x_1 - x_2 = 0 \rightarrow 3x_1 - 3 = 0$

$x_1 = 1$

$2x_2 = 6$

$x_2 = 3$

$$\begin{bmatrix} 1 & & & & \\ 0 & 1 & & & \\ \vdots & \ddots & \ddots & & \\ 0 & \cdots & 0 & 1 & \\ 0 & \cdots & \cdots & 0 & 1 \end{bmatrix}$$

$$2x_1 + 3x_2 + 4x_3 = 19 \quad (1)$$
$$4x_1 + 11x_2 + 14x_3 = 55 \quad (2)$$
$$2x_1 + 8x_2 + 17x_3 = 50 \quad (3)$$

$$\left[\begin{array}{ccc|c} 2 & 3 & 4 & 19 \\ 4 & 11 & 14 & 55 \\ 2 & 8 & 17 & 50 \end{array}\right]$$

f

$-2 \cdot (1)$

$-1 \cdot (1)$

$$\left[\begin{array}{ccc|c} 2 & 3 & 4 & 19 \\ 0 & 5 & 6 & 17 \\ 0 & 5 & 13 & 31 \end{array}\right]$$

$-1 \cdot (2)$

$$\left[\begin{array}{ccc|c} 2x_1 + 3x_2 + 4x_3 & = & 19 \\ 0 & 5x_2 + 6x_3 & = & 17 \\ 0 & 0 & 7x_3 & = & 14 \end{array}\right]$$

$x_3 = 2$

$2x_1 + 3 \cdot 1 + 4 \cdot 2 = 19$

$2x_1 = 8$

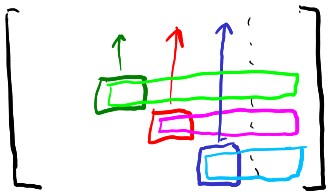
$x_1 = 4$

$5x_2 + 12 = 17$

$x_2 = 1$

Eliminate ▭ with ▭
Eliminate ▭ with ▭
etc

Back substitution



Solve for ▭ with ▭
Solve for ▭ with ▭
etc.

Example:

$$A = \begin{bmatrix} 1 & 2 & -1 \\ 3 & 9 & -2 \\ 2 & 1 & -1 \end{bmatrix} \qquad b = \begin{bmatrix} 1 \\ 9 \\ 2 \end{bmatrix}$$

$$x = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix}$$

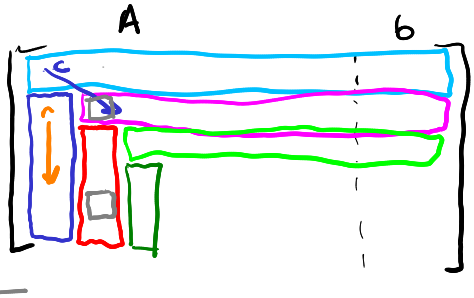
input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output: solution $\{x\}$ of $[A]\{x\} = \{b\}$

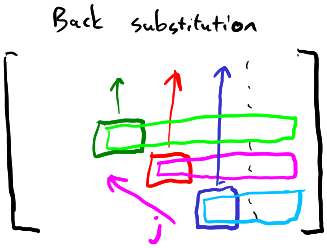
Elimination

```
// forward elimination
for c ← 1 to n − 1 do
    for r ← c + 1 to n do
        f ← A[r, c] / A[c, c]
        A[r, c:n] ← A[r, c:n] − f · A[c, c:n]
        b[r] ← b[r] − f · b[c]
    end
end
```

Backsub

```
// back substitution
for r ← n, n − 1, . . . , 1 do
    s ← b[r]
    for j ← r + 1 to n do
        s ← s − A[r, j] · x[j]
    end
    x[r] ← s / A[r, r]
end
```





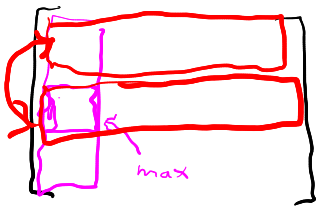
$$A = \begin{bmatrix} 0 & 2 & 3 \\ 4 & 6 & 7 \\ 2 & -3 & 6 \end{bmatrix} \qquad b = \begin{bmatrix} 8 \\ -3 \\ 5 \end{bmatrix}$$

$f \leftarrow Aug[1,2] / Aug[1,1]$ Divide by Zero

Solution: Partial Pivoting

input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output: solution $\{x\}$ of $[A]\{x\} = \{b\}$

```
// forward elimination
for c ← 1 to n − 1 do
    // find the largest entry in column c
    p ← argmax(abs(A[c:n, c])) + c − 1
    // swap that row into the pivot position
    A[c, :] ↔ A[p, :]
    b[c] ↔ b[p]
    for r ← c + 1 to n do
        f ← A[r, c] / A[c, c]
        A[r, c:n] ← A[r, c:n] − f · A[c, c:n]
        b[r] ← b[r] − f · b[c]
    end
end
// back substitution
for r ← n, n − 1, . . . , 1 do
    s ← b[r]
    for j ← r + 1 to n do
        s ← s − A[r, j] · x[j]
    end
    x[r] ← s / A[r, r]
end
```



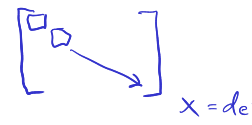
Algorithms and Operation Counting

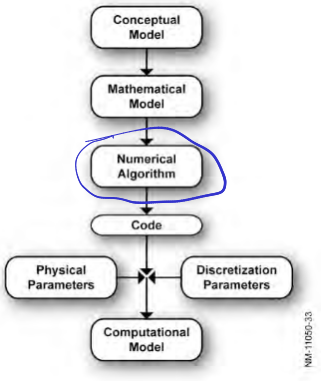
ASEN 3502 — Computational Methods

Please grab a handout in the back!

HW 1 - Due Thursday

Lab min through 1.3
rec through 1.5





Today
- Algorithms
- Operation Counting
- Asymptotic Notation

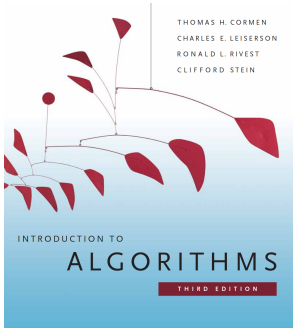
$O(n)$

ASME V&V 10, *Guide for Verification and Validation in Computational Solid Mechanics*.

What is an algorithm?

"Informally, an algorithm is any well-defined computational **procedure** that takes some value, or set of values, as **input** and produces some value, or set of values, as **output**."

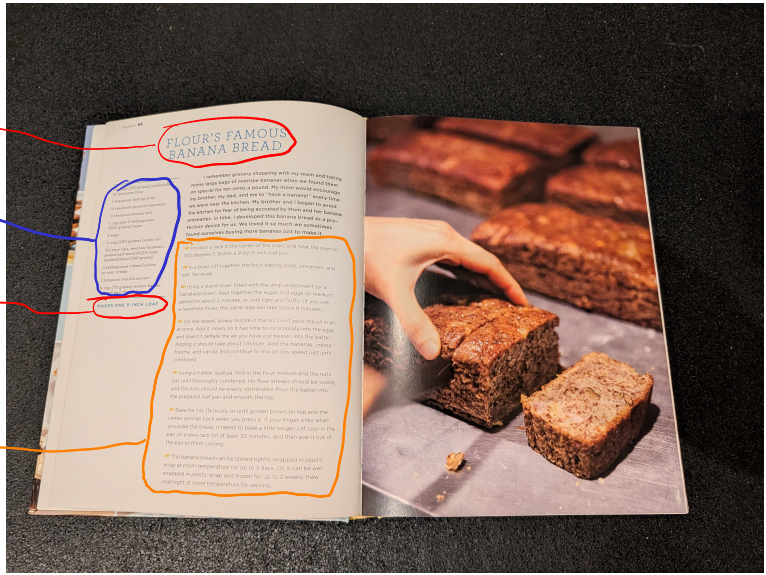
— Cormen et al.



Introduction to Algorithms
Cormen, Leiserson,
Rivest, and Stein

mitpress.mit.edu

Another type of algorithm



Why algorithms?

- Algorithms are mathematical objects. Why are they useful?

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

    > "An algorithm is said to be **correct** if, for every input instance, it halts with the correct output." — Cormen et al.

Why algorithms?

- Algorithms are mathematical objects. Why are they useful?
  - We can reason about properties like correctness and resource use

    > "An algorithm is said to be **correct** if, for every input instance, it halts with the correct output." — Cormen et al.
  - We can make precise and rigorous statements without having to deal with issues like roundoff error, memory allocation, etc.

How do we represent algorithms?

Matrix-vector multiplication: three ways

English

To get entry r of $\{b\}$, multiply each entry in row r of $[A]$ by the matching entry of $\{x\}$, and add up the results. Do this for every row.

Code

```
b = zeros(n,1);
for r = 1:n
    for c = 1:n
        b(r) = b(r) ...
             + A(r,c)*x(c);
    end
end
```

Pseudocode

input : $n \times n$ matrix $[A]$,
vector $\{x\}$
output : $\{b\} = [A]\{x\}$

for $r \leftarrow 1$ **to** n **do**
  $b[r] \leftarrow 0$
  for $c \leftarrow 1$ **to** n **do**
    $b[r] \leftarrow b[r] + A[r, c] \cdot x[c]$
  end
end

"What separates pseudocode from "real" code is that in pseudocode, we employ whatever expressive method is most clear and concise to specify a given algorithm." — Cormen et al.

Analysis of algorithms

- *Analyzing* an algorithm means predicting the resources that an algorithm requires
  - For today, we will count floating point operations ("Flops"): multiplication, division, addition, and subtraction of floating point numbers

Analysis of algorithms

- *Analyzing* an algorithm means predicting the resources that an algorithm requires
  - For today, we will count floating point operations ("Flops"): multiplication, division, addition, and subtraction of floating point numbers

- **Discuss with your neighbor:** Consider the equation $[A]\{x\} = \{b\}$. As n gets larger, which requires more floating point operations? Finding $\{x\}$ given $[A]$ and $\{b\}$, or finding $\{b\}$ given $[A]$ and $\{x\}$?

① $b = Ax$

② $Ax = b$ Harder if using Gaussian Elimination

Matrix-vector multiplication

input : $n \times n$ matrix $[A]$, vector $\{x\}$
output : $\{b\} = [A]\{x\}$

		flops	times
1	**for** $r \leftarrow 1$ **to** n **do**	0	$\sum_{r=1}^{n} 1 = n$
2	$b[r] \leftarrow 0$	0	n
3	**for** $c \leftarrow 1$ **to** n **do**	0	n^2
4	$b[r] \leftarrow b[r] + A[r,c] \cdot x[c]$	2	n^2
5	**end**		
6	**end**		

$$\sum_{r=1}^{n} \sum_{c=1}^{n} 1 = \sum_{r=1}^{n} n = n^2$$

FLOPS: $2n^2$

Gaussian elimination

input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output : solution $\{x\}$ of $[A]\{x\} = \{b\}$

```
   // forward elimination
1  for c ← 1 to n − 1 do
2  |   for r ← c + 1 to n do
3  |   |   f ← A[r, c] / A[c, c]
4  |   |   for j ← c to n do
5  |   |   |   A[r, j] ← A[r, j] − f · A[c, j]
6  |   |   end
7  |   |   b[r] ← b[r] − f · b[c]
8  |   end
9  end

   // back substitution
10 for r ← n, n − 1, . . . , 1 do
11 |   s ← b[r]
12 |   for j ← r + 1 to n do
13 |   |   s ← s − A[r, j] · x[j]
14 |   end
15 |   x[r] ← s / A[r, r]
16 end
```

	flops	times
	0	$\sum_{i=1}^{n-1} 1 = n-1$
→	1	$\frac{n^2-n}{2}$
	0	
→	2	$\frac{n^3}{3} + O(n^2)$
→	2	$\frac{n^2-n}{2}$
	0	n
	0	n
	0	$\frac{n^2-n}{2}$
→	2	
→	1	n

Useful sum identities

$$\sum_{i=1}^{m} c f(i) = c \sum_{i=1}^{m} f(i) \qquad \sum_{i=1}^{m} f(i) + g(i) = \sum_{i=1}^{m} f(i) + \sum_{i=1}^{m} g(i)$$

$$\sum_{i=1}^{m} 1 = m \qquad \sum_{i=k}^{m} 1 = m - k + 1$$

$$\sum_{i=1}^{m} i = \frac{m(m+1)}{2} \qquad \sum_{i=1}^{m} i^2 = \frac{m(m+1)(2m+1)}{6} = \frac{m^3}{3} + O(m^2)$$

$$\sum_{c=1}^{n-1} \sum_{r=c+1}^{n} 1 = \sum_{c=1}^{n-1} n - c = \sum_{c=1}^{n-1} n - \sum_{c=1}^{n-1} c$$

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

$$\sum_{c=1}^{n-1} \sum_{r=c+1}^{n} \sum_{j=c}^{n} 1 = \frac{n^3}{3} - \frac{n}{3}$$

see flop-count.pdf section 2.2

https://asen3502.com/lectures/020-algorithms/flop-count.pdf

FLOPS : $\frac{n^2-n}{2} + 2\left(\frac{n^3}{3} + O(n^2)\right) + 2\frac{n^2-n}{2}$

FLOPS $2\frac{n^3}{3} + O(n^2) + 2\frac{n^2-n}{2} + n$

```
function [C] = matmult([A], [B])
    for i = 1 to n
        for j = 1 to n
            c_ij = 0
            for k = 1 to n
                c_ij = c_ij + a_ik b_kj
            end
        end
    end
    return [C]
end
```

flops	times
0	n
0	n^2
0	⋮
0	n^3
2	n^3

Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote

Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote
- Chapra: $O(m^n)$ means "terms of order m^n and lower"

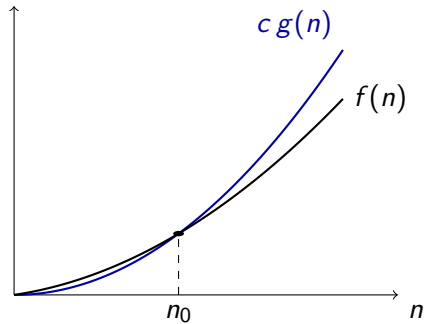
Asymptotic notation

- Asymptotic notation is a tool for keeping track of only the terms that are most important as a function approaches an asymptote
- Chapra: $O(m^n)$ means "terms of order m^n and lower"

Definition

$f(n) = O(g(n))$ as $n \to \infty$ if there exist positive constants c and n_0 such that

$$0 \leq f(n) \leq c\,g(n) \quad \text{for all } n \geq n_0$$



$f(n) \leq c\,g(n)$ for all $n \geq n_0$

← $f(n) = 3n^2 + 5n + 2$

$f(n) = 3n^2 + 5n + 2$

$c = 4.0$

$n_0 = 6$

then $0 \leq f(n) \leq c\,g(n)$

for all $n \geq n_0$

Therefore $f(n) = O(n^2)$

$c \geq$ sum of coefficients

$n_0 = 1$

$c = 10$

$n_0 = 1$

Useful Rules

1) $c\,f(n) = O(f(n)) \quad \forall\ c > 0$
2) if $f_1(n) = O(n^a)$ and $f_2(n) = O(n^b)$, $f_1(n) + f_2(n) = O(n^{\max(a,b)})$
3) if $f_1(n) = O(n^a)$ and $f_2(n) = O(n^b)$, $f_1(n) \cdot f_2(n) = O(n^{a+b})$
4) if $|f(n)|$ is bounded, $f(n) = O(1)$

Example: $f(n) = n^2 + 12n\sin(n) + 50 = O(n^2)$

n^2: $O(n^2)$; $12n\sin(n)$: $O(n)$ $O(1)$; 50: $50 \cdot 1$, $O(1)$

Gaussian elimination

input : $n \times n$ matrix $[A]$, right-hand side $\{b\}$
output : solution $\{x\}$ of $[A]\{x\} = \{b\}$

```
// forward elimination
for c ← 1 to n − 1 do                                  ≤ n
    for r ← c + 1 to n do                              ≤ n
        f ← A[r, c] / A[c, c]
        for j ← c to n do                              ≤ n
            A[r, j] ← A[r, j] − f · A[c, j]
        end
        b[r] ← b[r] − f · b[c]
    end
end
// back substitution
for r ← n, n − 1, . . . , 1 do
    s ← b[r]
    for j ← r + 1 to n do
        s ← s − A[r, j] · x[j]
    end
    x[r] ← s / A[r, r]
end
```

flops	Order of times called
×	$O(n)$
×	$O(n^2)$
✓	
×	$O(n^3)$
✓	
×	$O(n^2)$
✓	
×	$O(n)$
×	
×	$O(n^2)$
✓	
✓	$O(n)$

$O(n^3)$ Flops

Today

LU - Factorization

HW1 Due tomorrow
HW2 Assigned tonight
Lab 1: catch up with team to
1.3 (minimum)
1.5 (recommended)
before Friday

Study Hall tonight!

Recall: with a triangular matrix, T, solving $Tx = b$ for x is $O(n^2)$

Suppose we have

$$LU = A$$

L: lower triangular, U: upper triangular

$$Ax = b$$
$$L\underbrace{Ux}_{d} = b$$

0. factorize A into LU
1. solve $Ld = b$
2. solve $Ux = d$

both steps solve with a triangular matrix

$$\left(\underbrace{I}_{\begin{bmatrix}1&0&0\\0&1&0\\0&0&1\end{bmatrix}} + \begin{bmatrix}0&0&0\\2&0&0\\0&0&0\end{bmatrix}\right)\begin{bmatrix}2&3&4\\0&5&6\\2&8&17\end{bmatrix} \overset{\checkmark}{=} \begin{bmatrix}2&3&4\\4&11&14\\2&8&17\end{bmatrix}$$

$\Rightarrow L$ $\Rightarrow U$

$$\left(I + \begin{bmatrix}0&0&0\\2&0&0\\1&0&0\end{bmatrix}\right)\begin{bmatrix}2&3&4\\0&5&6\\0&5&13\end{bmatrix} \overset{\checkmark}{=} \quad 11$$

$$\left(I + \begin{bmatrix}0&0&0\\2&0&\\1&1&0\end{bmatrix}\right)\begin{bmatrix}2&3&4\\0&5&6\\0&0&7\end{bmatrix} \overset{\checkmark}{=} \quad 11$$

$$\underbrace{\begin{bmatrix}1&0&0\\2&1&0\\1&1&1\end{bmatrix}}_{L}\underbrace{\begin{bmatrix}2&3&4\\0&5&6\\0&0&7\end{bmatrix}}_{U} = \begin{bmatrix}2&3&4\\4&11&14\\2&8&17\end{bmatrix}$$

$$L = \begin{bmatrix}1&&&\\f_{21}&1&&\\f_{31}&f_{32}&\ddots&\\\vdots&&&\\f_{n1}&&f_{n,n-1}&1\end{bmatrix}$$

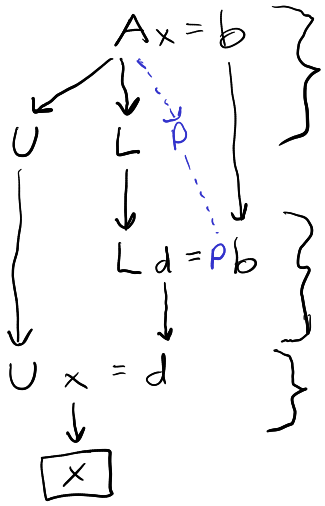
U = result of GE

f_{ij} = factor from GE for eliminating element ij

LU factorization: run GE and save factors in L

$$A = \begin{bmatrix}2&1\\6&5\end{bmatrix}\begin{matrix}①\\②\end{matrix}$$ subtract $3 \times ①$ from ②

$$L = \begin{bmatrix}1&0\\3&1\end{bmatrix} \quad U = \begin{bmatrix}2&1\\0&2\end{bmatrix} \quad LU = \begin{bmatrix}2&1\\6&5\end{bmatrix} \overset{\checkmark}{=} A$$



Factorization $O(n^3)$ only involves A not b!

(Forward) Substitution $O(n^2)$

(Back) Substitution $O(n^2)$

If we have many b's to solve for, factorize once $(O(n^3))$ then do forward and back substitution for each b $(O(n^2))$!

What about Pivoting?

$LU = PA$

↑ permutation matrix (keep track of pivots)

$$\begin{bmatrix} 4 & 11 & 14 \\ 2 & 3 & 4 \\ 2 & 8 & 17 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 3 & 4 \\ 4 & 11 & 14 \\ 2 & 8 & 17 \end{bmatrix}$$

A' P A

each row i determines which row j of A ends up in A'

$p_{ij} = 1$ if row j of A becomes row i of A'

Exercise 2

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ c_1 & c_2 & c_3 \\ b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$$\begin{bmatrix} -c- \\ -b- \\ -a- \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \\ c_1 & c_2 & c_3 \end{bmatrix}$$

$Ax = b$

$LU = PA$

$PAx = Pb$

$LUx = Pb$

Matrix Inverse
Condition Number
Roundoff Error

Homework 2 Due Thursday

$$Ax = b$$

inverse → $A^{-1}b = x$

$$AA^{-1}b = Ax = b$$

$$AA^{-1}b = b$$

$$AA^{-1} = I \quad (3\times 3)$$

$$A\begin{bmatrix} a_{11}^{-1} \\ a_{21}^{-1} \\ a_{31}^{-1} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \qquad A\begin{bmatrix} a_{12}^{-1} \\ a_{22}^{-1} \\ a_{32}^{-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \qquad A\begin{bmatrix} a_{13}^{-1} \\ a_{23}^{-1} \\ a_{33}^{-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Matrix Inverse via LU factorization

input : $n \times n$ matrix $[A]$
output: $[A]^{-1}$

$[L], [U] \leftarrow$ `lu_decompose`$([A])$
for $k \leftarrow 1$ **to** n **do**
  $\{e_k\} \leftarrow$ unit vector with 1 at element k; 0 elsewhere
  $A^{-1}[:, k] \leftarrow$ `lu_solve`$([L], [U], \{e_k\})$
end

What can go wrong? from APPM 2360

$Ax = b$ does not have a unique solution $\iff$ A is singular $\iff$ $\det(A) = 0$ $\iff$ A is not full rank $\iff$ A^{-1} does not exist

$\begin{bmatrix} 2 & 4 \\ 1 & 2 \end{bmatrix}$ not full rank

What if A is almost singular?

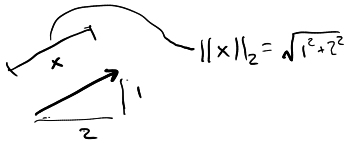
$$\text{cond}(A) = \|A\| \cdot \|A^{-1}\|$$

large cond $\Rightarrow$ nearly singular

$\| \ \|$ means "norm" Book covers many types of norms

For now $\| \ \| = \| \ \|_2$ "2-norm"

For a vector x, $\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \dots x_n^2}$, "length" of x



For a matrix A, $\|A\|_2 = \max_{x \neq 0} \frac{\|Ax\|_2}{\|x\|_2} = \sqrt{\mu_{max}}$ ← max eigenvalue

"The most A can elongate a vector"

Key property of cond:

$$\frac{\|\Delta x\|}{\|x\|} \leq \text{cond}(A) \frac{\|\Delta A\|}{\|A\|}$$

relative error in result

relative error in matrix

Roundoff Errors

Computer uses only 3 significant figures

747 takeoff mass

3.97×10^5 kg burns 4.21 kg/s fuel

$$\begin{array}{rl} 397{,}000 & \text{kg} \\ - \quad 4.21 & \text{kg} \\ \hline 396{,}995.71 & \end{array} \longrightarrow \text{3 sig figs} \longrightarrow 3.97 \times 10^5 \text{ kg}$$

How computers store numbers: bits (1 or 0)

Humans think in decimal digits 0, ..., 9

Computers think in Binary 0, 1

Unsigned Integers

	1000	100	10	1
3	0	0	0	3
137	0	1	3	7
43	0	0	4	3

1 byte

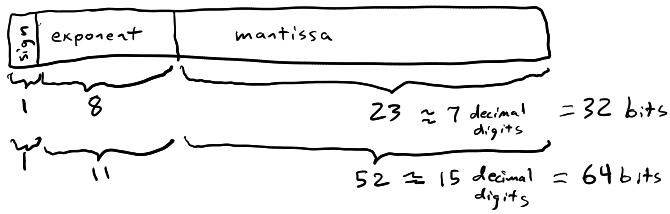
128	64	32	16	8	4	2	1
0	0	0	0	0	0	1	1
1	0	0	0	1	0	0	1
0	0	1	0	1	0	1	1

Scientific Notation

$$\pm s \times 10^{e}$$ (e: exponent; s: significand)

Floating Point IEEE 754

$$\pm (1+f) \times 2^{(e-127)}$$ (e: exponent; f: mantissa)



precision
single
double

matlab/julia eps(x) distance to next floating point number above x

$$\frac{\|\Delta A\|}{\|A\|} \approx \text{eps}(1.0)$$

$$\frac{\|\Delta x\|}{\|x\|} \lesssim \text{cond}(A)\,\varepsilon$$

Please Grab a handout in the back.

Today

Solving Nonlinear Function

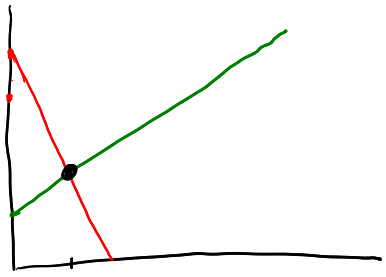
Bracketing Methods: Most important: Bisection

Announcements / Reminders

Lab 1 due Thursday - Check Auto-grader!

HW 1 Released (Graded)

Linear

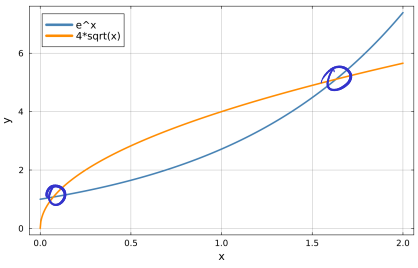


$-2x + 4 = x + 1$

$x_2 = -2x_1 + 4$ $\quad x_2 = x_1 + 1$

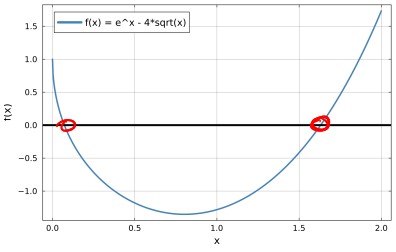
$x = 1$

Nonlinear



$e^x = 4\sqrt{x}$

no analytical in "simple functions"



$f(x) = 0 = e^x - 4\sqrt{x}$

find "roots"

x-values where $f(x) = 0$

Bracketing Open

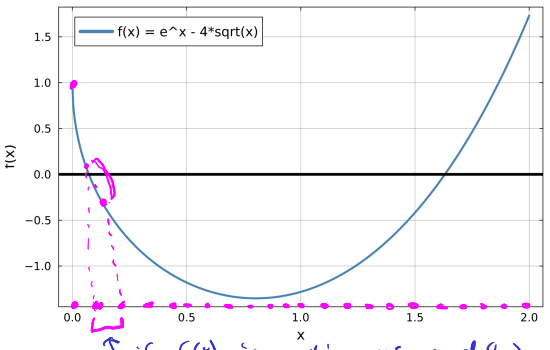
Today

Incremental Search

Incremental Search

input : function f, search range $[x_{\min}, x_{\max}]$, number of steps n
output: list of brackets $[x_l, x_u]$, each containing a root

$\Delta x \leftarrow (x_{\max} - x_{\min}) / n$
brackets $\leftarrow \{\}$
$x_l \leftarrow x_{\min}$
for $k \leftarrow 1$ **to** n **do**
  $x_u \leftarrow x_l + \Delta x$
  if $f(x_l) \cdot f(x_u) < 0$ **then**
    // sign change ⇒ a root lies in $[x_l, x_u]$
    append $[x_l, x_u]$ to brackets
  end
  $x_l \leftarrow x_u$
end
return brackets



if $f(x)$ is continuous and $f(x_l)$ and $f(x_r)$ have different signs, a root is between x_l and x_r

Strength: Can find multiple roots

Weakness: Can miss roots, slow at finding one root (we will see shortly)

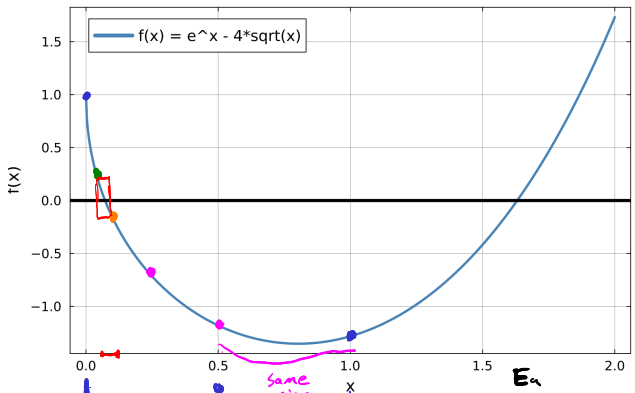
error → $E_a = \frac{\Delta x}{n-1}$

Bisection

Bisection

input : function f, bracket $[x_l, x_u]$ with $f(x_l) \cdot f(x_u) < 0$, tolerance ε_s
output: estimate x_r of the root

$x_r \leftarrow x_l$
repeat
  $x_{r,\text{old}} \leftarrow x_r$
  $x_r \leftarrow (x_l + x_u) / 2$
  // keep the half that still brackets the root
  if $f(x_l) \cdot f(x_r) < 0$ **then**
    $x_u \leftarrow x_r$
  else if $f(x_l) \cdot f(x_r) > 0$ **then**
    $x_l \leftarrow x_r$
  else
    return x_r // exact root
  end
  $\varepsilon_a \leftarrow \left| \frac{x_r - x_{r,\text{old}}}{x_r} \right| \cdot 100\%$
until $\varepsilon_a < \varepsilon_s$
return x_r



Iter:	1		2		3		4		5
	1		1		1		1		H
		•		•		•	1		↑↑↑
		1		1		1			0.0675 0.125

E_a

$\Delta x = 1$

$\frac{\Delta x}{2} = 0.5$

$\frac{\Delta x}{4} = 0.25$

$\frac{\Delta x}{8} = 0.125$

$\frac{\Delta x}{16} = 0.0675$

$E_a = \frac{\Delta x}{2^n}$

$$E_a = \frac{\Delta x}{2^n}$$

function calls for desired $\frac{\Delta x}{E_s}$

Incremental

$E_a = E_s = \frac{\Delta x}{n-1}$

$n - 1 = \frac{\Delta x}{E_s}$

$$n = \frac{\Delta x}{E_s} + 1$$

$O\left(\frac{\Delta x}{E_s}\right)$

Bisection

$E_a = E_s = \frac{\Delta x}{2^n}$

$2^n = \frac{\Delta x}{E_s}$

$$n = \log_2\left(\frac{\Delta x}{E_s}\right)$$

$O\left(\log\left(\frac{\Delta x}{E_s}\right)\right)$

$E_s = 0.01\,\Delta x$

$\frac{\Delta x}{E_s} = 100$

101 func evals

$\log(100) \approx 6.64$

7 function evals !

$E_s = 0.0001\,\Delta x$

$\frac{\Delta x}{E_s} = 10{,}000$

10,001 func evals

14 func evals

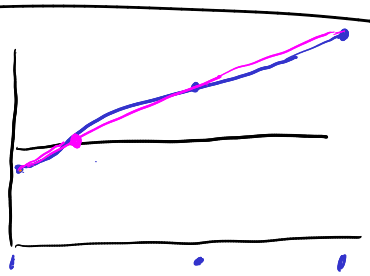
False Position

False Position

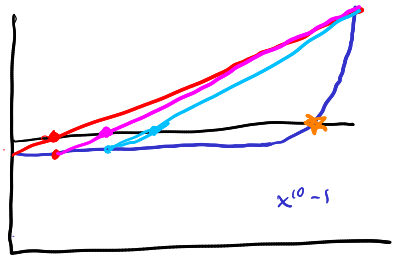
input : function f, bracket $[x_l, x_u]$ with $f(x_l) \cdot f(x_u) < 0$, tolerance ε_s
output: estimate x_r of the root

$x_r \leftarrow x_l$
repeat
  $x_{r,\text{old}} \leftarrow x_r$
  $x_r \leftarrow x_u - \frac{f(x_u)(x_l - x_u)}{f(x_l) - f(x_u)}$
  // keep the side that still brackets the root
  if $f(x_l) \cdot f(x_r) < 0$ **then**
    $x_u \leftarrow x_r$
  else if $f(x_l) \cdot f(x_r) > 0$ **then**
    $x_l \leftarrow x_r$
  else
    return x_r // exact root
  end
  $\varepsilon_a \leftarrow \left|\frac{x_r - x_{r,\text{old}}}{x_r}\right| \cdot 100\%$
until $\varepsilon_a < \varepsilon_s$
return x_r

← only difference from bisection



Problem: Doesn't handle curvature well



Today

Open Methods: Newton-Raphson + Friends

Secant, Brent's, Fixed Point Iter (maybe)

Reminders

Lab 1 due tomorrow

First: Errors (Chapra's notation)

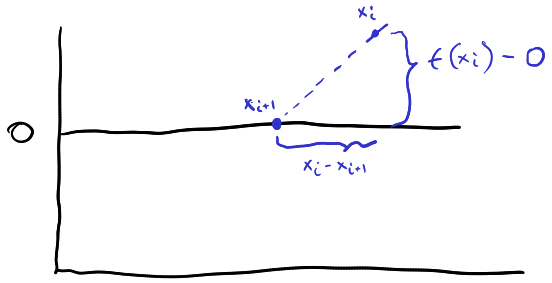
E: absolute error (units)

ε: relative (percent) error

Subscripts

- t: true (typically unknown)
- a: approximate
- s: stopping

Newton-Raphson: use $f'(x)$



$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}} \quad \frac{\text{rise}}{\text{run}}$$

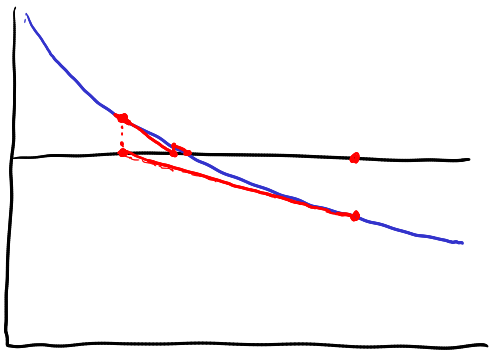
$$(x_i - x_{i+1}) f'(x_i) = f(x_i)$$

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

Newton-Raphson Most widely used

input : function f and its derivative f', initial guess x_0, tolerance ε_s, maximum iterations N
output: estimate x_r of the root

$x_r \leftarrow x_0$
for $i \leftarrow 1$ **to** N **do**
  $x_{r,\text{old}} \leftarrow x_r$
  $x_r \leftarrow x_{r,\text{old}} - \frac{f(x_{r,\text{old}})}{f'(x_{r,\text{old}})}$ ← update
  $\varepsilon_a \leftarrow \left|\frac{x_r - x_{r,\text{old}}}{x_r}\right| \cdot 100\%$
  if $\varepsilon_a < \varepsilon_s$ **then**
    return x_r
  end
end
return x_r // did not converge in N iterations



Beauty of Newton-Raphson: Extremely fast convergence near root!

Bisection

$E_{a,i+1} = \frac{1}{2} E_{a,i}$

"linear convergence"

number of sig figs increases linearly with i

(different terminology from $E_{a,i} = O\left(\left(\frac{1}{2}\right)^k\right)$)

Gauranteed to converge if
- Δx contains root
- $f(x)$ is continuous

fast

Newton Raphson

Near Root ↓ "can be shown"

→ $E_{t,i+1} \simeq -\frac{f''(x_r)}{2f'(x_r)} E_{t,i}^2$

"quadratic convergence"

number of sigfigs doubles every iteration

Can diverge

FAST near root

Downside: Newton requires $f'(x)$

Secant Method

$$f'(x_i) \approx \frac{f(x_{i-1}) - f(x_i)}{x_{i-1} - x_i}$$

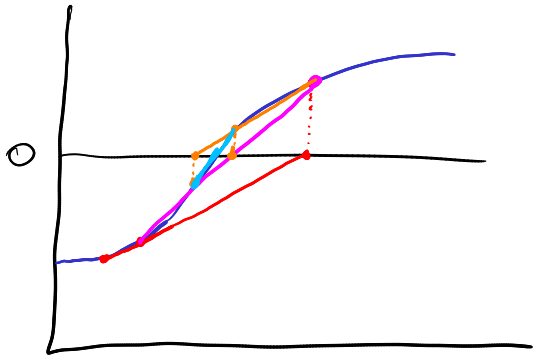
plug into Newton update

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

Secant Method

input : function f, initial guesses x_{-1} and x_0, tolerance ε_s, maximum iterations N
output: estimate x_r of the root

$x_{\text{prev}} \leftarrow x_{-1}$; $x_r \leftarrow x_0$
for $i \leftarrow 1$ **to** N **do**
  $x_{r,\text{old}} \leftarrow x_r$
  $x_r \leftarrow x_{r,\text{old}} - \dfrac{f(x_{r,\text{old}})\,(x_{\text{prev}} - x_{r,\text{old}})}{f(x_{\text{prev}}) - f(x_{r,\text{old}})}$ ← only difference from Newton
  $x_{\text{prev}} \leftarrow x_{r,\text{old}}$
  $\varepsilon_a \leftarrow \left|\dfrac{x_r - x_{r,\text{old}}}{x_r}\right| \cdot 100\%$
  if $\varepsilon_a < \varepsilon_s$ **then**
    return x_r
  end
end
return x_r // did not converge in N iterations



Brent's Method

- Combination of Bracketing and Open Methods
- Does not require derivatives
- Basis for Matlab's fzero

Brent's Method

input : function f, bracket $[a, b]$ with $f(a) \cdot f(b) < 0$, tolerance ε_s
output: estimate b of the root

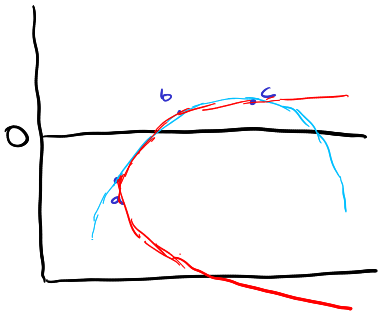
```
// b is the best estimate, c is on the other side of the root, a
   is the previous b
c ← a
// d is the signed step b will move by, e is the previous step;
   both start as the bracket width
d ← b − a;   e ← d
repeat
    if f(b) · f(c) > 0 then
        // b and c no longer bracket the root; a does
        c ← a;   d ← b − a;   e ← d
    end
    if |f(c)| < |f(b)| then
        // make b the point with the smallest |f|
        a ← b;   b ← c;   c ← a
    end
    m ← (c − b) / 2                                  // bisection step
    δ ← ε_s |b|                                      // absolute tolerance
    if |m| ≤ δ or f(b) = 0 then
        return b
    end
    if |e| < δ or |f(a)| ≤ |f(b)| then
        d ← m;   e ← m                               // bisect
    else
        s ← f(b) / f(a)
        if a = c then
            // only two distinct points:  secant step
            p ← 2ms;   q ← 1 − s
        else
            // three distinct points:  inverse quadratic
               interpolation
            q ← f(a) / f(c);   r ← f(b) / f(c)
            p ← s (2mq (q − r) − (b − a)(r − 1))
            q ← (q − 1)(r − 1)(s − 1)
        end
        if p > 0 then q ← −q
        else p ← −p
        if 2p < 3mq − |δq| and p < |eq/2| then
            // interpolated step stays in the bracket and shrinks
               it fast enough
            e ← d;   d ← p / q
        else
            d ← m;   e ← m                           // bisect
        end
    end
    a ← b
```

← bisection

← secant

← inverse quadratic

- Gauranteed to converge
- As fast near root as open methods



Fixed Point

Previously $f(x) = 0$

Fixed Point $\boxed{x = g(x)}$

repeatedly evaluate $g(x)$

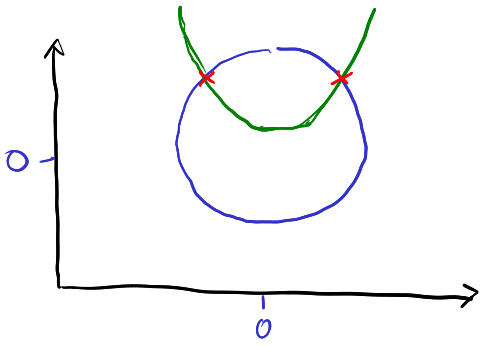
Today

Systems of nonlinear equations
Newton-Raphson for systems
Numerical differentiation

Announcements

- HW 3 due Thurs
- Exam Next Monday
  - Will release Eqn. Sheet / practice exam today / tomorrow
  - Calculator
  - Study HW, Labs, Notes

Up to now $f(x) = 0$



$x_1^2 + x_2^2 = 4$

$x_2 = x_1^2$

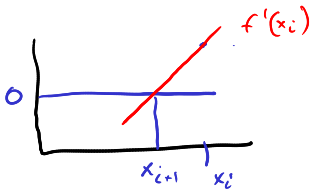
$\{f\}(\{x\}) = \{0\}$

$\vec{f}(\vec{x}) = \vec{0}$

$$\vec{f}(\vec{x}) = \begin{bmatrix} x_1^2 + x_2^2 - 4 \\ x_2 - x_1^2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Newton-Raphson for scalar:

$$f(x_i) = f(x_{i+1}) + f'(x_i)(x_i - x_{i+1})$$



vector case:

choose $\vec{x}_{i+1}$ so this is zero

first element of f →

$$f_1(\vec{x}_i) = f_1(\vec{x}_{i+1}) + \frac{\partial f_1}{\partial x_1}(x_{1,i} - x_{1,i+1}) + \frac{\partial f_1}{\partial x_2}(x_{2,i} - x_{2,i+1}) + \dots \frac{\partial f_1}{\partial x_n}(x_{n,i} - x_{n,i+1})$$

$$= \sum_{j=1}^{n} \frac{\partial f_1}{\partial x_j}(x_{j,i} - x_{j,i+1})$$

$$f_2(\vec{x}_i) = \sum_{j=1}^{n} \frac{\partial f_2}{\partial x_j}(x_{j,i} - x_{j,i+1})$$

$$\vdots$$

$$f_n(\vec{x}_i) = \sum_{j=1}^{n} \frac{\partial f_n}{\partial x_j}(x_{j,i} - x_{j,i+1})$$

$$\vec{f}(\vec{x}_i) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \\ \frac{\partial f_n}{\partial x_1} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \left(\begin{bmatrix} x_{1,i} \\ \vdots \\ x_{n,i} \end{bmatrix} - \begin{bmatrix} x_{1,i+1} \\ \vdots \\ x_{n,i+1} \end{bmatrix} \right)$$

$$\vec{f}(\vec{x}_i) = J \left(\vec{x}_i - \vec{x}_{i+1}\right)$$

$\vec{f}(\vec{x}_i)$: known; J: known; $\vec{x}_i$: known; $\vec{x}_{i+1}$: unknown

$$J^{-1}\vec{f}(\vec{x}_i) = \vec{x}_i - \vec{x}_{i+1}$$

$$\boxed{\vec{x}_{i+1} = \vec{x}_i - J^{-1}\vec{f}(\vec{x}_i)}$$

(scalar)

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

want $\vec{x}_{i+1} = \vec{x}_i + \Delta\vec{x}$

$\Delta\vec{x} = \vec{x}_{i+1} - \vec{x}_i$

solve $J\Delta\vec{x} = -\vec{f}(\vec{x}_i)$

$\vec{x}_{i+1} \leftarrow \vec{x}_i + \Delta\vec{x}$

or

$\vec{x}_{i+1} \leftarrow \vec{x}_i - J \backslash \vec{f}(\vec{x}_i)$

$A \backslash b \equiv$ solution to $Ax = b$

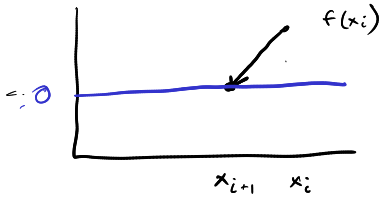
Newton-Raphson for Systems

input : function $\{f\} : \mathbb{R}^n \to \mathbb{R}^n$ and its Jacobian $[J]$ (with $J_{ij} = \partial f_i / \partial x_j$), initial guess $\{x\}_0$, tolerance ε_s, maximum iterations N

output: estimate $\{x\}_r$ of the root

$\{x\}_r \leftarrow \{x\}_0$
for $i \leftarrow 1$ **to** N **do**
  $\{x\}_{r,\text{old}} \leftarrow \{x\}_r$
  solve $[J](\{x\}_{r,\text{old}})\{\Delta x\} = -\{f(\{x\}_{r,\text{old}})\}$
  $\{x\}_r \leftarrow \{x\}_{r,\text{old}} + \{\Delta x\}$
  $\varepsilon_a \leftarrow \dfrac{\|\{x\}_r - \{x\}_{r,\text{old}}\|}{\|\{x\}_r\|} \cdot 100\%$
  if $\varepsilon_a < \varepsilon_s$ **then**
    return $\{x\}_r$
  end
end
return $\{x\}_r$ // did not converge in N iterations

(scalar)



Two Challenges

1. Convergence
2. Calculating Jacobian

Strength

"quadratic convergence" near root

Optimization

minimize $\sum_{k=1}^{n} \left(f_k(\vec{x})\right)^2$

Calculating Derivatives

Symbolic

- By hand
- Computer Algebra System (CAS)
  - e.g. Symbolic Toolbox
- Automatic Code gen

Numerical

Automatic

- Tracks exact derivative through each operation

Numerical Differentiation

Forward: $f'(x) \approx \frac{f(x+h) - f(x)}{h}$

Central: $f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$ ← more accurate

Column of Jacobian → $\frac{\partial \vec{f}}{\partial x_j} \approx \frac{\vec{f}(\vec{x} + h\vec{e}_j) - \vec{f}(\vec{x})}{h}$

size of h ?

$h \approx \sqrt{\varepsilon} \times$

↑ machine precision eps

Numerical Jacobian (forward difference)

input : function $\{f\} : \mathbb{R}^n \to \mathbb{R}^n$, point $\{x\}$, step size h
output: approximation $[J]$ of the Jacobian at $\{x\}$, $J_{ij} \approx \partial f_i / \partial x_j$

$\{f\}_0 \leftarrow \{f(\{x\})\}$
for $j \leftarrow 1$ **to** n **do**
  $\{e_j\} \leftarrow$ unit vector in the j direction // jth column of $[I]$
  $[J][:, j] \leftarrow \frac{\{f(\{x\} + h\{e_j\})\} - \{f\}_0}{h}$ // jth column of $[J]$
end
return $[J]$