

9.1

Outline

1) Nonlinear equations



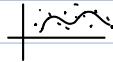
2) Machine Arithmetic (floating point)

3) Linear Systems ($A\vec{x} = \vec{b}$)

4) Interpolation



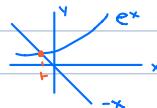
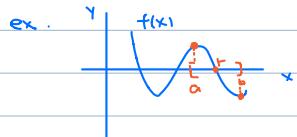
5) Approximation



(least squares)

6) Differentiation / Integration

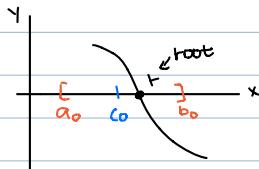
7) ODEs

Root Finding : Goal to find $r \in \mathbb{R}$ s.t. $f(r) = 0$ ex. solve $e^x = -x$ Rewrite $g(x) := e^x + x = 0$ 

Intermediate Value Thm (IVT)

Let $f \in C[a,b]$ (i.e. f is continuous on $[a,b]$)If $f(a)f(b) < 0$, then $\exists r \in (a,b)$ s.t. $f(r) = 0$.

Bisection Method

Know $r \in (a_0, b_0)$ by IVTEstimate $r \approx c_0 = (a_0 + b_0)/2$ Error: $e_0 = |r - c_0| \leq (b_0 - a_0)/2$ ① If $f(c_0) = 0$, then $r = c_0$.② If $f(a_0)f(c_0) > 0$, then $r \in (c_0, b_0)$ set $[a_1, b_1] = [c_0, b_0]$ ③ If $f(a_0)f(c_0) < 0$, then $r \in (a_0, c_0)$ by IVT and set $[a_1, b_1] = [a_0, c_0]$ choose midpoint $c_1 = (a_1 + b_1)/2$ and repeat this entire process

Error Estimate:

$$e_0 = |r - c_0| \leq (b_0 - a_0)/2$$

$$e_1 = |r - c_1| \leq (b_1 - a_1)/2 \leq (b_0 - a_0)/2^2$$

⋮

$$e_n = |r - c_n| \leq \dots \leq (b_0 - a_0)/2^{n+1}$$

$$\text{so } e_n \leq (b_0 - a_0)/2^{n+1} \text{ and } r \approx c_n = (a_n + b_n)/2$$

Remarks :

① With $e_n \leq (b_0 - a_0)/2^{n+1} < \text{Tol}$ tolerance (10^{-6})can solve for n to ensure tolerance is met.

$$n \geq \frac{\ln(\frac{b_0 - a_0}{\text{Tol}})}{\ln(2)} - 1$$

- (2) Method always converges if you have a starting bracket.
 (3) $f(x)$ plays no role in computing n for Tol satisfaction (for error)



(4) double root, no bracket possible.

method can fail!

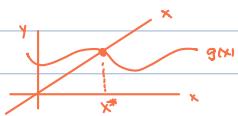
- (5) Convergence is slow:

Have $e_{n+1} \approx \frac{1}{2} e_n$ (linear convergence)

Fixed Point Iteration (FPI)

Consider solving $\cos(x) = 0$, can use bisection...

OR. $g(x) := \cos(x) + x = x$. can instead solve $g(x) = x$. A point x^* s.t. $g(x^*) = x^*$ is called a fixed point of g .



9/3

Iterative Method: Try to solve $g(x) = x$ using

$$\begin{cases} x_{i+1} = g(x_i) \\ x_0 = \text{starting value} \end{cases}$$

ex: 1) $\begin{cases} x_{i+1} = x_i (\frac{1}{10}) + 1 \\ x_0 = 0 \end{cases}$

[$g(x) = 1 + \frac{x}{10}$]

$x_1 = 1, x_2 = 1.1, x_3 = 1.11, \dots, x^* = 1.111\dots = \frac{10}{9}$

Have $\frac{10}{9} = g(\frac{10}{9})$

2) $\begin{cases} x_{i+1} = 3x_i + 1 \\ x_0 = 0 \end{cases}$

$x_1 = 1, x_2 = 4, x_3 = 13, x_4 = 40, \dots$ FPI blows up!

3) $\begin{cases} \vec{x}_{i+1} = A\vec{x}_i \\ \vec{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \in \mathbb{R}^2 \end{cases}$

4) $\begin{cases} y' = f(t, y) \\ y(0) = 3 \end{cases} \Leftrightarrow \begin{cases} y_{i+1}(t) = 3 + \int_0^t f(s, y(s)) ds \\ y_0(t) = 0 \end{cases}$

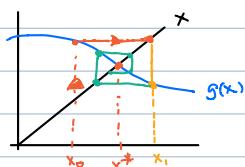
5) $\begin{cases} x_{i+1} = x_i + \cos x_i \\ x_0 = 0 \end{cases}$

$x_1 = 1, x_2 = 1.5403\dots$

$x_3 = 1.57079\dots$ correct up to 5 decimal places

Geometric Interpretation.

$x_{i+1} = g(x_i)$



cobweb diagrams

Convergence of FPI

existence of fixed points in Ley's note.

contraction mappings.

let $g \in [a, b]$, g is a contraction on $[a, b]$

$$\text{if } \exists L, 0 \leq L < 1, \text{ s.t. } |g(x) - g(y)| \leq L|x-y| \quad \forall x, y \in [a, b]$$

contraction Mapping THM

let $g : [a, b] \rightarrow [a, b]$ be a contraction.

then 1) \exists unique fixed point x^* of g . $g(x^*) = x^*$

2) FPI converges to x^* starting from any $x_0 \in [a, b]$

$$x_{i+1} = g(x_i)$$

Remarks :

1) If $g \in C^1(a, b)$, g is a contraction provided $|g'(x)| \leq L \leq 1$ for $\forall x \in (a, b)$

$$|g(x) - g(y)| = |g'(y)(x-y)| \leq |g'(y)||x-y| \leq L|x-y|$$

by mean-value theorem

2) If fixed point x^* is known, can check $|g'(x^*)| < 1$ for g to be a contraction on a small interval around x^*
(have contraction locally around x^*)

3) To stop iteration, need stopping criteria.

(i) Run FPI $N < \infty$ times.

(ii) Or stop when $|x_{i+1} - x_i| < TOL$
absolute error

ex: $\cos x = 0$

$$g(x) = \cos x + x = x$$

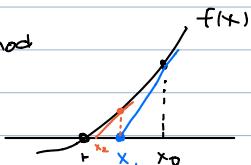
Have $g'(x) = -\sin x + 1$, At least on $[\frac{\pi}{4}, \frac{3\pi}{4}]$ have $|g'(x)| \leq L < 1$

For $x_0 \in [\frac{\pi}{4}, \frac{3\pi}{4}]$, $x_{i+1} = g(x_i)$ will converge.

Can also note $x^* = \frac{\pi}{2}$ is a root / fixed point

and $|g'(\frac{\pi}{2})| = 0 < 1$ so FPI converges locally near $x^* = \frac{\pi}{2}$

Newton's Method



$$\text{Taylor Expansion: } f(x) = f(x_0) + f'(x_0)(x-x_0) + \underbrace{\frac{1}{2}f''(x_0)(x-x_0)^2}_{L(x) \text{ (Linear)}} + O(x-x_0)^3$$

$$f(x) \approx L(x) = f(x_0) + f'(x_0)(x-x_0)$$

$$\Rightarrow 0 = L(x_1) = f(x_0) + f'(x_0)(x_1-x_0)$$

$$\Rightarrow x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Method:

$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$
$x_0 = \text{starting guess}$

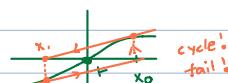
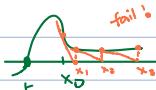
Remark:

Can approximate $f(x) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$
(secant method)

Remarks: 1) Newton is an FPI, with $g(x) = x - \frac{f(x)}{f'(x)}$

$$g'(r) = 1 - \frac{f(r)f''(r) - f'(r)f'''(r)}{(f'(r))^2} = 0 < 1$$

2) Newton can fail!



9.18 Error Analysis + Convergence

$f(r) = 0$. want root r

$$\text{let } e_n = |x_n - r|$$

$$\text{consider Newton's method : } x_{n+1} = \underbrace{x_n - \frac{f(x_n)}{f'(x_n)}}_{=: g(x_n)}$$

$$\text{Remark . } f(x) = f(x_0) + (x-x_0)f'(x_0) + \frac{1}{2}(x-x_0)^2 f''(x_0) + \frac{1}{6}(x-x_0)^3 f'''(\xi) \quad \begin{matrix} \text{b/w } x, x_0 \\ \xi \end{matrix}$$

Taylor expand :

$$f(r) = f(x_n) + (r-x_n)f'(x_n) + \frac{1}{2}(r-x_n)^2 f''(\xi_n)$$

root, so $f(r) = 0$

\hookrightarrow is between x_n and r

$$-f(x_n)/f'(x_n) = (r-x_n) + \frac{1}{2}(r-x_n)^2 f''(\xi_n)/f'(x_n)$$

\uparrow

$$x_n - \frac{f(x_n)}{f'(x_n)} = r + \frac{1}{2}(r-x_n)^2 \frac{f''(\xi_n)}{f'(x_n)}$$

$$= x_{n+1}$$

$$x_{n+1} - r = \frac{1}{2}(r-x_n)^2 f''(\xi_n)/f'(x_n)$$

$$\text{OR } e_{n+1} = \frac{1}{2} e_n^2 |f''(\xi_n)/f'(x_n)|$$

$$e_n = |x_n - r|$$

this roughly says $e_{n+1} \propto e_n^2$ (quadratic convergence)

$$\lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n^2} = \lim_{n \rightarrow \infty} \frac{1}{2} \left| \frac{f''(\xi_n)}{f'(x_n)} \right| = \frac{1}{2} \left| \frac{f''(r)}{f'(r)} \right|$$

\hookrightarrow this provided 1) have convergence

2) $f \in C^2$ near r

3) $f' \neq 0$

Definition:

Let $e_n = |x_n - r|$ be the error of an iterative method.

$$\text{If } \lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n^p} = s < \infty \quad (p \geq 1)$$

we say the convergence is of order p .

(can think $e_{n+1} \propto S e_n^p$)

- $p=2$ (quadratic convergence)

Ex: Newton's method (provided $f' \neq 0$ near the root)

- $p=1$ (linear convergence)

Ex: Bisection method is linearly convergent $e_{n+1} \leq \frac{1}{2} e_n^1$

note: we need $s < 1$ for linear convergence.

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

otherwise may get blow up

Notions of Error and Accuracy.

Want r s.t. $f(r) = 0$

r = real root

r_{approx} = approximate root

Have $r \approx r_{\text{approx}}$ and $f(r_{\text{approx}}) = \epsilon \ll 1$

Definition:

forward error := $|r - r_{\text{approx}}|$

need to know r beforehand.

backward error := $|\epsilon| = |f(r_{\text{approx}})|$

Motivation : data $[f(x)] \rightarrow$ Eqn solver \rightarrow solution $[r]$

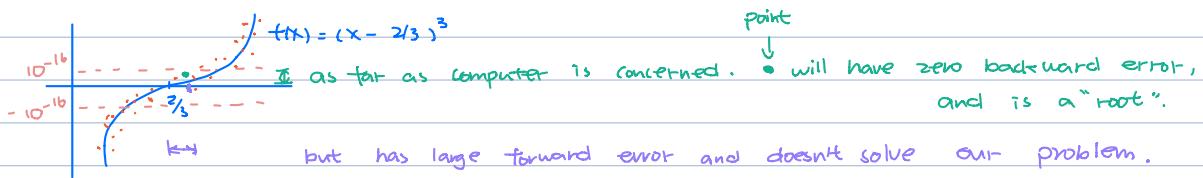
"backward error"

"forward error"

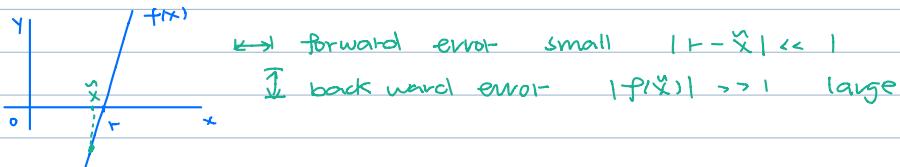
$$\text{Ex. } f(x) = (x - \frac{2}{3})^3 = x^3 - 2x^2 + \frac{4}{3}x - \frac{8}{27} = 0$$

$$\text{Actual root} = r = \frac{2}{3}$$

Lots of machine (floating point) numbers nearby



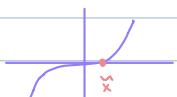
Ex:



$$\text{Ex. } f(0) = 0, f(x) = x^{15}. \text{ Suppose } r=0, r_{\text{approx}} = 10^{-1}$$

$$\text{bkwd. error} = (10^{-1})^{15} = 10^{-15} \ll 1$$

$$\text{fwd. error} = |0 - 10^{-1}| = 0.1 \text{ which is fairly large}$$



Convergence of FPI : $x_{n+1} = g(x_n)$

Have local convergence by contraction mapping if $|g'(r)| \leq L < 1$

By Mean-value theorem, $\exists \bar{x}_n$ between x_n and r .

$$\begin{aligned} \text{Let } \frac{g(x_n) - g(r)}{x_n - r} &= g'(\bar{x}_n)(x_n - r) \\ &= x_{n+1} - r \\ \Rightarrow \frac{x_{n+1} - r}{x_n - r} &= g'(\bar{x}_n) \Rightarrow \lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n} = |g'(r)| \end{aligned}$$

So FPI is (at least) linearly convergent, provided $|g'(r)| < 1$

Remark : If $g'(r) = 0$ (this is the case for Newton's method)

$$\text{then } g(x_n) - g(r) = g'(r)(x_n - r) + \frac{1}{2}g''(\bar{x}_n)(x_n - r)^2$$

$$\text{then } \lim_{n \rightarrow \infty} \frac{e_{n+1}}{e_n^2} = \frac{1}{2}|g''(r)| \quad (\text{quadratic convergence})$$

9/10

Binary Numbers

$b_i = 0$ or 1 (bits)

$$\dots b_2 b_1 b_0 \cdot b_{-1} b_{-2} b_{-3} \dots = \dots b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0 + b_{-1} \times 2^{-1} + b_{-2} \times 2^{-2} \dots$$

ex: (Terminating Binary)

$$(100.1)_2 = 0 \times 2^0 + 0 \times 2^1 + \underbrace{1 \times 2^2}_{4} + \underbrace{1 \times 2^{-1}}_{0.5} = (4.5)_{10}$$

↑ denotes binary

ex: (Nonterminating)

$$(0.\overline{10})_2 = (0.10101010\dots)_2$$

$$x = 0.\overline{10}$$

$$x = 0.\overline{9}$$

$$10x = 9.\overline{9}$$

$$2^2 x = 10.\overline{10} \leftarrow \text{multiplication by 2 shifts radix}$$

$$\text{subtract } 9x = 9$$

$$(2^2 - 1)x = (10)_2$$

$$x = 1$$

$$= (2)_{10}$$

$$\Rightarrow x = \frac{2}{3}$$

$$0.\overline{97} \times 100 = 97.\overline{97}$$

How to convert decimal to binary?

$$\text{ex. } (13.6)_{10} = (13)_{10} + (0.6)_{10}$$

$$\begin{aligned}\text{integer part : } \frac{13}{2} &= 6 \quad R_1 \\ \frac{6}{2} &= 3 \quad R_0 \\ \frac{3}{2} &= 1 \quad R_1 \\ \frac{1}{2} &= 0 \quad R_0\end{aligned}$$

stop when zero

read backward

$$(1101)_2 = (13)_{10}$$

$$\text{note } 1+2+8=13$$

$$\text{fractional part : } 0.6 \times 2 = 1.2 = 0.2 + 1$$

$$0.2 \times 2 = 0.4 = 0.4 + 0$$

$$0.4 \times 2 = 0.8 = 0.8 + 0$$

$$0.8 \times 2 = 1.6 = 0.6 + 1$$

$$0.6 \times 2 = 1.2 = 0.2 + 1$$

read the integer part

forward.

$$(0.\overline{1001})_2 = (0.6)_{10}$$

starts to repeat

$$\text{so } (13.6)_{10} = (1101.\overline{1001})_2$$

Floating point numbers:

- finite precision arithmetic

- IEEE - 754 standard (1985 - Institute of Electrical and Electronic Engineers)

Normalized (base 2) floating point.

$[\pm] 1. [\text{mantissa}] \times 2^{\text{exponent}}$

sign | string of bits
(0 or 1)
leading 1 not stored (it is implied)

this is the normalized part

$$\text{ex. } (13.6)_{10} = (1101.1)_2 = +1.1011 \times 2^3$$

Levels of position:

precision	sign	exponent	mantissa	Total bits
single	1	8	23	32
double	1	11	52	64

$x \in \mathbb{R} \rightarrow f(x)$ (floating point) \rightarrow computer word
 $\pm 1. \underline{\quad} \times 2^{\square}$ how floats are stored
 on a computer

Double precision word:

$s | E_1 E_2 \dots E_{11} | b_1 b_2 \dots b_{52}$

sign exponent mantissa

(1 bit) (11 bits) (52 bits)

$s = 0$ if positive

$s = 1$ if negative

Remark: For the exponent, we don't store the exponent E exactly, but rather

we store $E + \underbrace{10}_{\text{exponent bits}}$

Remark : The range of exponents we are allowing is $-1022 \leq E \leq 1023$
 for the float $\pm 1.\underline{b_1 b_2 \dots b_{52}} \times 2^E$

Rounding :

$$\text{ex: } (9.4)_{10} = (1001.\underline{0110})_2$$

$$= +1.\underline{001 \dots 1100} \underbrace{110 \dots}_{\text{remainder of the mantissa}} \times 2^3$$

needs to be cut somehow

option 1: (chopping)

- throw away the remainder
- bad since always rounding down!

option 2: IEEE Round to Nearest Rule

Rules: (the point of this rule is to try to cancel out equally likely errors.)

$$1) \pm 1.\underline{b_1 b_2 \dots b_{52}} \underbrace{00 \dots}_{\text{if all zeros, then chop bits beyond 52nd bit}} \times 2^E$$

$$2) \pm 1.\underline{b_1 b_2 \dots b_{52}} \underbrace{1 \dots}_{\text{if } b_{53} = 1 \text{ and rest of bits are not all zero, then round up by adding 1 to the 52nd bit and cutting if necessary.}} \times 2^E$$

$$3) \pm 1.\underline{b_1 b_2 \dots b_{52}} \underbrace{1 \dots}_{\text{if } b_{53} = 1 \text{ and rest of bits are all 0's then }} \times 2^E$$

then $\begin{cases} \text{if } b_{52} = 1, \text{ round up by adding 1 to } b_{52} \\ \text{else, if } b_{52} = 0, \text{ round down by truncating the remainder} \end{cases}$

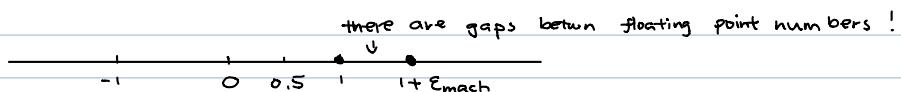
9/15

Floating Point : Part II

$$\pm 1.\underline{b_1 b_2 \dots b_{52}} \times 2^E$$

$\leftarrow -1022 \leq E \leq 1023$
 sign | mantissa $b_i = 0, 1$
 leading 1 " bits of precision
 $E = E_1, E_2, \dots, E_n$
 $E_1 = 0 \text{ or } 1$

Machine ϵ



- Defn: ϵ_{mach} is the distance between 1 and the next smallest representable number bigger than 1.

- In double precision

$$\epsilon_{\text{mach}} = 2^{-52} \approx 2.22 \times 10^{-16} \rightarrow \text{can expect accuracy of about 16 decimal places in double precision}$$

- Relative Rounding Error

For IEEE-754 with rounding to Nearest Rule,

$$|f(x) - x| < \frac{1}{2} \epsilon$$

$$\frac{1}{1 \times 1} = 2^{\text{mach}}$$

where $x \in \mathbb{R}$, $f_1(x)$ = normalized floating point number

$x \in \mathbb{R}$. $f_1(x)$ = float form

$$f_1(1) = +1.0\ldots01 \times 2^0$$

↳ replace $b_{52}=1$ to get next highest float

$$f_1(1 + \epsilon_{\text{mach}}) = +1.000\ldots01 \times 2^0 = 1 + \frac{2^{-52}}{\epsilon_{\text{mach}}}$$

numbers between 1 , $1 + \epsilon_{\text{mach}}$ get rounded to either 1 or $1 + \epsilon_{\text{mach}}$

ex: $f_1(x) \neq x$ for $x = 9.4$

$$(9.4)_{10} = (1001.0110)_2 \quad \text{remainder R to cut off}$$

$$f_1(9.4) = +1.00101100\ldots1100 \frac{1100}{1100} \times 2^3$$

(lost) By truncating, we lose $R = (0.1100)_2 \times 2^{-52} \times 2^3 = (0.1100)_2 \times 2^{-49} = 0.8 \times 2^{-49}$ in base 10

(add) By rounding to nearest rule 2), we add 1 to the b_{52} bit. to get an addition

$$\text{of } 2^{-52} \times 2^3 = 2^{-49}$$

$$\lceil \text{So } f_1(9.4) = 9.4 + 2^{-49} - (0.8 \times 2^{-49}) = 9.4 + 0.2 \times 10^{-49} \text{ rounding error}$$

not in float form, just the magnitude.

Remark:

1) 3 rules for arithmetic: $a \oplus b$, $a \ominus b$, $a \otimes b$, $a \oslash b$.

see Michael Overton notes for details.

2) Computations are performed in higher precision, but results are stored in double precision.

3) VPA (variable precision arithmetic)

⇒ help (vpa) in Matlab.

⇒ vpa(pi, 100) want 100 digits

Loss of significance

$$\Rightarrow \frac{1}{1 - (1 + \epsilon_{\text{mach}}/2)}$$

⇒ -Inf (note $1 + \frac{1}{2}$ rounded to 1)

- the answer should be $-\frac{2}{\epsilon_{\text{mach}}} = -2^{53}$

- the problem here is due to cancellation of nearly equal numbers.

called catastrophic cancellation.

ex: $x^2 + 9^{12}x = 3$

$$\text{solution should be } \frac{-9^{12} \pm \sqrt{(9^{12})^2 - 4(1)(-3)}}{2(1)}$$

$$x_{\pm} = \frac{-9^{12} \pm \sqrt{9^{24} + 12}}{2} \quad \text{in Matlab} \quad x_1 = 0 \quad x_2 \approx -2.82 \times 10^{11}$$

$$x_2^2 + 9^{12}x_2 - 3 = -3 \quad (\text{matlab})$$

using matlab $x_+ = 0$, definitely not a solution

$$x_- \approx -2.82 \times 10^{11}$$

Two options to get positive root x_+ ?

option 1: use VPA or symbolic arithmetic

option 2: stay in double precision but modify computation.

$$ax^2 + bx + c = 0 \quad x_+ = \frac{-b + \sqrt{b^2 - 4ac}}{2a} / \left(b + \frac{\sqrt{b^2 - 4ac}}{2a} \right) = -2c \quad (\text{no more catastrophic cancellation})$$

$$\text{By hand } x_+ = \frac{b}{q^{12} + \sqrt{q^{24} + 12}} = 1.0622 \times 10^{-11}$$

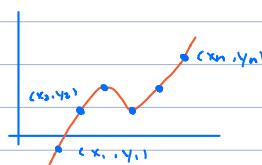
$$x_+^2 + q^k x_+ - 3 = 0$$

$$\frac{b}{b + \sqrt{b^2 - 4ac}} \quad \text{if } b \gg 1$$

9/17

System of Equations

ex:



Suppose want to fit a polynomial

$$p(x) = a_0 + a_1 x + \dots + a_{n-1} x^{n-1} \in P_{n-1}$$

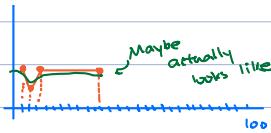
$\nwarrow \uparrow$
n unknown coefficients

$$\text{Get } p(x_0) = y_1, \quad \left[\begin{array}{l} \text{system of eqns} \\ p(x_1) = y_2 \\ \vdots \\ p(x_n) = y_n \end{array} \right]$$

-for n unknowns, $\{a_i\}_{i=0}^{n-1}$

$$\text{ex: } \left[\begin{array}{l} -u''(x) = -p(x) \\ u(0) = u_0, \quad u'(0) = u_1 \end{array} \right] \quad [0 \leq x \leq 100]$$

IVP



After discretizing, get linear system $A\vec{x} = \vec{b}$ to solve.

usually very large.

How to solve $A\vec{x} = \vec{b}$?

$$A = A_{n \times n} = \begin{pmatrix} a_{00} & a_{01} & \dots & a_{0n} \\ \vdots & & & \vdots \\ a_{n0} & a_{n1} & \dots & a_{nn} \end{pmatrix}$$

$$\vec{x} = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix}$$

$$(A^{-1})_{n \times n} \text{ is the matrix s.t. } AA^{-1} = A^{-1}A = I_{n \times n} = \begin{pmatrix} 1 & & 0 \\ 0 & \dots & 0 \\ 0 & & 1_n \end{pmatrix}_{n \times n}$$

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

$$A^{-1}A\vec{x} = A^{-1}\vec{b}$$

$\vec{x} = A^{-1}\vec{b}$ taking matrix inverse is computationally bad.

Componentwise Computations.

$$\textcircled{1} \quad \vec{x} \cdot \vec{y} = \sum_{i=1}^n x_i y_i =: \underline{\underline{x_i y_i}}$$

[called Einstein summation convention.]

$$\textcircled{2} \quad A\vec{x} = \vec{b}$$

$$(\vec{b})_i = (A\vec{x})_i = \sum_{j=1}^n a_{ij} x_j = a_{ij} x_j \quad \text{two repeated indices imply a summation}$$

$$\textcircled{3} \quad (AB)_{ij} = \sum_{k=1}^n a_{ik} b_{kj} = a_{ik} b_{kj}$$

Big O Notation

Def: $f(n) = O(g(n))$ as $n \rightarrow \infty$ if $\exists M, N$ s.t. $|f(n)| \leq M|g(n)| \quad \forall n \geq N$

$$\text{ex: } n^3 + 2n^2 - n = O(n^3) = n^3 + O(n^2)$$

Apply triangle inequality $|n^3 + 2n^2 - n| \leq n^3 + 2n^2 + n \leq 4n^3$ for $M=4, N=1$
 $n^2 \leq n^3, n \leq n^3$, provided $n \geq 1$

$$\text{ex: } 5n^4 + n^3 + 1 = O(6n^4) = O(n^4) = 5n^4 + O(n^3)$$

suppose $n = 10^6$

then $5n^4 = 5 \cdot 10^{24}$ & dominating term

$$n^3 + 1 = O(n^3) + 1 = O(n^3)$$

(Naive) Gaussian Elimination ($A\vec{x} = \vec{b}$)

Augmented matrix

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 2 & -2 & 1 & 2 \end{array} \right)$$

pivot: $R_3 - 2R_1$

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 0 & -6 & 3 & -2 \end{array} \right)$$

Subtract $-2R_2$ from R_3

$$\left(\begin{array}{ccc|c} 1 & 2 & -1 & 2 \\ 0 & 3 & 1 & 4 \\ 0 & 0 & 5 & 6 \end{array} \right)$$

for:

$$\begin{aligned} x_1 + 2x_2 - x_3 &= 2 \\ 3x_2 + x_3 &= 4 \\ 2x_1 - 2x_2 + x_3 &= 2 \end{aligned}$$

Elimination step

upper triangular matrix

$$\left(\begin{array}{ccc|c} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{array} \right)$$

upper-tria.

$$U\vec{x} = \vec{c}$$

$U_{ij} = \begin{cases} \text{something} & i \leq j \\ 0 & i > j \end{cases}$

Back solve step.

$$x_1 + 2x_2 - x_3 = 2$$

$$x_2 = 2 - 2x_1 + x_3$$

$$3x_2 + x_3 = 4$$

$$x_2 = 1/3(4 - x_3)$$

$$5x_3 = 6$$

$$x_3 = 6/5$$

(Naive) Gaussian Elimination ($A\vec{x} = \vec{b}$)

Operation Counts: $A = An \times n$

① Elimination step (to get $U\vec{x} = \vec{c}$)

Requires $\frac{n^3}{3} + \frac{n^2}{2} - \frac{5n}{6}$ multiplications / divisions.

$$= O\left(\frac{n^3}{3}\right)$$
 with a similar number of additions / subtractions.

+, -, ., /, * counts are called flops.

ex: For $A_{100 \times 100}$, need $\approx \frac{1}{3}10^3 = \frac{1}{3}10^6$ multiplications / divisions

② Backsolve step (to get \vec{x})

Requires $\frac{n^2}{2} + \frac{n}{2}$ multiplications / divisions.

$$= O\left(\frac{n^2}{2}\right)$$

ex: For $A_{1000 \times 1000}$, need $\approx \frac{100^2}{2} = \frac{1}{2}10^4$ flops

Remarks: ① Elimination is much more expensive

② Operation counts can be used to predict computation times.

ex: If know $A_{10 \times 10}$ takes 10^{-2} seconds to solve, can use operation count to give an estimate

for $A_{1000000 \times 1000000}$

③ Counts can be used to compare algorithms.

Operation count example:

$$(A\vec{x})_i = \sum_{j=1}^n a_{ij}x_j \quad (1 \leq i \leq n)$$

For just the i^{th} entry, need n multiplications, one for each $1 \leq j \leq n$

$n-1$ additions, since adding n terms

$$= n + (n-1) = 2n - 1 \text{ flops}$$

For entire vectors $A\vec{x}$, need $n(2n-1) = 2n^2 - n = O(n^2)$ flops

$$\text{ex: } (AB)_{ij} = \sum_{k=1}^n A_{ik}B_{kj} \quad AB \in \mathbb{R}^{n \times n}$$

$$AB = A[C_1, C_2, \dots, C_n]$$

flops.

Using previous result from matrix-vector product, we need $n(2n^2 - n) = 2n^3 - n^2 = O(n^3)$

LU Decomposition (matrix form of elimination)

$$\text{Goal: } A = LU = \begin{pmatrix} 1 & 0 \\ * & 1 \end{pmatrix} \begin{pmatrix} 1 & * \\ 0 & 1 \end{pmatrix} = \text{lower} \leftarrow \text{triangular} \quad \text{Upper} \leftarrow \text{triangular}$$

- If 1's on main diagonal of L this is Pivoted decomposition.

- Idea: Getting LU encodes the elimination step of Gaussian elimination (this is $O(n^3)$)

If $Ax = b$, then $LUx = b$

- ① solve $Ly = b$ for y by forward substitution
 - ② solve $Ux = y$ for x by back substitution
-] both are $O(n^2)$ in complexity

Remark: Don't solve ①, ② by inverting L or U.

$$\text{ex: } A = \begin{pmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \\ 2 & -2 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 2 & -1 \\ (0) & 3 & 1 \\ (2) & -6 & 3 \end{pmatrix} \xrightarrow{\substack{\text{subtract } (0) \text{ from } R_2 \\ \text{subtract } -2R_1 \text{ from } R_3}} \begin{pmatrix} 1 & 2 & -1 \\ (0) & 3 & 1 \\ (2) & (-2) & 5 \end{pmatrix}$$

↓ storing multiplies
 for convenience
 these will form L
 there is really zeros
 in these slots

$$\text{claim: } A = LU = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & -1 \\ 0 & 3 & 1 \\ 0 & 0 & 5 \end{pmatrix} \quad \text{Pivoted since 1's on main diagonal of L}$$

in Matlab.

$\Rightarrow [L, U] = \text{lu}(A)$;

$\Rightarrow [L, U, P] = \text{lu}(A)$;

↳ pivot matrix

Remark: Suppose want to solve $Ax_i = b_i$ with same A each time.
 $Ax_1 = b_1$
 $Ax_2 = b_2$
 \vdots
 $Ax_r = b_r$

If separately solving, cost is $O(rn^3 + rn^2) \sim O(rn^4)$ if $r \ll n$

But if eliminating once (via LU decomposition) then cost is $O(n^3 + rn^2) \sim O(n^3)$ if $r \ll n$.

⇒ cheaper.

Sources of error in Gaussian elimination (LU)

- ① $A = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ Have 0 pivot and can't eliminate A_{21} entry.
 solution is to swap rows (will get us a $PA = LU$ decomposition)
 ↳ permutation matrix

② Swapping: suppose $\epsilon < \epsilon_{\text{mach}} \approx 10^{-16}$ (say $\epsilon = 10^{-20}$)

$$A = \begin{pmatrix} \frac{1}{\epsilon} & 1 \\ \epsilon & 1 \end{pmatrix} = \underbrace{\begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix}}_L \underbrace{\begin{pmatrix} \frac{1}{\epsilon} & 1 \\ 0 & -\frac{1}{\epsilon} \end{pmatrix}}_U$$

problem term, in double precision, stored as $-\frac{1}{\epsilon}$

$$\text{stored as } \rightarrow \begin{pmatrix} 1 & 0 \\ \frac{1}{\epsilon} & 1 \end{pmatrix} \begin{pmatrix} \frac{1}{\epsilon} & 1 \\ 0 & -\frac{1}{\epsilon} \end{pmatrix} = \begin{pmatrix} \epsilon & 1 \\ 1 & 0 \end{pmatrix} \neq A$$

↳ ↳

- In finite precision can lead to catastrophic cancellation (called swamping)

- solution again is to swap rows.

$$\begin{pmatrix} 1 & 1 \\ \epsilon & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \epsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1-\frac{1}{\epsilon} \end{pmatrix}$$

stored as 1 on computer

$$= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \text{ stored as } \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}$$

$PA = LU$ Decomposition

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \xrightarrow{\text{switch}} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

R₁ with R₂

switch 1st column for larger entry and swapping that row to the top (called partial pivoting by rows)

In matrix form, use $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ permutation matrix (an nn matrix with a single 1 in each column / row)

$$\rightarrow \text{get } PA = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \text{ row swapped}$$

suppose have $PA = LU$ to solve $Ax = b$, 1) $PAx = Pb$

$$2) LUx = Pb$$

$\xrightarrow{\text{L}} \text{solve } Lc = Pb \text{ for } c \text{ using}$

$$\Rightarrow x = A^{-1}b;$$

forward substitution

choose most optimal

method (Matlab \ command)

then solve $Ux = c$ for x by back substitution.

$$\text{ex: } \begin{pmatrix} 3 & 7 \\ 6 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \text{ solve using } PA = LU$$

$$A = \begin{pmatrix} 3 & 7 \\ 6 & 1 \end{pmatrix} \xrightarrow{P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}} \begin{pmatrix} 6 & 1 \\ 3 & 7 \end{pmatrix} \xrightarrow{\substack{\text{subtract } (\frac{1}{2})R_1 \text{ from } R_2 \\ \rightarrow \text{swap } R_1, R_2 \text{ of } A}} \begin{pmatrix} 6 & 1 \\ -\frac{9}{2} & \frac{13}{2} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 6 & 1 \\ 0 & \frac{13}{2} \end{pmatrix} = LU$$

$$\text{Have } PA = LU \quad (P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}) A = \begin{pmatrix} 1 & 0 \\ \frac{1}{2} & 1 \end{pmatrix} \begin{pmatrix} 6 & 1 \\ 0 & \frac{13}{2} \end{pmatrix}$$

$$\text{Now (1) solve } Lc = Pb \text{ for } c \text{ to get } c = \begin{pmatrix} -1 \\ \frac{13}{2} \end{pmatrix}$$

$$\text{(2) solve } Ux = c \text{ for } x \text{ to get } \vec{x} = \begin{pmatrix} -1 \\ \frac{13}{2} \end{pmatrix}$$

Algorithm for back-substitution

$$\begin{aligned} Ux &= \vec{b} \\ &\quad \begin{matrix} u_{11}x_1 + u_{12}x_2 + \dots + u_{1n}x_n = b_1 \\ \vdots \\ u_{nn}x_n = b_n \end{matrix} \\ \text{Have } x_n &= \frac{b_n}{u_{nn}} \text{ in general, } x_i = \frac{1}{u_{ii}}(b_i - \sum_{j=i+1}^n u_{ij}x_j) \end{aligned}$$

For HW 3, do an operation count to construct $\vec{x} = (x_1, \dots, x_n)$

Q124

Error for Systems $Ax = b$

Error : suppose x_a is an approximate solution

Definition : $r = b - Ax_a$ is the residual vector

backward error = $\|b - Ax\|$ (think |f'(x)| for root-finding)

forward error = $\|x - x_a\|$

vector / matrix norms : suppose $x \in \mathbb{R}^n$

A norm satisfies (i) $\|x\| \geq 0$ and $\|x\| = 0 \iff x = 0$

(ii) $\|tx\| = |t|\|x\|$ (scaling)

(iii) $\|x+y\| \leq \|x\| + \|y\|$ (triangle inequality)

- can also define matrix norms (for $\mathbb{R}^{n \times m}$)

ex: $\|x\|_2 = \left(\sum_{i=1}^n x_i^2 \right)^{1/2}$ Euclidean or 2-norm

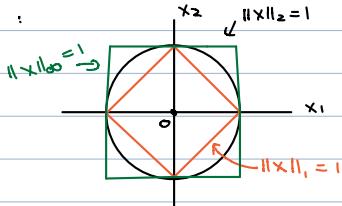
$\|x\|_1 = \sum_{i=1}^n |x_i|$ taxicab norm



$\|x\|_\infty = \max_{i=1,\dots,n} |x_i|$

there are others, for instance, $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$

Unit circles:



Matrix Norms: $A \in \mathbb{R}^{n \times n}$, $\|A\|$ (or max matrix)

need $\|\cdot\|: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}$ to satisfy norm properties

① $\|A\|_F := \left(\sum_{i=1}^n \sum_{j=1}^n |a_{ij}|^2 \right)^{1/2}$ Frobenius norm

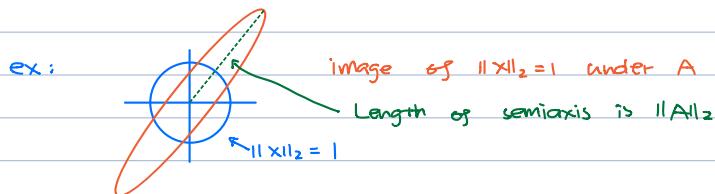
② Operator p-norms:

$$\|A\|_p = \max_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p=1} \|Ax\|_p \quad \text{since } \frac{\|Ax\|_p}{\|x\|_p} = \|A\left(\frac{x}{\|x\|_p}\right)\|_p$$

Vector of length 1

- measures maximum stretching of unit sphere $\|x\|_p=1$ by A

- matrix norm is inherited from vector norm



Common p-norms:

① $\|A\|_1 = \max_{j=1,\dots,n} \sum_{i=1}^n |a_{ij}|$ = maximum absolute column sum of A

② $\|A\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^n |a_{ij}|$ = maximum absolute row sum of A

③ $\|A\|_2 = \sqrt{\sigma(A^T A)}$ where $\sigma(B) = \max_i |\lambda_i|$ (the "spectral radius")
 λ_i are eigenvalues of B

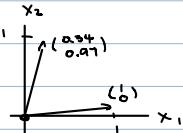
In matlab, would use `>> norm (A, "1, 2, inf, --")`

Error Magnification

$$A = \begin{pmatrix} 4.1 & 2.8 \\ 9.7 & 6.6 \end{pmatrix} \quad b = A(:,1) = \begin{pmatrix} 4.1 \\ 9.7 \end{pmatrix} \quad x = A \setminus b = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

suppose now b is $b_{\text{err}} = \begin{pmatrix} 4.11 \\ 9.7 \end{pmatrix}$ ^{error of 0.01}

then $x = A \setminus b_{\text{err}} = \begin{pmatrix} 0.34 \\ 0.97 \end{pmatrix}$



\downarrow Approximate solution
 $Ax = b$. On a computer $A\tilde{x} = \tilde{b} \leftarrow b$ stored with error (so is A but we ignore for now)

blkwd. error	full. error
absolute $\ b - A\tilde{x}\ _p$	$\ x - \tilde{x}\ _p$
relative $\frac{\ b - A\tilde{x}\ _p}{\ b\ _p}$	$\frac{\ x - \tilde{x}\ _p}{\ x\ _p}$

How does $\frac{\|b - \tilde{b}\|}{\|b\|}$ affect $\frac{\|x - \tilde{x}\|}{\|x\|}$?
error in problem data error in problem solution

$$A(x - \tilde{x}) = \tilde{b} - b$$

$$\|x - \tilde{x}\| = \|A^{-1}(\tilde{b} - b)\| \leq \|A^{-1}\| \cdot \|\tilde{b} - b\|$$

Also have $b = Ax$ and so $\|b\| = \|Ax\| \leq \|A\| \|x\|$

then we have $\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\|A\| \|A^{-1}\|}{\|A\|} \frac{\|\tilde{b} - b\|}{\|b\|}$

error in soln. error in data
 this factor is called the condition

number of A : $\text{cond}_p(A) := \|A\|_p \|A^{-1}\|_p$

Remarks: ① $\text{Cond}_p(A)$ is the max. error magnification in solving $Ax = b$

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \text{cond}_p(A)$$

$$\frac{\|\tilde{b} - b\|}{\|b\|}$$

② If $\text{cond}(A)$ is "small", this is good. If $\text{cond}(A)$ is "large" ($10^9, 10^{20} \dots$) is bad.

↳ called these ill-conditioned problems

③ data $\frac{\|b - \tilde{b}\|}{\|b\|}$ error is on order of $\epsilon_{mach} \approx 10^{-16}$ in double precision

④ If $\text{cond}_p(A) \sim \frac{1}{\epsilon_{mach}}$, can expect good accuracy.

If $\text{cond}_p(A) \sim 10^k$, expect to lose k digits of accuracy in computing x .

9/29

Iterative Method for systems

Until now, solve $Ax = b$ via direct method

(i.e. solve in finite # of steps in extra arithmetic).

(i.e. $PA = LU$)

Can try iteration:

$$\underbrace{Ax - b + x}_{g(x)} = x \quad \text{with } x_{k+1} = g(x_k) \quad \text{it will turn out to not guarantee convergence}$$

Jacobi Method:

split A as $A = D + R$ ($D = \text{diagonal of } A$, $R = \text{everything else}$)

$$\begin{matrix} A & D & R \\ \text{ex: } & \begin{pmatrix} 2 & 0 \\ -1 & 4 \end{pmatrix} & = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix} + \begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} \end{matrix}$$

Now $Ax = b \Rightarrow (D + R)x = b \Rightarrow Dx = b - Rx \Rightarrow x = D^{-1}(b - Rx)$

Jacobi Method: $x_{k+1} = D^{-1}(b - Rx_k)$

$x_0 = \text{starting guess}$

$$D = \begin{pmatrix} d_1 & & \\ & d_2 & \\ & & \ddots & d_n \end{pmatrix} \quad D^{-1} = \begin{pmatrix} d_1^{-1} & & \\ & d_2^{-1} & \\ & & \ddots & d_n^{-1} \end{pmatrix}$$

Remarks : ① Need stopping criteria.

i) Iterate finite (say N) number of times, then stop.

ii) can use forward or backward "error": $\|x_{k+1} - x_k\|_p < \text{TOL} (10^{-6})$

$$\text{or } \|Ax_k - b\|_p < \text{TOL}$$

② Convergence guaranteed if A is strictly diagonally dominant: $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$

$$\text{ex: } A = \begin{pmatrix} 2 & 0 & 1 \\ -1 & 4 & 1 \\ 0 & 3 & 5 \end{pmatrix} \quad A \text{ is SPD since } |2| > |0| + |1|, |5| > |0| + |3| \\ |4| > |-1| + |1|$$

Note If A is not SPD, can try swapping rows or columns to make it SPD

③ Iterative method often preferred if A is sparse (has a lot of zeros)

- can sometime get $O(n^2)$ cost in solving sparse system.

④ Two other methods are Gauss-Seidel and SOR (successive over-relaxation)

Nonlinear Systems :

How to solve $\vec{x} = \vec{g}(\vec{x})$ If \vec{g} is not linear? (can't write \vec{g} as matrix A)

$$\text{ex: } \begin{cases} f_1(x, y) = e^x - y = 0 \\ f_2(x, y) = xy - e^x = 0 \end{cases} \quad \text{can write as } \begin{matrix} \vec{F}(\vec{x}) = \vec{0} \\ \begin{pmatrix} e^x - y \\ xy - e^x \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \end{matrix}$$

Recall 1d Newton:

$$O = f(x_1) = f(x_0) + f'(x_0)(x_1 - x_0) + O((x_1 - x_0)^2)$$

new guess initial guess

$\Rightarrow x_1 = x_0 - f(x_0)/f'(x_0)$ and then iterate ...

Multivariable Newton method:

$$O = \vec{F}(\vec{x}_1) = \vec{F}(\vec{x}_0) + \underbrace{D\vec{F}(\vec{x}_0)(\vec{x}_1 - \vec{x}_0)}_{= \begin{pmatrix} \partial x_1 f_1 & \partial x_2 f_1 & \dots & \partial x_m f_1 \\ \vdots & & & \\ \partial x_1 f_m & \partial x_2 f_m & \dots & \partial x_m f_m \end{pmatrix}} + O(\|\vec{x}_1 - \vec{x}_0\|^2) \quad \text{vector-valued taylor expansion}$$

$$-\vec{F}(\vec{x}_0) = D\vec{F}(\vec{x}_0)(\vec{x}_1 - \vec{x}_0)$$

$$-[D\vec{F}(\vec{x}_0)]^{-1} \vec{F}(\vec{x}_0) = \vec{x}_1 - \vec{x}_0 \quad \Rightarrow \quad \vec{x}_1 = \vec{x}_0 - [D\vec{F}(\vec{x}_0)]^{-1} \vec{F}(\vec{x}_0)$$

$$\text{Method: } \begin{cases} \vec{x}_{n+1} = \vec{x}_n - [D\vec{F}(\vec{x}_n)]^{-1} \vec{F}(\vec{x}_n) \\ \vec{x}_0 = \text{starting guess} \end{cases}$$

To avoid computing inverses, method is to get

$$\vec{s} = [D\vec{F}(\vec{x}_n)]^{-1} \vec{F}(\vec{x}_n)$$

$$D\vec{F}(\vec{x}_n) \cdot \vec{s} = \vec{F}(\vec{x}_n) \quad \leftarrow \text{linear system}$$

① Fix \vec{x}_0 starting guess

② Solve for \vec{s} in the linear system $D\vec{F}(\vec{x}_0) \vec{s} = \vec{F}(\vec{x}_0)$

with PA = LU or Matlab \ command

③ Update $\vec{x}_1 = \vec{x}_0 - \vec{s}$

④ Repeat the iteration

Convergence of Multivariable Newton

$$\text{Suppose } \vec{x}_{n+1} = \vec{g}(\vec{x}_n)$$

In 1d, FPI converges locally if $|g'(r)| < 1$ (since then have contraction mapping)

In the vector case, FPI converges locally if $\rho(D\vec{g}(r)) < 1$

$$\text{where } \rho(A) = \max_{i=1,\dots,n} |\lambda_i| \text{ and } \lambda_i \text{ are eigenvalues of } A$$

Thm: $\rho(A) \leq \|A\|$ for any matrix operator norm

Remark: For Newton, $\tilde{F}(F) = F - [DF(F)]^{-1}F(0)$ so $D\tilde{F}(F)$ is very complicated.

Ex: $\begin{cases} f_1(x, y) = e^x - y = 0 & (\tilde{F}(x) = 0) \\ f_2(x, y) = xy - e^x = 0 \end{cases}$ Solution is $(x, y) = (1, e)$

Jacobian: $DF(\vec{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{pmatrix} = \begin{pmatrix} e^x & -1 \\ y - e^x & x \end{pmatrix}$

① Fix $\vec{x}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ (just need close to root)

② Solve $D\tilde{F}(\vec{x}_0)\vec{s} = \vec{F}(\vec{x}_0)$ for \vec{s}

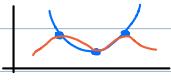
or $\begin{pmatrix} 1 & -1 \\ -1 & 0 \end{pmatrix} \begin{pmatrix} s_1 \\ s_2 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \Rightarrow \begin{cases} s_1 = 1 \\ s_2 = 0 \end{cases} \text{ so } \vec{s} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$

③ $\vec{x}_1 = \vec{x}_0 - \vec{s} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$

④ Repeat

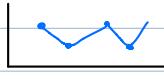
10/6

Interpolation



- polynomial $p \in P_n$ degree n

- trig



- piecewise (local) interpolation

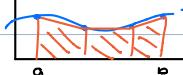
pw linear

Uses:

- i) To approximate functions



ii) Riemann sum



pw linear interpolant

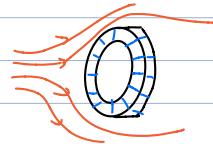
iii) ODE: $y' = f(t, y(t))$

$$\rightarrow y(t) = y_0 + \int_0^t f(s, y(s)) ds$$

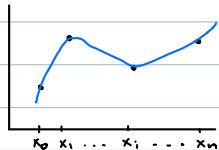
can approximate using interpolation

iv) PDE: finite element methods.

use 2D linear interpolation



Polynomial Interpolation



$(x_i, y_i) \quad i=0, 1, \dots, n$ data points

want to fit degree n (or less) polynomial $p \in P_n$ set of n degree polynomials (or less)

Theorem: If the $n+1$ points x_i are distinct, then $\exists! p \in P_n$ interpolating $\{(x_i, y_i)\}_{i=0}^n$.

To construct $p \in P_n$, need a basis of P_n .

C a linearly independent spanning set

Def: ① A set $\{v_i\} \subset V$ is linearly independent.

$$\text{if } c_1v_1 + \dots + c_nv_n = 0 \Rightarrow c_i = 0 \quad \forall i$$

(2) $\{V_i\}$ span V if any $v \in V$ can be written as a linear combination of the $\{V_i\}$.

Monomial basis of $P_n = \{1, x, x^2, \dots, x^n\}$

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \dots + a_n x^n \in P_n$$

to interpolate, we want $P_n(x_i) = y_i \quad \forall i = 0, 1, \dots, n$

get system to solve for $\{a_i\}_{i=0}^n$.

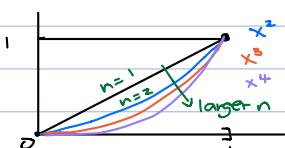
$$\begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}$$

$\vec{y} = A\vec{a}$ - can solve system in Matlab $a = A \setminus y$

A - a "Vandermonde matrix" (very ill-conditioned)

For $n \approx 10$, $\text{cond}(A) = 10^{12}$ in general

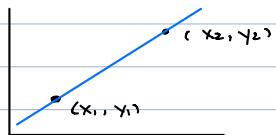
$\{1, x, x^2, \dots, x^n\}$ is not "sufficiently" linearly independent



so for large n the basis functions look less and less linearly independent.

So for n large, monomial basis is bad.

Lagrange Interpolation - basis for P_n .



linear interpolation of 2 points

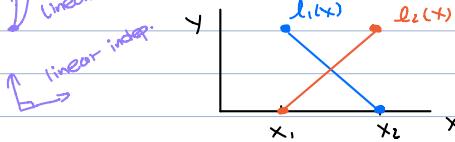
$$\text{monomial basis} = p(x) = a_0 + a_1 x$$

$$\text{Lagrange basis} = p(x) = \underbrace{y_1 \frac{x-x_2}{x_1-x_2}}_{l_1(x)} + \underbrace{y_2 \frac{x-x_1}{x_2-x_1}}_{l_2(x)}$$

linear Lagrange basis functions

$$p(x) = y_1 \frac{x-x_2}{x_1-x_2} + y_2 \frac{x-x_1}{x_2-x_1} = y_1 l_1(x) + y_2 l_2(x)$$

linear depen.
linear interp.



$$l_1(x) = \text{plug } x_1 \text{ into } p(x) \text{ get } \frac{x_1-x_2}{x_1-x_2} = 1$$

$$x_2 \text{ into } p(x) \text{ get } \frac{x_2-x_1}{x_2-x_1} = 0$$

l_1, l_2 "sufficiently" linearly independent

$$\text{at the nodes, } l_i(x_j) = \begin{cases} 1 & \text{if } i=j \\ 0 & \text{if } i \neq j \end{cases} = \delta_{ij} \quad (\text{Kronecker delta})$$

matrix system looks like

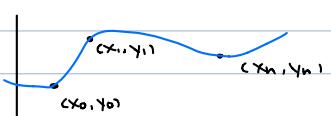
$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

$$A_{ij} = l_i(x_j) = \delta_{ij}$$

note that $\text{cond}(I) = 1$

$$\text{cond}(I) = \|I\| \|I^{-1}\| = \|I\|^2 = 1 \quad \text{well-conditioned.}$$

$n+1$ point cases



Lagrange interpolant is $P_n(x) = \sum_{i=0}^n y_i l_i(x)$

skipped $j=i$ term

$$l_i(x) = \frac{\prod_{j=0, j \neq i}^n (x - x_j)}{(x_i - x_0)(x_i - x_1) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_n)}$$

Remarks : (1) note $l_i \in P_n, \forall i=0, 1, \dots, n$ since have $(n+1)-1$ terms in the product

(2) we have interpolation:

$$P_n(x_k) = \sum_{i=0}^n y_i \underline{l_i(x_k)} = y_{k+1} = y_k$$

$$= \delta_{ik}$$

$$= \begin{cases} 1 & \text{if } i=k \\ 0 & \text{if } i \neq k \end{cases}$$

Newton's Divided Differences (interpolation)

Recall : ① Monomial basis : $p(x) = \sum_{i=0}^n a_i x^i$ basis functions

- simple form but ill-conditioned system (Vandermonde) to obtain.

② Lagrange basis : $p_n(x) = \sum_{i=0}^n y_i \left(\prod_{j=0, j \neq i}^n \frac{x - x_j}{x_i - x_j} \right)$ basis functions

[easy to find, well-conditioned, but more complicated form.]

Ex: (1, 1), (2, 5), (3, 4) use Lagrange.

$$P_2(x) = 1 + \frac{(x-2)(x-3)}{(1-2)(1-3)} + 5 \cdot \frac{(x-1)(x-3)}{(2-1)(2-3)} + 4 \cdot \frac{(x-1)(x-2)}{(3-1)(3-2)}$$

Newton basis for interpolation : $\{(x_i, y_i)\}_{i=0}^n$

$$P(x) = a_0 \cdot 1 + a_1 \cdot (x - x_0) + a_2 \cdot (x - x_0)(x - x_1) + \dots + a_n \cdot (x - x_0)(x - x_1) \dots (x - x_{n-1})$$

↳ basis functions

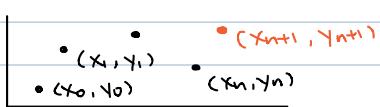
basis functions : $N_0(x) = 1, N_i(x) = \prod_{k=0}^{i-1} (x - x_k)$ for $i = 1, 2, \dots, n$

$$\text{Ex: } y_0 = P(x_0) = a_0 \cdot 1 + 0 + 0 \quad \text{get lower triangular system } O(n^2)$$

$$y_1 = P(x_1) = a_0 \cdot 1 + a_1 \cdot (x_1 - x_0) + 0$$

$$y_2 = P(x_2) = a_0 \cdot 1 + a_1 \cdot (x_2 - x_0) + a_2 \cdot (x_2 - x_0)(x_2 - x_1)$$

For Newton, easy to add new points :



Let $P_n \in P_n$ interpolate $n+1$ points

set $P_{n+1}(x) = P_n(x) + C_{n+1} N_{n+1}(x)$ ↳ for $x = x_0, \dots, x_n$
some constant
 P_{n+1} interpolates original $n+1$ points.

now can choose C_{n+1} so P_{n+1} interpolates new point

$$y_{n+1} = P_{n+1}(x_{n+1}) = P_n(x_{n+1}) + C_{n+1} N_{n+1}(x_{n+1})$$

$$C_{n+1} = \frac{y_{n+1} - P_n(x_{n+1})}{N_{n+1}(x_{n+1})} \quad \text{known as the } (n+1)^{\text{st}} \text{ "divided difference"}$$

Notation : $C_{n+1} = f[x_0, \dots, x_n, x_{n+1}]$

Method : Let $y_i = f(x_i)$

$$\begin{array}{c|ccccc} x_0 & f(x_0) & \frac{f(x_1) - f(x_0)}{x_1 - x_0} & \frac{f(x_2) - f(x_1)}{x_2 - x_0} & \dots & \frac{f(x_n) - f(x_{n-1})}{x_n - x_0} \\ x_1 & f(x_1) & \frac{f(x_2) - f(x_1)}{x_2 - x_1} & \frac{f(x_3) - f(x_2)}{x_3 - x_1} & \dots & \frac{f(x_n) - f(x_{n-1})}{x_n - x_1} \\ x_2 & f(x_2) & \frac{f(x_3) - f(x_2)}{x_3 - x_2} & \frac{f(x_4) - f(x_3)}{x_4 - x_2} & \dots & \frac{f(x_n) - f(x_{n-2})}{x_n - x_2} \\ x_3 & f(x_3) & \frac{f(x_4) - f(x_3)}{x_4 - x_3} & \frac{f(x_5) - f(x_4)}{x_5 - x_3} & \dots & \frac{f(x_n) - f(x_{n-3})}{x_n - x_3} \end{array} \quad \begin{array}{l} \frac{f(x_1) - f(x_0)}{x_1 - x_0} =: f[x_0, x_1] \\ \frac{f(x_2) - f(x_1)}{x_2 - x_1} =: f[x_1, x_2] \\ \frac{f(x_3) - f(x_2)}{x_3 - x_2} =: f[x_2, x_3] \\ \vdots \\ \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}} =: f[x_{n-1}, x_n] \end{array} \quad \begin{array}{l} \frac{f(x_2) - f(x_1)}{x_2 - x_0} =: f[x_0, x_1, x_2] \\ \frac{f(x_3) - f(x_2)}{x_3 - x_0} =: f[x_0, x_1, x_2, x_3] \\ \vdots \\ \frac{f(x_n) - f(x_{n-2})}{x_n - x_0} =: f[x_0, \dots, x_n, x_{n-2}] \end{array} \quad \begin{array}{l} \text{top row gives coefficients} \\ \{a_n\} \end{array}$$

definition of divided differences :

$$f[x_k] := f(x_k)$$

$$f[x_k, x_{k+1}] = \frac{f[x_{k+1}] - f[x_k]}{x_{k+1} - x_k}$$

$$f[x_k, x_{k+1}, x_{k+2}] = \frac{f[x_{k+2}] - f[x_{k+1}]}{x_{k+2} - x_{k+1}}$$

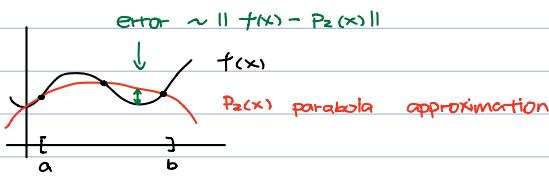
$$P(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \dots$$

Ex: (1, 1), (2, 5), (3, 4)

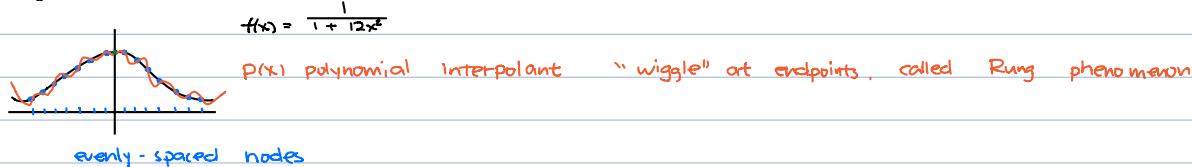
$$\begin{array}{c|ccccc} 1 & 1 & \frac{5-1}{2-1} = 4 & \frac{-1-4}{3-1} = -\frac{5}{2} & \frac{4-\frac{5}{2}}{4-2} = \frac{3}{2} & \frac{6-\frac{3}{2}}{4-3} = 2 \\ 2 & 5 & 4 & -\frac{5}{2} & \frac{3}{2} & \frac{3}{2} \\ 3 & 4 & \frac{4-5}{3-2} = -1 & 2-\frac{5}{2} = \frac{3}{2} & \frac{3}{2} & \frac{3}{2} \\ 4 & 6 & \frac{6-4}{4-3} = 2 & 4-2 & \frac{3}{2} & \frac{3}{2} \end{array}$$

$$P(x) = 1 + 4(x-1) + \left(-\frac{5}{2}\right)(x-1)(x-2) + \left(\frac{3}{2}\right)(x-1)(x-2)(x-3)$$

Interpolation Error (Approximation Theorem)



Runge Phenomenon



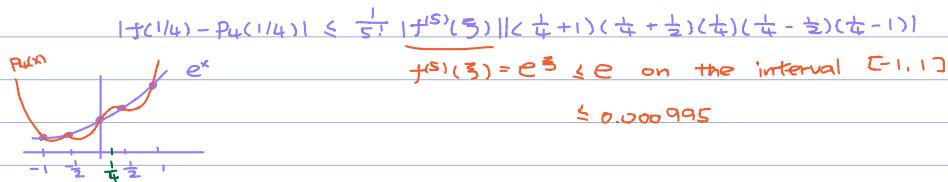
- Moral: High degree interpolation does not imply accuracy (in general)
- Solution: Move some nodes towards endpoints for higher accuracy

Thm: Let $f \in C^{n+1} [a, b]$, and $P_n \in P_n$ be a polynomial interpolant at nodes $x_0, \dots, x_n \in [a, b]$
 then $\forall x \in [a, b], \exists \xi_n \in (a, b)$
 s.t. $|f(x) - P_n(x)| = \frac{|f^{(n+1)}(\xi_n)|}{(n+1)!} \prod_{j=0}^n (x - x_j)$ basically Taylor series remainder.

Note error $\equiv 0$ at the nodes

Ex: let $f(x) = e^x$ and let $P_4(x)$ interpolate through $x = -1, -\frac{1}{2}, 0, \frac{1}{2}, 1$
 Find $|f(1/4) - P_4(1/4)|$ error bound

Have



Optimal Node Spacing (for polynomial interpolation)

$$|f(x) - P_{n-1}(x)| = \frac{|f^{(n)}(\xi_n)|}{n!} \prod_{j=1}^n (x - x_j) \leftarrow \text{want to minimize error by choosing nodes optimally}$$

- Suppose f and n are given
- Define $w(x) = \prod_{j=1}^n (x - x_j)$ for convenience (the "node polynomial")

Chebyshev's Theorem: The choice of nodes x_1, \dots, x_n in the interval $[-1, 1]$ that minimizes

$$\max_{-1 \leq x \leq 1} |w(x)| \text{ is } x_i = \cos \frac{(2i-1)\pi}{2n} \quad (i=1, \dots, n)$$

In which case the minimum value of $w(x)$ is $\frac{1}{2^{n-1}}$

x_i called the Chebyshev nodes

Ex: Find worse case error for approximating $f(x) = e^x$ by $P_4 \in P_4$ on $[-1, 1]$.

$$\text{error} = |f(x) - P_4(x)| \leq \frac{1}{5!} |f^{(5)}(\xi)| |w(x)|$$

$$x_i = \cos \frac{(2i-1)\pi}{2n},$$

$$\prod_{j=1}^5 (x - x_j) = (x - \cos \frac{\pi}{10})(x - \cos \frac{3\pi}{10}) \cdots (x - \cos \frac{9\pi}{10})$$

Have

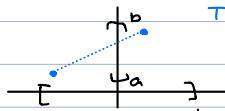
$$|w(x)| \leq \frac{1}{2^{n-1}} = \frac{1}{2^{5-1}} = \frac{1}{16}$$

$$\text{so error} \leq \frac{1}{5!} e \cdot \frac{1}{16} = 0.00142 \text{ on the interval } [-1, 1]$$

best possible upper bound obtained when Chebyshev nodes used
 any other choice of nodes yields a potentially larger error

change of Interval

$$T: [-1, 1] \rightarrow [a, b]$$



$$T(x) = ax + b \quad T(-1) = a \quad T(1) = b$$

$$\text{get } T(x) = \frac{b-a}{2}x + \frac{b+a}{2}$$

so on the interval $[a, b]$, the Chebyshev nodes are given by $x_i = \frac{b+a}{2} + \frac{b-a}{2} \cos \frac{(2i-1)\pi}{2n} \quad i=1, \dots, n$

How does the $|w(x)|$ error transform?

Let Chebyshev nodes $x_i \in [-1, 1]$

$$\text{and new Chebyshev nodes } y_i = \frac{b+a}{2} + \frac{b-a}{2} x_i \in [a, b]$$

$$\text{let } y = \frac{b+a}{2} + \frac{b-a}{2} x \in [a, b]$$

$$\text{Have } |w(y_i)| = \left| \prod_{i=1}^n \left(\frac{b-a}{2}x - x_i \right) \right| \leq \left(\frac{b-a}{2} \right)^n \left| \prod_{i=1}^n (x - x_i) \right| \leq \frac{1}{2^{n-1}} \left(\frac{b-a}{2} \right)^n \\ \leq \frac{1}{2^n} \text{ by Chebyshev's Thm}$$

So a bound on polynomial interpolation error on $[a, b]$ with Chebyshev points

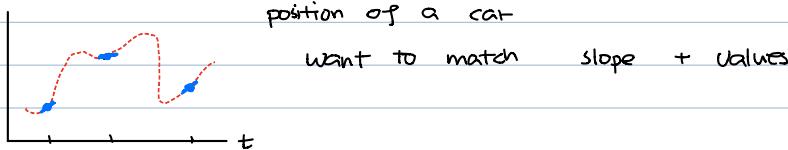
$$\text{is given by } |f(x) - P_n(x)| \leq \frac{|f^{(n)}(\xi)|}{n!} \cdot \frac{1}{2^{n-1}} \left(\frac{b-a}{2} \right)^n$$

$$\Rightarrow \|f - P_n\|_\infty \leq \frac{\|f^{(n)}\|_\infty}{n!} \cdot \frac{1}{2^{n-1}} \left(\frac{b-a}{2} \right)^n \quad \text{where } \|f\|_\infty = \max_{a \leq x \leq b} |f(x)|$$

the error band with Chebyshev nodes Σ

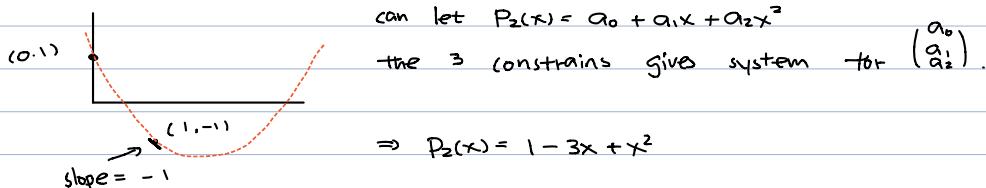
10/11/9

Hermite Interpolation



Goal: Given data points, or $f(n)$, want to find $p \in \mathbb{P}$ interpolating $\{(x_i, f(x_i))\}$ and possibly derivatives $f^{(j)}(x_i)$ at some x_i .

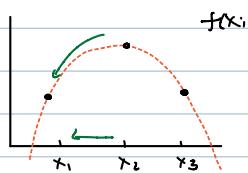
Global Hermite



consider $f(x)$ and a polynomial interpolant for (x_0, y_0) and also $f^{(j)}(x_0) \quad 0 \leq j \leq n$

$$\Rightarrow P_n(x) = \sum_{j=0}^n \frac{f^{(j)}(x_0)}{j!} (x - x_0)^j \quad \text{taylor polynomial}$$

Newton Approach (repeated nodes)



$$\begin{array}{c|ccc} & x_1 & x_2 & x_3 \\ \hline f(x_1) & & & \\ f(x_2) & & > f[x_1, x_2] & \\ f(x_3) & & > f[x_2, x_3] & > f[x_1, x_2, x_3] \end{array}$$

Newton polynomial

$$p(x) = f(x_1) + f[x_1, x_2](x - x_1) + f[x_1, x_2, x_3](x - x_1)(x - x_2)$$

what happens as $x_2 \rightarrow x_1$?

$$\lim_{x_2 \rightarrow x_1} p(x) = f(x_1) + \lim_{x_2 \rightarrow x_1} f[x_1, x_2](x - x_1) + \lim_{x_2 \rightarrow x_1} f[x_1, x_2, x_3](x - x_1)(x - x_2)$$

$$\text{let } \lim_{x_2 \rightarrow x_1} [f(x_1, x_2)] = \lim_{x_2 \rightarrow x_1} \frac{f(x_2) - f(x_1)}{x_2 - x_1} =: f'(x)$$

So repeated nodes in Newton divide differences are used to interpolate
derivative conditions:

$$\begin{array}{c|ccccc} x_1 & f(x_1) & & & \\ x_1 & f(x_1) & > f'(x_1) & = & [f(x_1, x_2)] \\ x_2 & f(x_2) & > f(x_1, x_2) & & \end{array} \quad \left/ \begin{array}{l} [f(x_2) - f(x_1)] \\ x_2 - x_1 \end{array} \right. =: [f(x_1, x_2, x_3)]$$

Ex: Find $p \in P_2$ interpolating $(1, -1), (0, 1)$ and $p'(1) = -1$

$$\begin{array}{c|ccccc} 1 & -1 & & & \\ 1 & -1 & > -1 & & \\ 0 & 1 & > \frac{1 - (-1)}{0 - 1} = -2 & & \end{array} \quad \left/ \begin{array}{l} -2 - (-1) \\ 0 - 1 \end{array} \right. = 1 \quad \text{so } P_2(x) = -1 - (x-1) + (x-1)^2 \\ = 1 - 3x + x^2$$

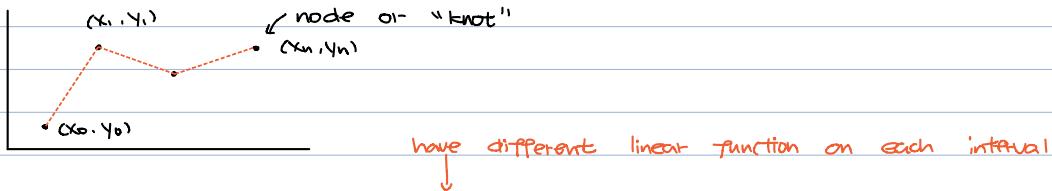
Remarks:

- ① For multiple nodes x_j of multiplicity m (m repetitions) we suppose we know $[f(x_j)], [f'(x_j)], \dots, [f^{(m-1)}(x_j)]$ [for existence/uniqueness]
- ② For higher derivatives, we define

$$[f(x_j, x_j, \dots, x_j)] := \underbrace{[f^{(m-1)}(x_j)]}_{m \text{ repetitions}} / (m-1)!$$

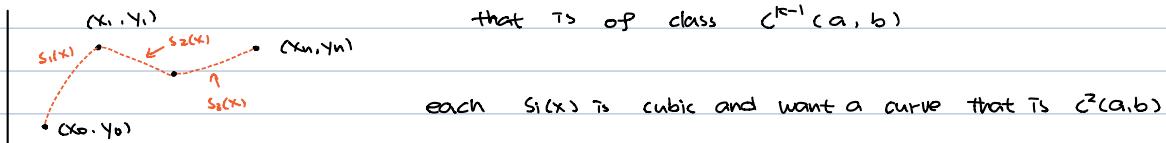
note rescaling

Piecewise Polynomial Interpolation

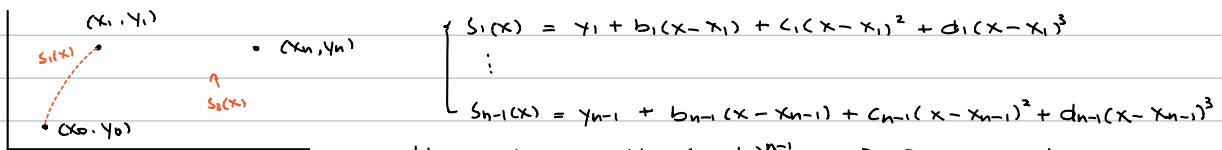


$$\text{linear piecewise: } L(x) = \begin{cases} L_0(x) & x_0 \leq x \leq x_1 \\ \vdots \\ L_{n-1}(x) & x_{n-1} \leq x \leq x_n \end{cases}$$

Spline interpolation



Cubic splines



of equations?

Each $s_i(x)$ already interpolates (x_i, y_i)

But want s_i to interpolate (x_{i+1}, y_{i+1}) too.

Get $n-1$ equations by forcing $y_{i+1} = s_i(x_{i+1}) \quad (i = 1, \dots, n-1)$

Also need $s_{i-1}'(x_i) = s_i'(x_i)$ for $i = 2, \dots, n-1$ in order to match slopes at interior pts.

\Rightarrow get $n-2$ equations.

Also want $s''_{i-1}(x_i) = s''_i(x_i)$ for $i = 2, \dots, n-1$, to match curvature at interior nodes.

\Rightarrow get $n-2$ equations.

Have $3n-5$ equations to solve for $3n-3$ unknowns.

so have 2 more unknowns than equations, can't solve for them uniquely.

Choices of two equations to close system:

(1) Natural spline: $s_i''(x_1) = s_{n-1}''(x_n) = 0$

(2) Clamped spline: Fix $s_i'(x_1)$ and $s_{n-1}'(x_n)$ arbitrarily at endpoints x_1, x_n .

(3) Curvature-adjusted: Fix $s_i''(x_1)$ and $s_{n-1}''(x_n)$ arbitrarily

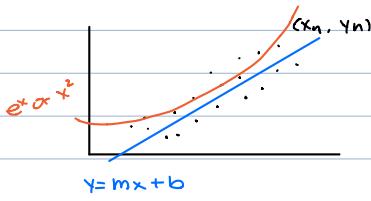
(4) not-a-knot spline:



set $s_i''(x_2) = s_{n-2}''(x_{n-1})$ and $s_{n-2}''(x_{n-1}) = s_{n-1}''(x_{n-1})$
 $\Rightarrow d_1 = d_2$ and $d_{n-2} = d_{n-1}$

10/22

Least Squares Approximation



- Try to fit data to a model (user's choice)
- Consider $y = mx + b$ (model), Force data to fit model

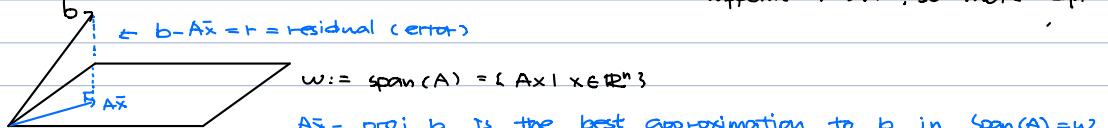
$$\begin{aligned} mx_1 + b &= y_1 \\ mx_2 + b &= y_2 \\ \vdots \\ mx_n + b &= y_n \end{aligned} \Rightarrow \begin{pmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{pmatrix} \begin{pmatrix} m \\ b \end{pmatrix}_{2 \times 2} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

Get overdetermined system to solve for m, b

General problem: Find \bar{x} that best approximates a "solution" of $Ax = b$

Geometric Approach. $Ax = b$, \bar{x} = approximate solution, $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$

Suppose $m > n$, so more eqns than unknowns.



$$w := \text{span}(A) = \{Ax \mid x \in \mathbb{R}^n\}$$

$A\bar{x} = \text{proj}_w b$ is the best approximation to b in $\text{Span}(A) = w$.

Since $r \perp w \Rightarrow \forall x \in \mathbb{R}^n, \langle Ax, r \rangle = 0$

$$\Rightarrow (Ax)^T(b - A\bar{x}) = 0 \quad (\text{recall } \langle x, y \rangle = x^T y)$$

$$\Rightarrow x^T A^T(b - A\bar{x}) = 0$$

$$\Rightarrow x^T(A^T b - A^T A\bar{x}) = 0 \quad \forall x \in \mathbb{R}^n$$

Since $\forall x \in \mathbb{R}^n, \Rightarrow A^T b - A^T A\bar{x} = 0$

$A^T A\bar{x} = A^T b$ ← normal equations that we can solve for \bar{x}

Note: (1) If $\langle x, y \rangle = 0 \quad \forall x \in \mathbb{R}^n$, then $y = 0$.

(2) Let $x = y$ then $\langle x, y \rangle = \langle y, y \rangle = \|y\|^2 = 0 \Rightarrow y = 0$

(3) In matlab, $\gg \bar{x} = (A^T A) \backslash (A^T b)$

Why called least-squares?

Residual Minimization:

$Ax = b$ want to minimize $r = b - Ax$ (residual) over all possible $x \in \mathbb{R}^n$

For least-squares, want $\|r\|_2^2 = r^T r = r_1^2 + r_2^2 + \dots + r_m^2$ minimized.

$\|r\|_2^2$ norm (hence least squares)

For rank 3, find x s.t. $\nabla_x \|r\|_2^2 = 0$ (critical points)

We'll compute this componentwise:

$(A)x_{ij} = a_{ij}$, $(x)_j = x_j$, $\bar{x} \cdot \bar{x} = \sum_i x_i^2$, $x_i x_j = x_i x_j$ (Einstein notation)

$(\nabla \Phi)_{ij} = \partial x_i \Phi$, $(Ax)_{ij} = a_{ij} x_j$

$$\nabla_x \|r\|_2^2 = 0 = (\nabla_x \|r\|_2^2)_{ij} = \partial x_i (\|r\|_2^2) = \partial x_i (r_j r_j) = r_j \partial x_i r_j + r_j \partial x_i r_j = 2 r_j \partial x_i r_j$$

To compute $\partial x_i r_j$, recall $r = b - Ax$, $r_j = b_j - a_{jk}x_k$

$$\begin{aligned}\partial x_i r_j &= \partial x_i b_j - \partial x_i a_{jk}x_k = -a_{jk} \delta_{ik} = -a_{ji} \\ &= 0 \quad \begin{cases} 1 & i=k \\ 0 & i \neq k \end{cases} = \delta_{ik}\end{aligned}$$

$$\text{So } O = 2r_j \partial x_i r_j = 2r_j (-a_{ji}) = 2(b_j - a_{jk}x_k)(-a_{ji})$$

$b - Ax$ $-A^T$

$$O = 2(-A^T)(b - Ax) \quad \text{switch order so dimensions work out correctly}$$

$$A^T A x = A^T b \quad (\text{normal equations})$$

Normal Eqn Pitfalls.

$$A = \begin{pmatrix} 1 & 1 \\ \varepsilon & 0 \\ 0 & \varepsilon \end{pmatrix}, \quad 0 < \varepsilon \leq \varepsilon_{\text{mach}} \approx 10^{-16}$$

$$A^T A = \begin{pmatrix} 1 & \varepsilon^2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ \varepsilon & 0 \end{pmatrix} = \begin{pmatrix} 1 + \varepsilon^2 & 1 \\ 1 & 1 + \varepsilon^2 \end{pmatrix}$$

stated as $\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$ in Matlab

$\det(A^T A) = 0$. not invertible, new method or higher precision

Based on QR decomposition.

$$A \in \mathbb{R}^{m \times n}, A = QR \quad Q: \text{orthogonal } \mathbb{R}^{m \times m} \text{ matrix } Q^{-1} = Q^T$$

R: upper triangular matrix in $\mathbb{R}^{m \times n}$.

$$R = \begin{pmatrix} r_{11} & \dots & r_{1n} \\ 0 & \dots & r_{2n} \\ 0 & \dots & 0 \end{pmatrix}_{m \times n}$$

Thm: QR always exists

$$\Rightarrow [Q, R] = qr(A)$$

Thm: If Q is orthogonal, then $\|Qr\|_2 = \|r\|_2$.

Any minimizer x of $\|b - Ax\|_2$ also minimizes $\|Q^T Ax - Q^T b\|_2$

$$\text{note: } \|r\|_2^2 = \|Q^T r\|_2^2 \Leftrightarrow \|Qr\|_2^2 = \|r\|_2^2$$

$$\text{with } A = QR, \quad Q^T A = Q^T Q R = R$$

Ex: let $A \in \mathbb{R}^{4 \times 2}$, $x \in \mathbb{R}^2$, $b \in \mathbb{R}^4$, let $A = QR$.

$$\min_{x \in \mathbb{R}^2} \|Ax - b\|_2 = \min_{x \in \mathbb{R}^2} \|Q^T Ax - Q^T b\|_2 = \min_{x \in \mathbb{R}^2} \left\| \begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} \right\|_2 \quad \text{where } \vec{c} = Q^T b$$

- have c_3, c_4 constant for all $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, so ignore c_3, c_4 , choose $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ to solve.

$$\begin{pmatrix} r_{11} & r_{12} \\ 0 & r_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

solution is \bar{x} and this easily solved by back-substitution

$$\text{have then } \min_{x \in \mathbb{R}^2} \|Ax - b\|_2 = \|\bar{A}\bar{x} - b\|_2 = \sqrt{c_3^2 + c_4^2}$$

least-squares by QR

let $Ax = b$, and $A = QR$, $A \in \mathbb{R}^{m \times n}$

set $\bar{R} = \text{upper } n \times n \text{ submatrix of } R$

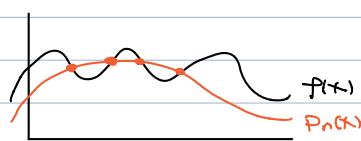
$\bar{c} = \text{upper } n \text{ entries of } \vec{c} = Q^T b$

solve $\bar{R}\bar{x} = \bar{c}$ for the least-squares solution \bar{x} .

preferred over $A^T A x = A^T b$ if done in finite precision.

10/27

least squares for functions



want "best" approximation of f out of P_n
or some other finite dimensional set
need norm to quantify "best"

Def: let V be a vector space, $\|\cdot\|: V \rightarrow \mathbb{R}$ is a norm if

- 1) $\|f\| > 0$ and $\|f\| = 0 \Leftrightarrow f = 0$
- 2) $\|\alpha f\| = |\alpha| \|f\| \quad \forall \alpha \in \mathbb{R}$
- 3) $\|f + g\| \leq \|f\| + \|g\|$

• Distance between f, g is $\|f - g\|$

• Choice of norm: (suppose f is $C[a,b]$ so everything is well-defined)

$$\|f\|_{L^\infty(a,b)} = \|f\|_\infty := \max_{a \leq x \leq b} |f(x)|$$

$$\|f\|_{L^2(a,b)} = \|f\|_2 := \left(\int_a^b |f(x)|^2 dx \right)^{1/2} \quad \text{use for } L^2 \text{ or least-squares minimization}$$

$$\|f\|_{L^1(a,b)} = \|f\|_1 := \int_a^b |f(x)| dx$$

• For 2-norm, can define inner-product $\langle f, g \rangle = \int_a^b f(x)g(x) dx$ think $\sum_{i=1}^n x_i y_i$ in \mathbb{R}^n
with $\|f\|_2 = \sqrt{\langle f, f \rangle} = \left(\int_a^b |f(x)|^2 dx \right)^{1/2}$

• Useful result: $\|f+g\|_2^2 = \|f\|_2^2 + 2\langle f, g \rangle + \|g\|_2^2$

• If $\langle f, g \rangle = 0$, we say f and g are orthogonal, and if also $\|f\|_2 = \|g\|_2 = 1$.
we say they are orthonormal.

Least-squares problem:

Goal: Let $V_n = \text{span}(g_1, \dots, g_n)$ be a finite dimensional space (think P_n)
with $\{g_i\}_{i=1}^n$ forming a basis of V_n .

Find $f_n \in V_n$ closest to f in the L^2 -sense (least-squares)
i.e. find f_n s.t $\|f - f_n\|_2^2 = \min_{v \in V_n} \|f - v\|_2^2 = \min_{v \in V_n} \left(\int_a^b |f(x) - v(x)|^2 dx \right)$

Expand $v = \sum_{i=1}^n c_i g_i \in V_n$ c_i : unknown coefficients $\{c_i\}_{i=1}^n$

g_i : basis vectors

$$\begin{aligned} \text{then } \|f - v\|_2^2 &= \|f\|_2^2 - 2\langle f, v \rangle + \|v\|_2^2 \\ &= \|f\|_2^2 - 2\langle f, \sum_{i=1}^n c_i g_i \rangle + \sum_{i=1}^n \sum_{j=1}^n c_i c_j \langle g_i, g_j \rangle \\ &= \|f\|_2^2 - 2 \sum_{i=1}^n c_i \langle f, g_i \rangle + \sum_{i=1}^n \sum_{j=1}^n c_i c_j \langle g_i, g_j \rangle \\ &= \phi(\vec{c}) \end{aligned}$$

Want to minimize over all possible \vec{c} , so set $\nabla_{\vec{c}} \phi(\vec{c}) = \vec{0}$

$$\begin{aligned} 0 &= \frac{\partial}{\partial c_k} \phi(\vec{c}) = 0 - 2 \sum_{i=1}^n \frac{\partial c_i}{\partial c_k} \langle f, g_i \rangle + \sum_{i=1}^n \sum_{j=1}^n \frac{\partial}{\partial c_k} (c_i c_j) \langle g_i, g_j \rangle \\ &\quad = \sum_{i=1}^n c_i \frac{\partial}{\partial c_k} \langle f, g_i \rangle + c_k \sum_{i=1}^n c_i \langle g_i, g_k \rangle \\ &\quad = \sum_{i=1}^n c_i \delta_{ik} + c_k \sum_{i=1}^n c_i \langle g_i, g_k \rangle \end{aligned}$$

$$\begin{aligned} &= -2 \langle f, g_k \rangle + \sum_{i=1}^n c_i \langle g_i, g_k \rangle + \sum_{i=1}^n c_i \langle g_i, g_k \rangle \\ &= -2 \langle f, g_k \rangle + 2 \sum_{i=1}^n c_i \langle g_i, g_k \rangle \end{aligned}$$

$$\Rightarrow \text{Get } \sum_{i=1}^n c_i \langle g_i, g_k \rangle = \langle f, g_k \rangle$$

This is a linear system $A\vec{c} = \vec{b}$ with $A_{ik} = \langle g_i, g_k \rangle$ and $b_k = \langle f, g_k \rangle$

note: $A_{ik} = A_{ki}$ since $\langle g_i, g_k \rangle$ is symmetric.

If $\{g_i\}_{i=1}^n$ are orthonormal, then $\langle g_i, g_k \rangle = \delta_{ik} = \begin{cases} 1 & i=k \\ 0 & \text{else} \end{cases}$

then we get $c_k = \langle f, g_k \rangle$

$\Rightarrow v = \sum_{k=1}^n c_k g_k = \sum_{k=1}^n \langle f, g_k \rangle g_k$ is our L^2 -minimizing vector.

from linear algebra, this is nothing other than orthogonal projection of f onto V_n .

Generalized Least-Squares

$$\langle f, g \rangle_w := \int_a^b f(x)g(x)w(x) dx \quad w(x): \text{weight function, need } w(x) \geq 0,$$

and

$$\|f\|_{2,w} = (\int_a^b |f(x)|^2 w(x) dx)^{1/2}$$

All previous theory applies with $\|f\|_{2,w}$ and $\langle f, g \rangle_w$.

Ex: $w(x) = \frac{1}{\sqrt{1-x^2}}$ on $[-1, 1]$



$$\min_{v \in V_n} \|f - v\|_{2,w}^2 = \min_{v \in V_n} \left(\int_{-1}^1 |f(x) - v(x)|^2 w(x) dx \right)^{1/2}$$

want $|f - v|$ small where w is large.

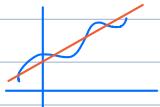
so this $w(x)$ places emphasis near $x = \pm 1$.

How to find orthonormal basis?

Example: $V_n = \text{span}(1, x) = \text{P}_1$, and $w(x) \equiv 1$ on $[0, 1]$

$$\text{so } \langle f, g \rangle_w = \int_0^1 f(x)g(x) dx$$

Note $\int_0^1 1 \cdot x dx = \frac{1}{2}x^2 \Big|_0^1 = \frac{1}{2} \neq 0$, so $\{1, x\}$ is not orthogonal.



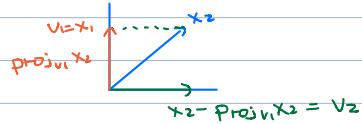
Use Gram-Schmidt to orthogonalize!

let $x_1 = 1$

$x_2 = x$

Define $v_1 = x_1 = 1$

$$v_2 = x_2 - \text{proj}_{v_1} x_2 = x_2 - \frac{\langle x_2, v_1 \rangle}{\|v_1\|^2} v_1$$



So now $v_2 = x_2 - \frac{\langle x_2, v_1 \rangle}{\|v_1\|^2} \cdot 1 = x_2 - \frac{1/2}{1} \cdot 1 = x_2 - 1/2$

$$\|v_2\|^2 = \int_0^1 |x_2 - 1/2|^2 dx = 1, \quad \langle x_2, 1 \rangle = \int_0^1 x_2 \cdot 1 dx = \frac{1}{2}x^2 \Big|_0^1 = \frac{1}{2}$$

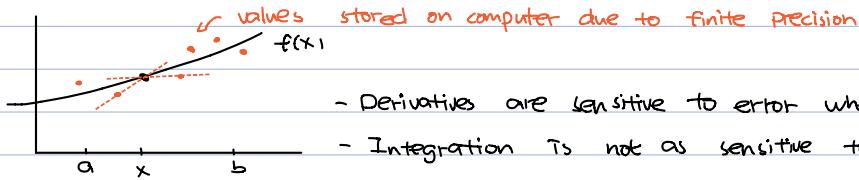
Then $v_1 = 1$, v_1 already normalized b/c $\|1\|_2^2 = 1$

and $v_2 = x - 1/2$, not normalized $\|v_2\|_2 = (\int_0^1 |x - 1/2|^2 dx)^{1/2} = \sqrt{1/12}$

then $\{v_1, v_2\} = \{1, \frac{x - 1/2}{\sqrt{1/12}}\}$ is an orthonormal basis of P_1 on $[0, 1]$ with $w(x) = 1$.

10/29.

Numerical Differentiation



- Derivatives are sensitive to error when computed numerically

- Integration is not as sensitive to error

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}, \quad f'(x) \approx \frac{f(x+h) - f(x)}{h} \quad (\text{forward 2-point difference formula})$$

$$\begin{aligned} \text{Error : } D_h f(x) &:= \frac{f(x+h) - f(x)}{h} \\ &= \frac{1}{h} [f(x) + h f'(x) + \frac{h^2}{2} f''(\xi)] - f(x) \\ &= f'(x) + \underbrace{\frac{h^2}{2} f''(\xi)}_{O(h^2) \text{ truncation error}} \end{aligned}$$

used Taylor expansion cut at the
 $O(h^2)$ term where ξ btw $x, x+h$.

$$\begin{aligned} |D_h f(x) - f'(x)| &\leq \frac{h}{2} |f''(\xi)| \leq \frac{h}{2} \|f''\|_\infty \\ \Rightarrow \|D_h f - f'\|_\infty &\leq \frac{h}{2} \|f''\|_\infty \quad \text{note as } h \rightarrow 0, D_h f \rightarrow f' \\ &\text{but in practice there is a limitation on the } h \text{ due to rounding error.} \end{aligned}$$

Backward 2-point difference

$$f(x-h) = f(x) - h f'(x) + \frac{h^2}{2} f''(\xi) \quad \text{stop Taylor here with } \xi \text{ btw } x-h, x.$$

$$f'(x) = \frac{f(x) - f(x-h)}{h} + \underbrace{\frac{h^2}{2} f''(\xi)}_{D_h f(x)} \quad O(h) \text{ error term}$$

$$\|f' - D_h f\|_\infty \leq \frac{h}{2} \|f''\|_\infty$$

Centered differencing

$$\text{Taylor} \quad \left\{ \begin{array}{l} f(x+h) = f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{3!} f'''(\xi_1) \quad \xi_1 \text{ btw } x, x+h \\ f(x-h) = f(x) - h f'(x) + \frac{h^2}{2} f''(x) - \frac{h^3}{3!} f'''(\xi_2) \quad \xi_2 \text{ btw } x-h, x \end{array} \right.$$

$$f(x+h) - f(x-h) = 2h f'(x) + \frac{h^3}{6} [f'''(\xi_1) + f'''(\xi_2)]$$

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{12} [f'''(\xi_1) + f'''(\xi_2)] \\ &= 2 f''(\xi) \text{ by IVT with } \xi \text{ btw } \xi_1 \text{ and } \xi_2 \end{aligned}$$

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f''(\xi) \quad \text{error is now } O(h^2)$$

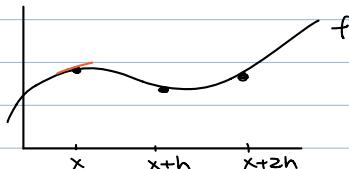
Theorem : (Generalized Intermediate Value Theorem)

let $f \in C[a, b]$, and let $x_1, \dots, x_n \in [a, b]$.

let $a_i > 0$ for $i=1, \dots, n$.

Then $\exists \xi \in [a, b]$ s.t. $(a_1 + \dots + a_n) f(\xi) = a_1 f(x_1) + \dots + a_n f(x_n)$.

Method of Undetermined Coefficients



How to get difference quotient for $f'(x)$ using only these points?

$$\begin{aligned}
 \text{write } f'(x) &= A f(x) + B f(x+h) + C f(x+2h) \quad A, B, C \text{ are unknown coeff} \\
 &= A f(x) \\
 &\quad + B [f(x) + h f'(x) + \frac{h^2}{2} f''(x) + \frac{h^3}{6} f'''(\xi_1)] \\
 &\quad + C [f(x) + 2h f'(x) + \frac{4h^2}{2} f''(x) + \frac{8h^3}{6} f'''(\xi_2)] \\
 &= (A+B+C)f(x) + (Bh+2Ch)f'(x) + (B\frac{h^2}{2}+C2h^2)f''(x) + O(h^3)
 \end{aligned}$$

$$\text{Request } A+B+C=0$$

$$h^2(\frac{B}{2}+2C)=0$$

$$h(B+2C)=1$$

solve for A, B, C Get $f'(x) \approx \frac{-3f(x) + 4f(x+h) - f(x+2h)}{2h}$

Numerical Differentiation and step size h

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \frac{1}{2} f''(\xi) =: D_h f(x) + \frac{h}{2} f''(\xi)$$

$$\Rightarrow |f'(x) - D_h f(x)| \leq \frac{h}{2} \|f''\|_\infty = \frac{Mh}{2}$$

In practice, f is stored with error: $\tilde{f}(x) = f(x) + e(x)$

computer-stored version

error $\sim E_{\text{mach}} \approx 10^{-16}$

$$\Rightarrow D_h \tilde{f}(x) - D_h f(x) = D_h e(x) \quad \text{note that } D_h \text{ is linear}$$

$$\text{so } |D_h \tilde{f}(x) - D_h f(x)| = \left| \frac{e(x+h) - e(x)}{h} \right| \leq \frac{|e(x+h) + e(x)|}{h} \leq \frac{2\varepsilon}{h}$$

tri-inequality

$$|f'(x) - D_h \tilde{f}(x)| \leq |f'(x) - D_h f(x)| + |D_h \tilde{f}(x) - D_h f(x)|$$

$\pm D_h e(x)$

$\leq \frac{Mh}{2}$

truncation error

$\leq \frac{2\varepsilon}{h}$

rounding error

$$\Rightarrow |f'(x) - D_h \tilde{f}(x)| \leq \frac{Mh}{2} + \frac{2\varepsilon}{h}$$

To minimize upper bound, set $\frac{d}{dh} (\frac{Mh}{2} + \frac{2\varepsilon}{h}) = 0 \Rightarrow h = 2\sqrt{\frac{\varepsilon}{M}}$ optimal value of h

In this case,

$$\text{error upper bound} = \frac{Mh}{2} + \frac{2\varepsilon}{h} = 2\sqrt{\varepsilon M} \approx \sqrt{\varepsilon} \approx 10^{-8}$$

Richardson Extrapolation

centered differences for $f'(x)$: $f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(3_h) \leftarrow$ error term $O(h^2)$

\downarrow $\hookrightarrow F(h)$: approximation formula

Q : the thing we want to approximate

Goal: want to find more accurate approximation.

let $F(h)$ approximate Q

then suppose $Q \approx F(h) + kh^n$ suppose $k \approx$ constant

note $\lim_{h \rightarrow 0} F(h) = Q$

Idea: If we replace $h \mapsto \frac{1}{2}h$, then error $\approx kh^n = k(\frac{1}{2}h)^n = kh^n \cdot \frac{1}{2^n}$
is reduced by a factor of 2^n .

$$\frac{Q-F(h)}{2} \approx \frac{kh^n}{2} \approx Q - F\left(\frac{h}{2}\right) \Rightarrow Q = \frac{2^n F\left(\frac{h}{2}\right) - F(h)}{2^n - 1} \quad \text{Richardson Extrapolation Formula}$$

will show the extrapolation is at least $O(h^{n+1})$

The Big-O properties

① $O(h^p) = O(ch^p)$ as $h \rightarrow 0$ for any $c = \text{constant}$, $c \neq 0$

② $O(h^p) \neq O(h^q) = O(h^r)$ as $h \rightarrow 0$ where $r = \min(p, q)$

suppose $Q = F_n(h) + kh^n + O(h^{n+1})$

\hookrightarrow denotes that $F(h)$ is $O(h^n)$

$$Q = F_n\left(\frac{h}{2}\right) + kh^n \frac{1}{2^n} + O(h^{n+1})$$

\hookrightarrow used Big-O Property ①

Extrapolation formula is

$$\frac{2^n F_n\left(\frac{h}{2}\right) - F_n(h)}{2^n - 1} = \frac{1}{2^n - 1} [2^n(Q - kh^n \frac{1}{2^n} - O(h^{n+1})) - (Q - kh^n - O(h^{n+1}))]$$

$$= Q + O(h^{n+1})$$

$$\text{so } Q = \frac{2^n F_n\left(\frac{h}{2}\right) - F_n(h)}{2^n - 1} + O(h^{n+1}) = F_{n+1}(h) + O(h^{n+1})$$

the error is now $O(h^{n+1})$ or possibly higher

Extrapolating Centered Differences for f''

$$Q = F_2(h) + kh^2$$

$$f(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{6} f'''(3_h)$$

$$\text{we get } F_3(h) = \frac{2^2 F_2\left(\frac{h}{2}\right) - F_2(h)}{2^2 - 1}$$

$$= \frac{1}{3} [4(\frac{f(x+\frac{h}{2}) - f(x-\frac{h}{2})}{2(\frac{h}{2})}) - \frac{f(x+h) - f(x-h)}{2h}]$$

$$= \frac{1}{6h} [f(x-h) - 8f(x-\frac{h}{2}) + 8f(x+\frac{h}{2}) - f(x+h)] + O(h^3)$$

\hookrightarrow our new higher order formula for $f''(x)$

Remarks: ① To get $O(h^3)$ term explicitly, need to Taylor expand.

③ In fact the error above is not $O(h^3)$, it is $O(h^4)$!

because $F_3(h) = F_3(-h)$

error term must be exactly the same for both h and $-h$, so the error can only be $O(h^4)$ for even p and so is at least $O(h^4)$

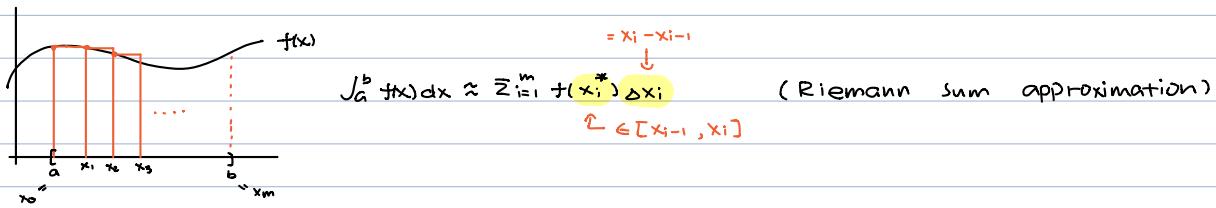
③ In general, $h \rightarrow \frac{h}{q}$ ($q \in \mathbb{N}, q \geq 2$)

we get

$$Q = \frac{q^p(F(h) - F(\frac{h}{q})))}{1 - q^p} + O(h^r)$$

with $r > p$ and where $F(h) = Q + kh^p + O(h^r)$

Quadrature (numerical integration)



In general,

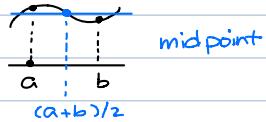
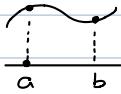
$$\int_a^b f(x) dx = \sum_{i=0}^m w_i f(x_i) + E(f) \leftarrow \text{error}$$

= Q(f) quadrature formula

where $\{w_i\}_{i=0}^m$ are weights and $\{x_i\}_{i=0}^m$ are nodes.

Newton-Cotes quadrature

- will approximate f by polynomials at equally-spaced points, and polynomials are easy to integrate.
- closed Newton-Cotes : use endpoints | open Newton-Cotes : don't use endpoints.



Lagrange basis for interpolation :

$$l_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n-1} \frac{(x-x_j)}{(x_i-x_j)}$$

(n+1) nodes $\{x_i\}_{i=0}^n$

$f(x) \approx P_n(x) = \sum_{i=0}^n f(x_i) l_i(x)$ is the n^{th} degree interpolating polynomial.

$$\text{so } \int_a^b f(x) dx = \int_a^b P_n(x) dx = \sum_{i=0}^n f(x_i) \int_a^b l_i(x) dx = Q(f)$$

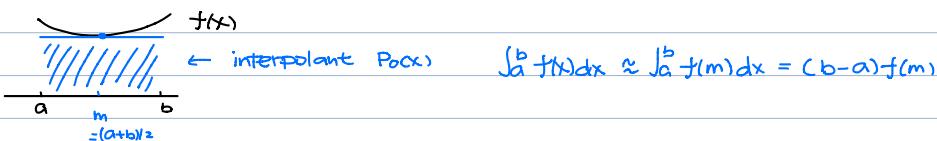
nodes ↑ weights w_i

notes: ① weights only depend on $\{x_i\}_{i=0}^n$, not on f .

② If $f \in P_n$, then $Q(f)$ integrates f exactly.

Midpoint Rule :

(open Newton-Cotes)



Error Analysis (using Taylor expansions)

$$\begin{aligned} \int_a^b f(x) dx &= \int_a^b [f(m) + (x-m)f'(m) + \frac{(x-m)^2}{2}f''(\bar{x}_1)] dx \\ &\quad = 0 \\ &= f(m)(b-a) + f'(m) \left[\frac{1}{2}x^2 - mx \right] \Big|_a^b + \frac{1}{2} \int_a^b f''(\bar{x}_2)(x-m)^2 dx \\ &\quad \text{where } \bar{x}_2 \in (a, b) \\ &= f(m)(b-a) + \frac{1}{2} \int_a^b (x-m)^2 dx \cdot f''(\bar{x}) \\ &= f(m)(b-a) + \frac{1}{24}(b-a)^3 f''(\bar{x}) \end{aligned}$$

Integral Mean-Value theorem

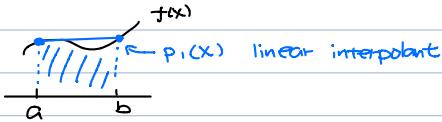
let $f, g \in [a, b]$ and $g \geq 0$. then $\exists \bar{x} \in [a, b]$ s.t. $\int_a^b f(x)g(x) dx = f(\bar{x}) \int_a^b g(x) dx$

note: thm also true if $g \leq 0$

Definition: A quadrature rule has degree of precision k if it integrates $p \in \mathbb{P}_k$ exactly.

Ex: midpoint Rule has degree of precision 1.

Trapezoid Rule: (closed Newton-Cotes)



$$\text{Have } f(x) = f(a) \frac{x-b}{a-b} + f(b) \frac{x-a}{b-a} + \frac{f''(\bar{x})}{2!} (x-a)(x-b) = \text{interpolation error } E(x)$$

= $P_1(x)$ in lagrange basis

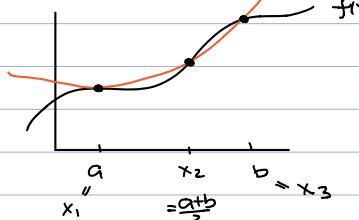
Recall: $f(x) - P_n(x) = \frac{f^{(n+1)}(\bar{x})}{(n+1)!} \prod_{j=0}^n (x-x_j)$ ← node polynomial

$$\begin{aligned} \int_a^b f(x) dx &= \int_a^b P_1(x) dx + \int_a^b E(x) dx = (b-a) \frac{f(a)+f(b)}{2} + \frac{1}{2} \int_a^b f''(\bar{x})(x-a)(x-b) dx \\ &= (b-a) \frac{f(a)+f(b)}{2} + \frac{1}{2} f''(\bar{x}) \int_a^b (x-a)(x-b) dx \\ &= -\frac{(b-a)^3}{6} \\ &= (b-a) \frac{f(a)+f(b)}{2} - \frac{1}{12} f''(\bar{x})(b-a)^3 \end{aligned}$$

degree of precision is 1

Simpson's Rule

$P_2(x)$ quadratic interpolant



write $f(x) = P_2(x) + E(x)$

↳ use Lagrange basis

$$\int_a^b f(x) dx = \int_a^b P_2(x) dx + \int_a^b E(x) dx$$

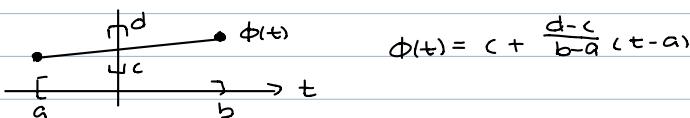
$$\int_a^b f(x) dx = \frac{b-a}{6} \left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b) \right] - \frac{(b-a)^5}{90 \cdot 32} f^{(4)}(\bar{x})$$

the degree of precision is 3.

Change of interval:

Suppose $\int_a^b f(t) dt \approx \sum_{i=0}^n w_i f(t_i) = Q(f)$

and we want to approximate $\int_c^d f(x) dx$ use a linear map from $[a, b] \rightarrow [c, d]$



$$\int_c^d f(x) dx = \int_a^b f(\phi(t)) \cdot \frac{d-c}{b-a} dt \approx \frac{d-c}{b-a} \sum_{i=0}^n w_i f(\phi(t_i))$$

↳ let $\kappa = \phi(t)$ $dx = \phi'(t)dt$

Example: $\int_0^1 f(t) dt \approx \frac{1}{2} (f(0) + f(1))$ convert to $[c, d]$

Have $\phi(t) = c + (d-c)t$ ($a=0, b=1$)

$$\begin{aligned} \int_c^d f(x) dx &= (d-c) \sum_{i=0}^1 w_i f(\phi(t_i)) \\ w_i &= \frac{1}{2} \quad \text{↳ use } \phi(0)=c \\ \text{for } i=0, 1 &\quad \phi(1)=d \end{aligned}$$

$$= \frac{d-c}{2} [f(c) + f(d)]$$

Method of Undetermined coefficients

$\int_a^b f(x) dx \approx Q(f) := w_0 f(x_0) + w_1 f(x_1)$ pick two points x_0, x_1 as the nodes
choose weights so that $Q(f)$ integrates polynomials exactly for highest possible degree.

Example: $\int_0^1 f(x) dx \approx w_0 f(0) + w_1 f(1)$ choose endpoints for example.

Want quadrature to be exact for $f(x)=1$ and $f(x)=x$

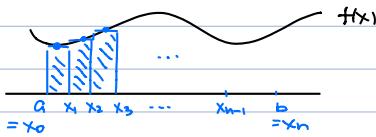
$$\Rightarrow \int_0^1 1 dx = 1 = w_0 \cdot 0 + w_1 \cdot 1 = Q(1)$$

$$\int_0^1 x dx = \frac{1}{2} = w_0 \cdot 0 + w_1 \cdot 1$$

$$\Rightarrow w_1 = \frac{1}{2}, w_0 = \frac{1}{2}$$

$$\text{Get } \int_0^1 f(x) dx \approx \frac{1}{2} [f(0) + f(1)] \quad (\text{trapezoid rule})$$

Composite Midpoint Rule



$n+1$ nodes $\{x_i\}$ n subintervals

let $h = \frac{b-a}{n}$ = constant subinterval width

let $m_i = \frac{x_i + x_{i+1}}{2}$ (mid point)

$$\int_{x_i}^{x_{i+1}} f(x) dx = h f(m_i) + \frac{1}{24} h^3 f'''(\bar{x}_i) \quad \text{where } \bar{x}_i \in [x_i, x_{i+1}]$$

so

$$\int_a^b f(x) dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x) dx = h \sum_{i=0}^{n-1} f(m_i) + \frac{1}{24} h^3 \sum_{i=0}^{n-1} f'''(\bar{x}_i)$$

By generalized I.V.T., $\sum_{i=0}^{n-1} f'(x_i) = n f''(\bar{x})$ since have n terms in the sum will all coefficients equal to 1.

Error terms is:

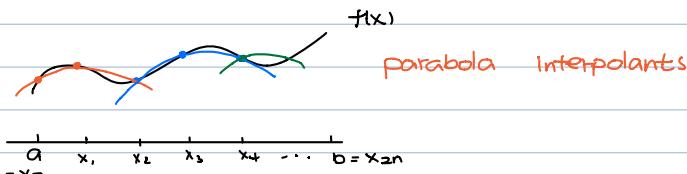
$$\frac{1}{24} h^3 n f'''(\bar{x}) = \frac{1}{24} h^2 (b-a) f'''(\bar{x})$$

$$\text{So } \int_a^b f(x) dx = h \sum_{i=0}^{n-1} f(m_i) + \frac{1}{24} h^2 (b-a) f'''(\bar{x})$$

Remarks: ① order of method is $O(h^2)$

② This integrates $p \in P_1$ exactly

Composite Simpson's Rule



- need odd # of nodes

- need even # of subintervals

- $2n+1$ nodes, $2n$ subintervals.

$$h = \frac{b-a}{2n} = \text{constant width}$$

Apply Simpson's Rule on $[x_{2i}, x_{2i+2}]$ for $i=0, \dots, n-1$:

$$\int_{x_{2i}}^{x_{2i+2}} f(x) dx = \frac{2h}{6} [f(x_{2i}) + 4 f(x_{2i+1}) + f(x_{2i+2})] - \frac{(2h)^5}{90} f^{(4)}(\bar{x}_i) \quad \text{with } \bar{x}_i \in [x_{2i}, x_{2i+2}]$$

Now sum:

$$\int_a^b f(x) dx = \sum_{i=0}^{n-1} \int_{x_{2i}}^{x_{2i+2}} f(x) dx$$

$$= \frac{h}{3} \sum_{i=0}^{n-1} [f(x_{2i}) + 4 f(x_{2i+1}) + f(x_{2i+2})] - \frac{h^5}{90} \sum_{i=0}^{n-1} f^{(4)}(\bar{x}_i) \\ = n f^{(4)}(\bar{x}) \text{ by generalized I.V.T}$$

$$= \frac{h}{3} [f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1})] - \frac{1}{90} h^5 n f^{(4)}(\bar{x})$$

$$= h^4 hn = h^4 \frac{b-a}{2}$$

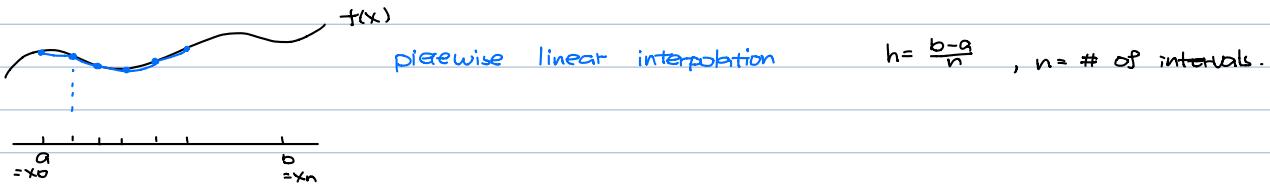
$$= \frac{h}{3} [f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_{2i}) + 4 \sum_{i=0}^{n-1} f(x_{2i+1})] - \frac{1}{180} h^4 (b-a) f^{(4)}(\bar{x})$$

Remarks: ① This method is $O(h^4)$

② Method integrates $P \in P_3$ exactly.

③ Some authors use n subintervals with n even. here we used $2n$ subintervals.

Composite Trapezoid Rule



$$\int_a^b f(x) dx = \frac{h}{2} [f(a) + f(b) + 2 \sum_{i=1}^{n-1} f(x_i)] - \frac{h^2}{12} (b-a) f''(\bar{x})$$

$$x_i = a + ih, i=0, 1, \dots, n.$$

11/17

Weighted Quadrature

quadrature weights

$$\int_a^b f(x) w(x) dx \approx \sum_{k=1}^n w_k f(x_k) = Q(f)$$

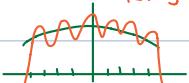
in general can consider integrals with fixed weights.

- Can we choose the nodes and weights so that $Q(f)$ is exact for the highest degree possible polynomial.

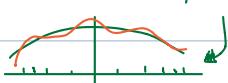
- We have now $2n$ degrees of freedom and will be able to integrate $P \in P_{2n-1}$ exactly. This is achieved by Gaussian quadrature.

Remark:

Runge phenomenon.



Chebyshev points



Example: $\int_{-1}^1 f(x) dx \approx Q(f) = w_1 f(x_1) + w_2 f(x_2)$ think $\int_{-1}^1 f(x) \cdot 1 dx$. weight $w(x) \equiv 1$

Request $Q(f)$ is exact for $\{1, x, x^2, x^3\}$: 4 unknowns, 4 equations.

$$w_1 + w_2 = \int_{-1}^1 1 = 2$$

$$w_1 x_1 + w_2 x_2 = \int_{-1}^1 x dx = 0 \Rightarrow w_1 = w_2 = 1$$

$$w_1 x_1^2 + w_2 x_2^2 = \int_{-1}^1 x^2 dx = 2/3 \quad x_1, x_2 = \pm \sqrt{1/3}$$

$$w_1 x_1^3 + w_2 x_2^3 = \int_{-1}^1 x^3 dx = 0$$

$$\text{So } Q(f) = f(\sqrt{1/3}) + f(-\sqrt{1/3})$$

Orthogonal Polynomials

$\{P_n\} \subset P_n$ is orthogonal with respect to the inner-product $\langle \cdot, \cdot \rangle_w$ on (a, b)

If $m \neq n$, have $\langle P_m, P_n \rangle_w = \int_a^b P_m(x) P_n(x) w(x) dx = 0$

Theorem: If $\{P_0, P_1, \dots, P_n\}$ is an orthogonal on $[a, b]$, and $\deg(P_i) = i$,

then P_i has exactly i distinct roots on (a, b) .

Example:

$$\text{let } \langle f, g \rangle_w = \int_{-1}^1 f(x) \cdot g(x) \cdot 1 dx, \text{ weight } w(x) \equiv 1$$

An orthogonal set in \mathbb{P}_2 is $\{1, x, \frac{3}{2}(x^2 - \frac{1}{3})\}$ on $[-1, 1]$ with $w(x) \equiv 1$.

These are called the first 3 Legendre polynomials.

Note: $\begin{cases} 0 \text{ has 0 roots} \\ x \text{ has 1 root on } (-1, 1) \\ x^2 - \frac{1}{3} \text{ has two roots } \pm \sqrt{\frac{1}{3}} \text{ on } (-1, 1) \end{cases}$

Gauss-Legendre quadrature (weight $w(x) \equiv 1$ and interval $[-1, 1]$)

Fix n . Let $\{x_i\}_{i=1}^n$ be the n roots of the n^{th} Legendre polynomial.

Interpolate f at x_i :

$$f(x) \approx \sum_{i=1}^n l_i(x) f(x_i) \quad \text{where} \quad l_i(x) = \prod_{j=1, j \neq i}^n \frac{(x - x_j)}{(x_i - x_j)}$$

Lagrange basis for interpolation

$$\int_{-1}^1 f(x) dx \approx \sum_{i=1}^n f(x_i) \underbrace{\int_{-1}^1 l_i(x) dx}_{= w_i} = w_i \text{ weights for quadrature}$$

$$= \sum_{i=1}^n w_i f(x_i)$$

$$= Q(f)$$



Theorem: $Q(f)$ obtained is exact for all $p \in \mathbb{P}_{2n-1}$

Example: $I = \int_{-1}^1 e^{-x^2} \cdot 1 dx, w(x) \equiv 1$

know degree 2 Legendre polynomial is $P_2(x) = \frac{3}{2}(x^2 - \frac{1}{3})$

By previous theorem, optimal nodes are the roots of $P_2(x)$ (for 2-node quadrature)
 $\Rightarrow x_1, x_2 = \pm \sqrt{\frac{1}{3}}$

So $Q(f) = w_1 f(\sqrt{\frac{1}{3}}) + w_2 f(-\sqrt{\frac{1}{3}})$ and use undetermined coefficients.

Want $Q(f)$ to evaluate $\int_{-1}^1 f(x) dx$ exactly for $f(x) = 1, x$

\rightarrow two unknowns w_1, w_2 , need two functions)

$$\text{so } \int_{-1}^1 1 dx = 2 = w_1 + w_2$$

$$\int_{-1}^1 x dx = 0 = w_1 \sqrt{\frac{1}{3}} + w_2 (-\sqrt{\frac{1}{3}})$$

$$\Rightarrow w_1 = w_2 = 1$$

$Q(f) = f(\sqrt{\frac{1}{3}}) + f(-\sqrt{\frac{1}{3}})$ and this is exact for any $p \in \mathbb{P}_{2n-1} = \mathbb{P}_3$ by Gaussian quad. theorem.
 $\hookrightarrow n=2$

$$\text{So } \int_{-1}^1 e^{-x^2} dx \approx e^{-(\sqrt{\frac{1}{3}})^2} + e^{-(-\sqrt{\frac{1}{3}})^2} \approx 1.6929.. \quad \text{Exact integral is } 1.71124...$$

Change of interval:

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

$$= g(t)$$

$$= \int_{-1}^1 g(t) dt \cdot \frac{b-a}{2}$$

then can apply Gauss-Legendre quadrature to g .

Quick way to obtain some common orthogonal polynomials.

without resorting to Gram-Schmidt.

Case: Gauss-Legendre for $\int_{-1}^1 f(x) \cdot 1 dx, w(x) \equiv 1$ on $[-1, 1]$

Recursion relation to generate orthogonal polynomials on $[-1, 1]$ with $w(x) \equiv 1$

$$\begin{cases} P_{-1}(x) = 0 \\ P_0(x) = 1 \end{cases}$$

:

$$L_{(j+1)} P_{j+1} = (2j+1) \times P_j - j P_{j-1}$$

$$j=0 : P_0 = x P_0 = x \cdot 1 = x$$

$$j=1 : 2P_2 = 3 \times P_1 - P_0 = 3x^2 - 1$$

$$P_2(x) = \frac{1}{2}(3x^2 - 1)$$

so $1, x, \frac{1}{2}(3x^2 - 1)$ are the first 3 Legendre polynomials.

In general, the Gaussian quadrature optimal node theorem remains valid for other orthogonal polynomials on different intervals or with different weights.

$$\int_a^b f(x) w(x) dx \approx \sum_{k=1}^n A_k f(x_k)$$

L fixed weight

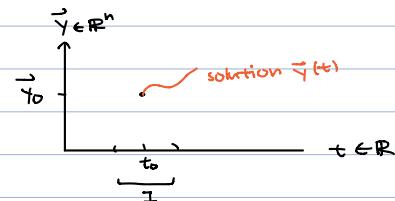
and to find x_k , would use orthogonal polynomials on $[a, b]$ w.r.t. t $w(x)$.

11/19 Numerical Method for ODE

Initial Value Problem (IVP)

- 1st order system $\begin{cases} \vec{y}'(t) = \vec{f}(t, y(t)) & \forall t \in I \subseteq \mathbb{R} \\ \vec{y}(t_0) = \vec{y}_0 \end{cases}$

$$\vec{y}: I \rightarrow \mathbb{R}^n, \quad \vec{f}: D \subset \mathbb{R}^{n+1} \rightarrow \mathbb{R}^n \quad (n \geq 1)$$



- Higher order ODE can always be reduced to a 1st order system.

$$\text{Example: } y'' = (y')^2 + y$$

$$\text{let } y_1 = y \Rightarrow y_1' = y' = y_2$$

$$y_2 = y' \quad y_2' = y'' = (y')^2 + y = (y_2)^2 + y_1$$

$$\text{So get the 1st order system for } \vec{y}(t) = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}(t), \quad \frac{d}{dt} \vec{y}(t) = \vec{f}(t, \vec{y}(t)) = \begin{pmatrix} y_2 \\ (y_2)^2 + y_1 \end{pmatrix}(t)$$

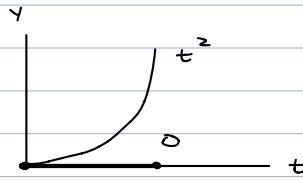
Potential Issues.

1. non-uniqueness of solutions!

$$\begin{cases} y'(t) = 2\sqrt{|y(t)|} \\ y(0) = 0 \end{cases}$$

$$\exists \text{ two solutions: } y \equiv 0 \quad (t > 0)$$

$$y(t) = t^2$$



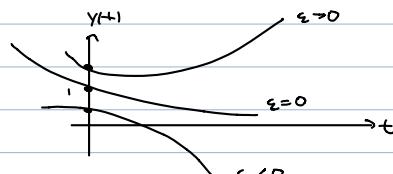
2. non-existence: $|y'(t)| = -1$ no solutions to this ODE.

3. Instability: $\begin{cases} y'(t) = 100y - 10e^{-t} \\ y(0) = 1 + \varepsilon \end{cases}$

$$\text{solution is } y(t) = e^{-t} + \varepsilon e^{100t}$$

- If we make a small ε error in $y(0)$, our new solution no longer decays toward 0!

- We say the ODE is unstable with respect to perturbations in the IC.



Existence & Uniqueness of IVP

Recall \vec{f} is Lipschitz w.r.t. \vec{y} if $\exists k \in \mathbb{R}$ s.t. $\forall t \in I$

$$\|\vec{f}(t, \vec{y}_1) - \vec{f}(t, \vec{y}_2)\| \leq k \|\vec{y}_1 - \vec{y}_2\|$$

(in 1d replace $\|\cdot\|$ with $| \cdot |$)

Remark: If $\vec{f} \in C^1$, then \vec{f} is Lipschitz provided $\left\| \frac{\partial \vec{f}}{\partial \vec{y}} \right\| \leq k$

where $\frac{\partial \vec{f}}{\partial \vec{y}}$ is the Jacobian matrix $[\frac{\partial \vec{f}}{\partial \vec{y}}]_{ij} = \frac{\partial f_j}{\partial y_i}$

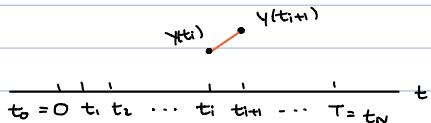
Theorem: let \vec{f} be Lipschitz in \vec{Y} near the point (t_0, \vec{y}_0) and suppose \vec{f} is continuous.

w.r.t t , then $\exists \epsilon > 0$ and $\exists \vec{y}(t)$ s.t $\begin{cases} \vec{y}(t) = \vec{f}(t, \vec{y}(t)) \\ \vec{y}(t_0) = \vec{y}_0 \end{cases}$
for $t_0 - \epsilon < t < t_0 + \epsilon$

One-step method

$$\begin{cases} \dot{y} = f(t, y(t)) \\ y(0) = y_0 \end{cases} \quad t \in [0, T]$$

$\begin{cases} N+1 \text{ nodes (mesh points)} \\ N \text{ intervals} \end{cases}$



only one step is used to approximate $y(t_{i+1})$

in general mesh not uniform: $h_i = t_i - t_{i-1}$ (step size) $i = 1, 2, \dots, N$

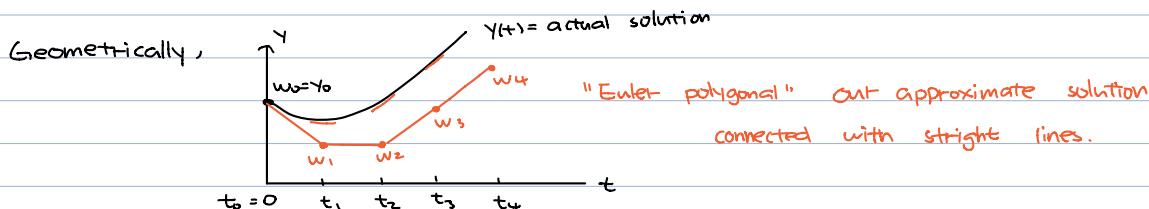
Forward Euler Method:

suppose $h_i = h = \text{constant}$ $\forall i$, let $\{w_i\}_{i=0}^N$ where $w_i \approx \underline{y(t_i)}$ approximate value exact solution at $t=t_i$

For small h , $y(t) \approx \frac{y(t+h) - y(t)}{h}$

$$y(t+h) \approx y(t) + \frac{h y'(t)}{h} = y(t) + h f(t, y(t)) = f$$

$$\text{so } \underbrace{y(t+h)}_{\approx w_{i+1}} \approx \underbrace{y(t)}_{\approx w_i} + h f(t_i, w_i) \Rightarrow \begin{cases} w_{i+1} = w_i + h f(t_i, w_i) \\ w_0 = y_0 \end{cases} \quad \forall i=0, \dots, N-1$$



this method is explicit, w_{i+1} is given explicitly in terms of other values (w_i)

Numerical Stability



Consider $\begin{cases} \dot{y} = \gamma y \\ y(0) = y_0 \end{cases}$ ("Dahlquist test equation")

solution to $y(t) = y_0 e^{\gamma t}$ and for $\gamma < 0$ expect decay as $t \rightarrow \infty$

so request that $|w_i| \rightarrow 0$ as $i \rightarrow \infty$

check Forward Euler:

$$w_{i+1} = w_i + h f(t_i, w_i) = w_i + h \gamma w_i = (1 + h\gamma) w_i = (1 + h\gamma)^2 w_{i-1} = \dots = (1 + h\gamma)^{i+1} w_0 = \alpha^{i+1} w_0$$

magnification factor $\alpha^{i+1} = (1 + h\gamma)$

For $|w_i| \rightarrow 0$ as $i \rightarrow \infty$ need $\alpha^{i+1} \rightarrow 0$ as $i \rightarrow \infty$ iff $|\alpha| < 1$

so need $|1 + h\gamma| < 1$ for stability

$$-1 < 1 + h\gamma < 1 \Rightarrow -\frac{2}{\gamma} > h > 0 \quad (\gamma < 0)$$

\Rightarrow stability for forward Euler requires $0 < h < -\frac{2}{\gamma}$ if $h > -\frac{2}{\gamma}$ blow up.

if $h = -\frac{2}{\gamma}$

Note: ① If $\gamma = -10$, $y' = -10y$, $0 < h < 1/5$ for stability.

② If need size restriction on h , we call the method conditionally stable.

11/24

Euler Methods : Part II.

$$\text{Forward Euler} \quad \left\{ \begin{array}{l} w_{i+1} = w_i + h f(t_i, w_i) \\ w_0 = y_0 \end{array} \right. \quad \text{for solving} \quad \left\{ \begin{array}{l} y' = f(t, y) \\ y(0) = y_0 \end{array} \right. \quad \text{with} \quad w_i \approx y(t_i)$$

this method requires small h for numerical stability: $\left| \begin{array}{l} y' = \gamma y \\ y(0) = y_0 \end{array} \right. \quad (\text{test equation}, \gamma < 0)$

$$\text{need } |1 + \gamma h| < 1 \quad \text{for stability} \Rightarrow 0 < h < -\frac{2}{\gamma}$$

Definition: let $\gamma < 0$ and $y' = \gamma y$ be the "test equation" (complex numbers)

The linear stability domain of a numerical method is the set $D := \{z \in \mathbb{C} \mid 1 + z \in D \subseteq \mathbb{C}$
s.t. the numerical solution of $y' = \gamma y$ decays.

Ex. For Forward Euler, $D = \{z \in \mathbb{C} \mid |1+z| < 1 \quad (|z - (-1)| < 1)\}$



Definition: If $D = \{z \in \mathbb{C} \mid \text{Re}(z) < 0\}$ we call the method absolutely stable, or A-stable.
the point is A-stable method have no restriction on h .

$$\text{Forward Euler for systems.} \quad \left\{ \begin{array}{l} \vec{y}'(t) = \vec{f}(t, \vec{y}(t)) \\ \vec{y}(0) = \vec{y}_0 \end{array} \right. \quad \text{denotes } n^{\text{th}} \text{ components of a vector}$$

can apply Forward Euler componentwise $\left\{ \begin{array}{l} w_i^{(n)} = w_i^{(n)} + h f_i^{(n)}(t_i, w_i^{(n)}) \\ w_0^{(n)} = y_0^{(n)} \end{array} \right. \quad i \text{ denotes the } i^{\text{th}}$
 \downarrow
 $w_i^{(n)}$ iteration.

Error Analysis and Convergence.

- rounding error: due to computer finite precision. - very difficult to analyze in ODEs.
- truncation error: due to numerical discretization $y' = f(t, y)$ Euler method.
 - local truncation error: Error made in one-step of the method.
 - global truncation error: the accumulation of all local errors on a time interval $[0, T]$

Local error for Forward Euler. $w_{i+1} = w_i + h f(t_i, w_i)$

$$\begin{aligned} \text{replace } w_i \text{ with } y(t_i) \text{ to get } y(t_{i+1}) &\stackrel{?}{=} y(t_i) + h f(t_i, y(t_i)) \\ \text{have } t_{i+1} = t_i + h \rightarrow y(t_i + h) - [y(t_i) + h f(t_i, y(t_i))] &= [y(t_i) + h \cancel{y'(t_i)} + \frac{h^2}{2} y''(t)] - [y(t_i) + h f(t_i, y(t_i))] \\ &\stackrel{?}{=} y'(t_i) = f(t_i, y(t_i)) \\ &= \frac{h^2}{2} y''(t) = O(h^2) \quad \text{local truncation error for forward Euler} \end{aligned}$$

$$\text{Global error for Forward Euler} \quad \left\{ \begin{array}{l} y' = f(t, y(t)) \\ y(a) = y_a \end{array} \right. \quad t \in (a, b)$$

$$\text{Theorem: } g_i := |w_i - y(t_i)| \leq \frac{M h}{2L} (e^{L(t_i - a)} - 1)$$

where $|y''(t)| \leq M$ on $[a, b]$ and L is a Lipschitz constant for f in the y -variable.

Remarks: (1) As $h \rightarrow 0$, have $g_i \rightarrow 0 \forall i$, so forward Euler converges.

(2) This is an $O(h)$ method (1st order method)

(3) Note local error is $O(h^2)$ but global error is $O(h)$ let $h = \frac{t_{\text{final}} - t_0}{N}$

Have N steps and each time get $O(h^2)$ error:

$$\text{global error} \approx O(h^2)N = O(h^2) \frac{t_{\text{final}} - t_0}{h} = O(h) (t_{\text{final}} - t_0)$$

Backward Euler Method

$$\begin{cases} y' = f(t, y) \\ y(0) = y_0 \end{cases}$$

$$y(t) \approx \frac{y(t) - y(t-h)}{h} \Rightarrow y(t) \approx y(t-h) + h y'(t) = y(t-h) + h f(t, y(t))$$

bkwd difference quotient

let $w_i \approx y(t_i)$ to get $\begin{cases} w_{i+1} = w_i + h f(t_{i+1}, w_{i+1}) & (\text{replaced } t \text{ with } t_{i+1}) \\ w_0 = y_0 \end{cases}$

- this is an implicit method, need to solve $F(z) := z - [w_i + h f(t_{i+1}, z)] = 0$ for z .

then update $w_i \rightarrow z = w_{i+1}$

- can use bisection, Newton, fzero ...
preferred: slow

Local truncation error.

$$\begin{aligned} y(t_{i+1}) - [y(t_i) + h f(t_{i+1}, y(t_{i+1}))] &= y'(t_{i+1}) \text{ from the ODE} \\ &= y(t_{i+1}) - \underline{y(t_i)} - h y'(t_{i+1}) \\ &= y(t_{i+1}) - \underline{y(t_{i+1}-h)} - h y'(t_{i+1}) \\ &= y(t_{i+1}) - [y(t_{i+1}) - h y'(t_{i+1}) + \frac{h^2}{2} y''(s)] - h y'(t_{i+1}) \\ &= -\frac{h^2}{2} y''(s) \end{aligned}$$

- local error is $O(h^2) \Rightarrow$ global error is $O(h)$, 1st order method

Stability of backward Euler

test equation $\begin{cases} y' = \gamma y = f(t, y) & \text{with } \gamma < 1 \\ y(0) = 1 \end{cases}$

Apply bkwd Euler : $\begin{cases} w_{i+1} = w_i + h f(t_{i+1}, w_{i+1}) = w_i + h \gamma w_{i+1} \\ w_0 = 1 \end{cases}$
 $\Rightarrow w_{i+1} = \frac{1}{1-\gamma h} w_i = \dots = \left(\frac{1}{1-\gamma h}\right)^i w_0$

require $|\frac{1}{1-\gamma h}| < 1$ for stability, but this holds for any $h > 0$ (since $\gamma < 0$ too)

so no restriction on h . Backward Euler is a A-stable method.

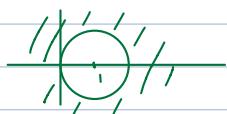
Remarks : ① Bkwd Euler is A-stable, forward Euler is not!

② Both methods are $O(h)$, 1st order methods.

③ For bkwd Euler, gain the stability in exchange for solving nonlinear equation.

④ Forward Euler is explicit (easy to get w_{i+1})

⑤ Stability region for bkwd Euler :



$$z = \gamma h \in \mathbb{C}$$

\Leftarrow the set $|1 - z| > 1$

contains entire left plane \Rightarrow A-stable.

More ODE methods.

$$\text{ODE} \left\{ \begin{array}{l} y' = f(t, y(t)) \\ y(0) = y_0 \end{array} \right.$$

- so far we seen - forward Euler (explicit, not A-stable)
 - Backward Euler (implicit, A-stable)

Centered differencing

$$y'(t) = \frac{y(t+h) - y(t-h)}{2h} + O(h^2) \quad \text{2nd order accurate}$$

- Fwd/Bkwd Euler, would have $O(h)$ error

plug into ODE: $[y(t+h) - y(t-h)]/2h \approx f(t, y(t))$

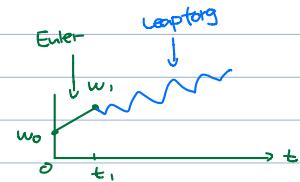
$$y(t+h) \approx y(t-h) + 2h f(t, y(t))$$

$$\begin{array}{ccccccc} & w_{i-1} & \longrightarrow & w_i & \longrightarrow & w_{i+1} & \\ t_{i-h} & \longleftarrow & t_i & \longleftarrow & t_{i+h} & \longleftarrow & t_{i+2h} \end{array}$$

$$w_i \approx y(t_i), h = \text{constant stepsize in time}$$

$$\text{Leapfrog method} \rightarrow \left\{ \begin{array}{l} w_{i+1} = w_{i-1} + 2h f(t_i, w_i) \\ w_0 = y_0, w_1 = y_1 \end{array} \right.$$

usually obtain a second point (in order to get Leapfrog started)
 by using a single forward Euler step (or something else)



Remarks: (1) Method is explicit, two-step. (multistep methods)

(2) Local truncation error is $O(h^3)$

\Rightarrow Global error is $O(h^2)$ so the method is 2nd order.

Stability: test eqn $\left\{ \begin{array}{l} y' = \gamma y = f(t, y) \\ y(0) = y_0 \end{array} \right.$ ($\gamma < 0$) actual solution is $y(t) = y_0 e^{\gamma t}$
 which decays.

$$\begin{aligned} w_{i+1} &= w_{i-1} + 2h f(t_i, w_i) \\ &= w_{i-1} + 2h (\gamma w_i) \end{aligned}$$

to solve, assume $w_i = t^i$ for some $t \in \mathbb{R}$ (or \mathbb{C})

$$t^{i+1} = t^{i-1} + 2h \gamma t^i$$

$t = t^{-1} + 2h \gamma$ rewrite use quadratic formula

$$t = t_{\pm} = -h \gamma \pm \sqrt{1 + (h \gamma)^2}$$

Get the general solution is $w_i = C_1 t_+^i + C_2 t_-^i$ took a linear combination,
 need to check that w_i decays as $i \rightarrow \infty$ C_1, C_2 arbitrary constant.

$$\text{Have } t_+ = \underbrace{-h \gamma}_{>0} + \underbrace{\sqrt{1 + (h \gamma)^2}}_{>1} \rightarrow 0+1=1, t_+^i \rightarrow \infty \text{ as } i \rightarrow \infty \text{ so } w_i \text{ blows up!}$$

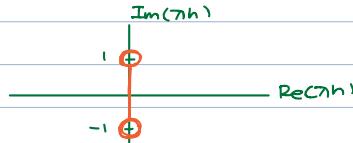
so Leapfrog is unstable for all $h > 0$

when solving $y' = f(t, y)$

Remark: (4) For systems, $\vec{y}' = A \vec{y}$.

when A has complex eigenvalues γ .

Leapfrog can be useful.



stability region for Leapfrog

Method based on Quadrature

$$\left\{ \begin{array}{l} y' = f(t, y) \\ y(0) = y_0 \end{array} \right.$$

$$h = t_{i+1} - t_i = \text{constant}$$

$$\int_{t_i}^{t_{i+1}} y'(t) dt = \int_{t_i}^{t_{i+1}} f(t, y(t)) dt$$

$y(t_{i+1}) - y(t_i)$ can apply any quadrature rule.
 (by FTC)

here we use Trapezoid Rule: $\frac{1}{2}h [f(t_i, y(t_i)) + f(t_{i+1}, y(t_{i+1}))]$

$$y(t_{i+1}) \approx y(t_i) + \frac{1}{2}h [f(t_i, y(t_i)) + f(t_{i+1}, y(t_{i+1}))]$$

$$\Rightarrow \text{Trapezoid method} \quad w_{i+1} = w_i + \frac{1}{2}h [f(t_i, w_i) + f(t_{i+1}, w_{i+1})]$$

$$w_0 = y_0$$

Remarks : (1) Method is implicit, one-step

(2) Local truncation error is $O(h^3)$, global error is $O(h^2)$

Stability : $y' = \gamma y, \gamma < 0$

$$w_{i+1} = \frac{1 + \frac{\gamma h}{2}}{1 - \frac{\gamma h}{2}} w_i \quad \text{want } \left| \frac{1 + \frac{\gamma h}{2}}{1 - \frac{\gamma h}{2}} \right| < 1, \text{ holds if } h > 0.$$

the method is A-stable. no h restriction

Explicit Trapezoid (Heun's method, Improved Euler)

$$\begin{cases} w_{i+1} = w_i + \frac{1}{2}h [-f(t_i, w_i) + f(t_{i+1}, w_i + hf(t_i, w_i))] \\ w_0 = y_0 \end{cases}$$

replaced w_{i+1} in Trapezoid w/ a Forward Euler-

Remarks : (1) Method is $O(h^2)$ in global error, explicit, one-step.

(2) For stability, need $0 < h < -\frac{2}{\gamma}$

from checking the test equation $y' = \gamma y (\gamma < 0)$

\Rightarrow conditionally stable

(3) This is a "predictor - corrector" method.

Euler predicts w_{i+1} , then trapezoid corrects it to a new w_{i+1} .

12/13

ode45 : 4th order method, 5th order method

↳ "Runge-Kutta methods"

$e_i \approx 1 \frac{y_{i+1} - y_i}{h}$ ↳ obtained from 5th order method

↳ obtained from 4th order method

If e_i is small enough, ode45 accepts the time stepsize and continues onto next y-value.

ode15s is a stiff ODE solver

stiff ODE

$$y' = \gamma y (\gamma < 0)$$

For forward Euler, need $0 < h < -\frac{2}{\gamma}$ for stability.

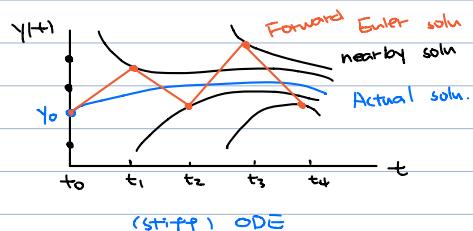
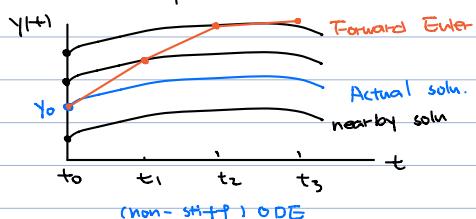
$\ll 1$ if γ large and negative

Definition : An ODE is stiff if explicit ODE solver require very small time steps for stability.

- For $y' = y^2 - y^3 ; y(0) = 10^{-6}$, ode45 requires 1,000,000 time steps

while ode15s only requires ≈ 100 timesteps.

Graphically : $y' = f(t, y)$



note: $y' = \tau y$ ($\tau < 0$) implicit methods typically are A-stable methods.

no restriction on h



Runge-Kutta methods

1-stage RK method (explicit, 1-step, 1st order)

$w_{i+1} = w_i + a s_1$, we assume the iteration has this form (ansatz)

$s_1 = h f(t_i, w_i)$ "stage" of the RK method

Q: Can we find $a \in \mathbb{R}$ so the method has the best possible (highest order) truncation error?

Have $w_{i+1} = w_i + a h f(t_i, w_i)$

$$y(t_{i+1}) - [y(t_i) + ah f(t_i, y(t_i))] = O(h^2)$$

$$y(t_i) + h y'(t_i) + O(h^2) - y(t_i) - ah y'(t_i) = O(h^2)$$

cancel if $a = 1 \rightarrow$

so $a = 1$ and we get

$$w_{i+1} = w_i + h f(t_i, w_i) \quad \text{Forward Euler.}$$

2-stage Runge-Kutta method (explicit, 1-step method, 2nd order)

Assumed $w_{i+1} = w_i + a s_1 + b s_2$

form of the $s_1 = h f(t_i, w_i)$

iteration scheme $s_2 = h f(t_i + \alpha h, w_i + \beta s_1)$

$$y(t_{i+1}) = y(t_i) + h y'(t_i) + O(h^2) \quad \text{Taylor exp.}$$

$$f(t_i, y(t_i)) = y'(t_i) \quad \text{from the ODE}$$

$$t_{i+1} = t_i + h, h \text{ constant}$$

- unknowns are a, b, α, β

- can we choose these so the truncation error order is as high as possible?

- Yes. Get $O(h^3)$ truncation error.

- skipping the details we compute

$$y(t_{i+1}) - [y(t_i) + a s_1 + b s_2]$$

and Taylor expand a lot to find

$$1 - a - b = 0$$

$$\frac{1}{2} - \alpha b = 0$$

$$\frac{1}{2} - b \beta = 0$$

- ∞ solutions since 3 eqns, 4 unknowns.

A natural choice is $a = b = \frac{1}{2}, \alpha = \beta = 1$ (for symmetry)

Get $w_{i+1} = w_i + \frac{1}{2}(s_1 + s_2)$

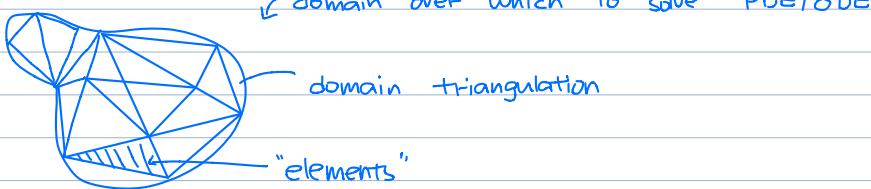
$$s_1 = h f(t_i, w_i)$$

$$s_2 = h f(t_i + h, w_i + s_1)$$

this is explicit trapezoid or improved Euler.

Crash Course in Finite Elements

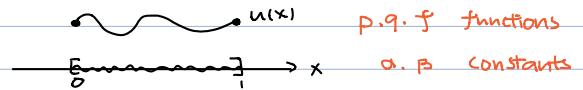
2D example



- FEM for ODE:
- ① form the variational (weak) formulation of ODE
 - ② project the infinite-dimensional problem to a finite-dim subspace.
 - ③ From the finite-dimensional problem obtain a system of equations that can be solved to get an approximate solution

ODE :
$$\begin{cases} (-pu')' + qu = f & \text{on } (0,1) \subset \mathbb{R} \\ u(0) = \alpha, \quad (pu')'(1) = \beta \end{cases}$$

(boundary-value problem)



Let's convert to weak formulation:

Multiply ODE by $v \in C^\infty([0,1])$ and integrate by parts

v is arbitrary called a test function

$$\int_0^1 (-pu')' v + \int_0^1 quv = \int_0^1 fv$$

$$\begin{aligned} \int_0^1 (-pu')' v &= -pu'v|_0^1 - \int_0^1 (-pu')v' \quad (\text{integrated by parts}) \\ &= -\underbrace{p(1)u'(1)v(1)}_{=\beta} + p(0)u'(0)v(0) + \int_0^1 (pu')v' dx. \end{aligned}$$

↑ not known but can choose $v(0)=0$
from the boundary condition
so this term disappears.

$$\Rightarrow \int_0^1 pu'v' + quv = \int_0^1 fv dx + \beta v(1)$$

this is the variational (weak) form of the ODE

- variational since v can vary, it's arbitrary.

- weak since we can accept weaker conditions on u , no longer need u'' to exist.

New problem asks to find $u \in V$ (V : some function space)

s.t. the weak form holds for all $v \in C^\infty([0,1])$. s.t. $v(0)=0$

Remarks:

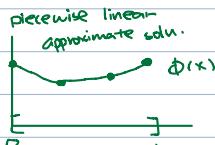
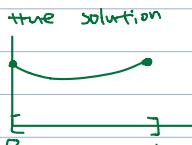
- ① If u is a classical solution, it is also a weak solution.

And if u is a weak solution, s.t. also $u \in C^2([0,1])$, then it is also a classical solution that satisfies the original ODE.

② For classical ODE, need u to be twice-differentiable.

For the weak form, one derivative is enough.

Even more so, we can allow u' to be undefined at a finite number of points. (since we're integrating and pointwise values don't affect the integral)

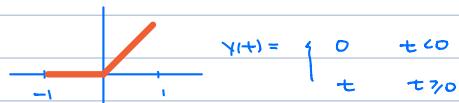


note derivative is defined everywhere except a finite # of points.

this function $\phi(x)$ won't satisfy the classical form but will satisfy the weak form.

Example: $y'(t) = H(t)$ on $(-1, 1)$, Heaviside function $H(t) = \begin{cases} 1 & t \geq 0 \\ 0 & t < 0 \end{cases}$

- Suppose $y(t)$ is continuous
- We expect our solution to be



$$y(t) = \begin{cases} 0 & t < 0 \\ t & t \geq 0 \end{cases}$$

this is the solution satisfying $y(-1) = 0$.

but $y(t)$ not differentiable at the origin.

- So $y(t)$ does not satisfy the classical form of the ODE but will instead satisfy the integral (weak) form.

check it: weak form:

let v be a smooth test function s.t. $v(-1) = v(1) = 0$.

Multiply ODE by v and integrate by parts.

$$\int_{-1}^1 y'v = \int_{-1}^1 Hv$$

$$\int_{-1}^1 y'v = yv|_{-1}^1 - \int_{-1}^1 yv'$$

$$= y(1)v(1) - y(-1)v(-1) - \int_{-1}^1 yv'$$

$$-\int_{-1}^1 yv' = \int_{-1}^1 Hv$$

to check $y(t) = \begin{cases} 0 & t < 0 \\ t & t \geq 0 \end{cases}$ satisfies $-\int_{-1}^1 yv' = \int_{-1}^1 Hv$ the weak form.

$$\text{LHS} = -\int_{-1}^1 yv' = -\int_0^1 v'(t) dt = -[v(t)]_0^1 + \int_0^1 v(t) dt = \int_0^1 v(t) dt = \int_{-1}^1 H(t)v(t) dt = \text{RHS}$$

FEM.

$$\begin{aligned} \text{ODE: } & \begin{cases} -(pu')' + qu = f & \text{on } (0, 1) \\ u(0) = \alpha, \quad (pu')(1) = \beta \end{cases} \end{aligned}$$

$$\begin{aligned} \text{weak form: } & \underbrace{\int_0^1 pu'v' + quv}_{=: B(u, v)} = \underbrace{\int_0^1 fv dx + \beta u(1)}_{=: L(v)} \end{aligned}$$

$B(u, v)$ since LHS is bilinear (linear in u, v separately)

$L(v)$ since RHS is linear in v

Problem:

find $u \in V$ s.t. $B(u, v) = L(v)$ for all $v \in C^\infty$ s.t. $v(0) = 0$

V is infinite dimensional function space (think continuous functions or polynomials)

Idea for getting to FEM:

Replace V with V_h a finite-dimensional subspace, and try to solve the following problem:

Find $u \in V_h$ s.t. $B(u, v) = L(v)$ holds for all $v \in V_h$ s.t. $v(0) = 0$.

the choice of V_h dictates the type of approximation for u .

$V_h = P_n \rightarrow$ polynomial u

$V_h = \text{trig functions.} \rightarrow$ trig solution u (spectral method)

For finite elements, there is another choice for V_h