Introduction to Computing

Taylor Expansion

$f(x \pm h) \approx f(x) \pm h \cdot f'(x) + \frac{h^2}{2} \cdot f''(x) \pm \frac{h^3}{6} \cdot f'''(x) \cdots$

x = point of approximation.
The approximation becomes more accurate as more terms are included.

1 Errors

Scalar quantity u is approximated by v

Absolute error: $|u - v|$

Relative error: $\frac{|u-v|}{|u|}$ relative usually more meaningful. Not dependent on units.

Useful: $\sqrt{a^2+b^2} = s\sqrt{\frac{a^2}{s^2} + \frac{b^2}{s^2}}$, $s = max(|a|, |b|)$
Formulas can be written in different ways where the error is smaller like in the following example where it initially has a cancellation error. Bsp: if $log(\sqrt{x+1} + \sqrt{x})$ dann binom. erw mit
$\sqrt{x+1} - \sqrt{x}$ so dass $(a-b)(a+b) = a^2 - b^2$

1.1 Approximaiton errors

Discretization error: in numerical differentiation, integration and interpolation
Convergence error: in iterative methods.

bot have smooth structuers and can be assessed using numerical analysis (Taylor expansion)

1.2 Roundoff error

Due to finite-precision representaiton of real numbers in computers.

Random in nature, can be assumed as smaller than approximation errors.
Accumulation of roundoff errors:
Linear accumulation: roundoff increases linearly with number of operations (unavoidable)
Exponential accumulation: unacceptable, roundoff error increases exponentially with number of operations. Leading to an unstable algorithm.
Over-/Underflow: a number is too large or small to be represented with given precision.

1.3 Error magnification:

- if x very different from y, $x + y$ has a large absolute error
- if $x \approx y$, $x - y$ has a large realtive error $\rightarrow$ cancellation error
- if $|y| \ll 1$, x/y may have large absolute and relative errors
- if $|y| \gg 1$, xy may have large absolute and relative errors

1.4 rounding unit

The rounding unit (u) is the maximum relative error introduced when a real number is rounded to the nearest representable value in a floating-point system. It is defined as:

$$u = \frac{\beta^{1-p}}{2},$$

where: - β: Base of the floating-point system (e.g., $\beta = 2$ for binary), - p: Number of significant digits (precision).
In IEEE 754 double precision, $u = 2^{-53} \approx 1.11 \times 10^{-16}$.

1.5 Error magnitude and comparison of errors

$$\text{err}_i(h) = O(h^q) \iff \exists c > 0 \text{ s.t. } \text{err}_i(h) \leq ch^q$$

For all h small enough
- To visualize errors and compare can plot on log scale:

$$log(err_i(h)) = log(ch^q) = log(c) + qlog(h)$$

Plot log(h) against log(err) where start on bottom is log(c) and the slope is q. Use to compare different errors.

1.6 Discretization error example procedure

Example: Given an approximation for $f'(x_0)$. Task is to calculate the discretization error in comparison to the true $f'(x_0)$ value (Absolute error). Use Taylor expansion to derive the approximation. Once derived the part that is left is the discretization error. Only keep the part with the smallest h^q, as the bigger ones will be approx 0 in comparison.

$$\text{err}(h) = \left| \frac{f(x_0+h) - f(x_0)}{h} - f'(x_0) \right| \approx$$

$$\frac{h}{2}|f''(x_0)| + \frac{h^2}{6}\cancel{|f'''(x_0)|} + \dots$$

Can cancel from h^2 on as these parts are extremely small compared to h
err(h) = $O(h)$ -> err goes to 0 linearly as h->0

2 Linear Systems

2.1 Determinante

$$\det(A) = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j})$$

Where: - a_{1j} is an element in the first row. - A_{1j} is the submatrix obtained by removing the first row and j-th column.

2.2 Inverse of a Matrix

$A^{-1}A = AA^{-1} = \mathbb{I}$ if $\lambda = \text{EW}(A)$, then $\text{EW}(A^{-1}) = \frac{1}{\lambda}$

2×2 Matrices: If $\det(A) \neq 0$.

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}, \quad A^{-1} = \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

Adjoint Method:

$$A^{-1} = \frac{1}{\det(A)} \cdot \text{adj}(A), \quad \text{if } \det(A) \neq 0,$$

where adj(A) is the transpose of the cofactor matrix.

2.3 Adjoint Matrix

- For each element a_{ij} in A, compute the cofactor:

$$C_{ij} = (-1)^{i+j} \det(A_{ij}),$$

where A_{ij} is the submatrix obtained by removing the i-th row and j-th column.

2.4 Basics

- **Linear system:** $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, $b \in \mathbb{R}^m$, $Ax = b$
- **Case $m = n$:** unique solution

$$\det A \neq 0$$
$$\Leftrightarrow \exists A^{-1} \text{ s.t. } AA^{-1} = A^{-1}A = I$$
$$\Leftrightarrow \text{columns of } A \text{ lin. indep.}$$
$$\Leftrightarrow \text{rows of } A \text{ lin. indep.}$$
$$\Leftrightarrow A \text{ non-singular}$$

- **Case $m > n$:** in general, no solution
- **Case $m < n$:** infinite solutions

2.5 Vector Norms

Norm of a vector: $||\cdot|| : \mathbb{R}^n \rightarrow \mathbb{R}$ with:

1. $||x|| \geq 0 \quad ||x|| = 0 \Leftrightarrow x = 0$
2. $||\alpha x|| = |\alpha| ||x||$ with $\alpha \in \mathbb{R}$
3. $||x + y|| \leq ||x|| + ||y||$ with $x, y \in \mathbb{R}^n$

- **l_2-norm** $||x||_2 = \sqrt{x^T x} = (\sum_i^n x_i^2)^{1/2}$
- **l_1-norm** $||x||_1 = \sum_i^n |x_i|$
- **l_∞-norm** $||x||_\infty = \max_i |x_i|$

Similarity transformation: for B square, S square non-singular, B and $S^{-1}BS$ have the same eigenvalues.

2.6 Norm of a Matrix

$A \in \mathbb{R}^{m \times n}$

Norm $||\cdot|| : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$ s.t.

1. $||A|| \geq 0, ||A|| = 0 \rightleftharpoons A = 0$
2. $||\alpha A|| = |\alpha| \cdot ||A||, \quad \alpha \in \mathbb{R}$
3. $||A + B|| \leq ||A|| + ||B||, \quad A, B \in \mathbb{R}^{m \times n}$
4. $||AB|| \leq ||A|| \cdot ||B||, \quad A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times l}$

For $A \in \mathbb{R}^{m \times n}$, vector induced norm $||\cdot||$,

$$||A|| = \max_{x \neq 0} \frac{||Ax||}{||x||} = \max_{||x||=1} ||Ax||$$

- **2-norm** $||A||_2 = \sqrt{|\lambda_{max}|(A^T A)}$
root of largest eigenvalue of $A^T \cdot A$
if A is symmetric solution is the largest eigenvalue of A.
- **1-norm** $||A||_1 = \max_{j=1\dots n} \sum_{i=1}^m |A_{ij}|$ maximum column sum
- **∞-norm** $||A||_\infty = \max_{i=1\dots m} \sum_{j=1}^n |A_{ij}|$ maximum row sum

Norm preservation: Q orthogonal

$$||Q\vec{v}|| = ||\vec{v}|| \quad ||QB|| = ||B|| \quad ||CQ|| = ||C||$$

2.7 Condition number

How sensitive is the solution x to a change in y?

$\kappa_i(A) := ||A||_i \cdot ||A^{-1}||_i$(different norms applicable)

upper bound of relative error estimation using condition number:

$$\frac{||x - \tilde{x}||}{||x||} = \frac{||\delta x||}{||x||} \leq \kappa(A) \frac{||\delta y||}{||y||}$$

$\rightarrow$ relative error in input is less or equal to relative error in output multiplied by condition number.
Identity Matrix (I): A diagonal matrix with all diagonal entries equal to 1.

$$\kappa(I) = 1$$

Diagonal Matrix (D): A matrix where all non-diagonal entries are zero.

$$\kappa(D) = \frac{\max |d_i|}{\min |d_i|}$$

Orthogonal Matrix (Q): A matrix satisfying $Q^T Q = I$.

$$\kappa(Q) = 1$$

Symmetric Positive Definite (SPD) Matrix (A): A symmetric matrix with all eigenvalues positive.
Only for l2-norm ???????? Check when this can be used!!!

$$\kappa(A) = \frac{\lambda_{\text{max}}}{\lambda_{\text{min}}}$$

Singular Matrix (A): A non-invertible matrix (determinant is zero).

$$\kappa(A) = \infty$$

- if $\kappa(A)$ is small, an error in y is not greatly amplified
- if $\kappa(A)$ is large, an error in y may translate into a much larger error in x
- $1 \leq \kappa(A) < \infty$ ($1 \Rightarrow A$ orthogonal, $\infty \Rightarrow A$ singular)
- $\kappa_1(A) \rightarrow$ 1-norm; $\kappa_2(A) \rightarrow$ 2-norm; $\kappa_\infty(A) \rightarrow \infty$-norm
- if $\kappa(A) >> 1$, A is ill-conditioned

2.8 SVD

$\forall A \in \mathbb{R}^{m \times n}$: $V \in \mathbb{R}^{n \times n}$ orthogonal

$\boxed{A = U\Sigma V^T}$ $U \in \mathbb{R}^{m \times m}$ orthogonal

$$\underset{m \times n}{\Sigma} = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 \dots 0 \\ 0 & \sigma_2 & \dots & & \\ \dots & & \dots & & \\ 0 & \dots & & \sigma_r & \\ 0 & \dots & & & 0 \dots 0 \\ \dots & & & & \dots 0 \end{bmatrix} \quad r = rank(A) = n$$

1. Eigenvalues of $AA^* \Rightarrow \lambda_1 \geq .. \geq \lambda_r$, define $\sigma_i = \sqrt{\lambda_i}$

2. Calculate eigenvalues of A^*A and find the corresponding eigenspaces $(u_1, \dots, u_n)$,

3. perform steps 2 and 3 on those vectors (see QR)
$V = \begin{bmatrix} | & & | \\ e_1 & \dots & e_m \\ | & & | \end{bmatrix}$, 5. $w_i := \frac{1}{\sigma_i} A v_i \Rightarrow U = \begin{bmatrix} | & & | \\ w_1 & \dots & w_m \\ | & & | \end{bmatrix}$

2.9 QR decomposition

$\forall A \in \mathbb{R}^{m \times n}, m \geq n, \exists Q \in \mathbb{R}^{m \times n}$ orthogonal and $R_0 \in \mathbb{R}^{m \times n}$ upper triangular s.t.

$$A = QR \quad \textbf{with} \quad R = \begin{bmatrix} R_0 \\ \dots \\ 0 \\ (m-n) \times n \end{bmatrix}$$

3 Data fitting

3.1 Interpolation

$$P_n(x) = v(x) = \sum_{j=0}^{n} c_j \phi_j(x) = c_0\phi_0(x) + \dots + c_n\phi_n(x)$$

Basic assumptions:

- ϕ_j are linearly indep.
- # of basic functions = # of data
- data are distinct (if $i \neq j$, $x_i \neq x_j$)

Interpolaiton condition: $v(x_i) = y_i$ or $P_n(x_i) = f(x_i)$

$$\underbrace{\begin{bmatrix} \phi_0(x_0) & \phi_1(x_0) & \dots & \phi_n(x_0) \\ \phi_0(x_1) & \phi_1(x_1) & \dots & \phi_n(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(x_n) & \phi_1(x_n) & \dots & \phi_n(x_n) \end{bmatrix}}_{A} \cdot \underbrace{\begin{bmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{bmatrix}}_{c} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}}_{y}$$

$A \cdot c = y \Rightarrow c = A^{-1}y$ $A \equiv$ Vandermonde matrix

n = length(x) - 1 : Order of P_n

3.1.1 Polynomial interpolation

```
p = polyfit(x_data, y_data, n);
y_pred = polyval(p, x_pred);
```

1) Polyfit finds coefficients $c_n, ..., c_0$. Polyval finds y_{pred} given p and query points vector x

$\phi_j(x)$ **are polynomials** $\Rightarrow v(x) = p_n(x)$, p_n is unique

Monomial form: $\phi_j(x) = x^j, \quad j = 0, ..., n$
simplest form

Lagrange form: $\phi_j(x) = L_j(x)$ with

$$L_j(x_i) = \begin{cases} 0 \text{ if } i \neq j \\ 1 \text{ if } i = j \end{cases}$$

$$L_j(x) = \Pi_{\substack{i=0 \\ i \neq j}}^{n} \frac{x - x_i}{x_j - x_i} \quad j = 0, 1, ..., n$$

- $A = I \quad \Rightarrow \quad c = y \quad c_i = y_i$
- Best form to use for computers

Most stable form

Newton polynomials: $\phi_j(x) = \Pi_{i=0}^{j-1}(x - x_i)$

- $\phi_0(x) = 1$,
  $\phi_1(x) = (x - x_0)$,
  $\phi_2(x) = (x - x_0)(x - X_1)$
- A lower triangular matrix

Datapoints can be added one by one
Error for global polynomial interpolation:

$$\max_{x \in [a,b]} |f(x) - p_n(x)| \leq \frac{1}{(n+1)!} \max_{t \in [a,b]} \left| f^{(n+1)}(t) \right| \cdot \max_{s \in [a,b]} \prod_{i=0}^{n} |s - x_i|$$

Chebyshev Points minimizes last part of polynominal error equation. Problem: Polynomials tend to oscillate more and more for higher degrees.
$\Rightarrow$ Global polynomial interp. is not a good idea for large n. Exception: Chebyshev points.

$$\hat{x}_i = cos\left[\frac{2i+1}{2(n+1)}\pi\right], \quad i = 0, ..., n$$

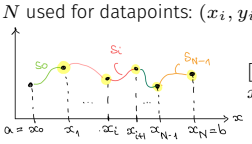
On an interval $[a, b] \neq [-1, 1]$:

$$x_i = \frac{b-a}{2}\hat{x}_i + \frac{b+a}{2}$$

3.1.2 Piecewise polynomial interpolation

```
%interpolating cublic spline
p = csape(x_data, y_data, 'variational');
y_pred = ppval(p, x_pred);
```

$n \equiv$ polynomial degree;
N used for datapoints: $(x_i, y_i), \ i = 0, 1, ..., N$



$[a, b]$ with
$x_0 = a < ... < x_N = b$

- divide $[a, b]$ in N subintervals,
  $x_0, ..., x_N$ breakpoints $\equiv$ datapoints
- local polynomial interpolation on each subinterval $[x_i, x_{i+1}], i = 0, ..., N-1$, call it $s_i(x)$
- stitch $s_i(x)$ together to form a global interpolating curve $v(x)$
- C^0:Meet at point, C^1:Continuety first deriv., C^2: First and second deriv.

Piecewise linear interpolation:

$$s_i(x) = \underbrace{a_i + b_i}_{2N \text{ unknowns}} \cdot (x - x_i) \quad x_i \leq x \leq x_{i+1}$$

$2N$ equations: $s_i(x_i) = y_i, \quad s_i(x_{i+1}) = y_{i+1}$

$$\Rightarrow \boxed{s_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i)} \quad x_i \underset{i=0,...,N-1}{\leq} x \leq x_{i+1}$$

Error bound:

$$|f(x) - v(x)| \leq \frac{h^2}{8} \max_{t \in [a,b]} |f''(t)|$$

$h = \max_{1 \leq i \leq N}(x_i - x_{i+1})$

- simple
- max & min at data points
- approximation is local
- non-smooth (only C^0)

Piecewise cubic interpolation/cubic spline:

$$s_i(x) = \underbrace{a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3}_{4 \text{ unknowns/subinterval } (4N \text{ total})}$$

$n = len(x) - 1$ -> number of subintervals
Setup n formulas with their respective intervals from $x_0 to x_1$ and so on
$s_i'(x) = b_i + 2c_i(x - x_i) + 3d_i(x - x_i)^2$
$s_i''(x) = 2c_i + 6d_i(x - x_i)$

- Interpolation + C^0-continuity ($2N$ eq.):
  - $s_i(x_i) = y_i \qquad i = 0, ..., N-1$
  - $s_i(x_{i+1}) = y_{i+1} \qquad i = 0, ..., N-1$
- C^1-continuity:
  - $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \quad i = 0, ..., N-2$
- C^2-continuity:
  - $s_i''(x_{i+1}) = s_{i+1}''(x_{i+1}) \quad i = 0, ..., N-2$
- Two last conditions. Options:
  - Free boundary: $s_0''(x_0) = s_{N-1}''(x_N) = 0$
  - Clamped bound.: $s_0'(x_0) = s_{N-1}'(x_N) = 0$

Global coupling: One yi changes, all coef. change, but not too much because of strict diag. dominance -> almost local
Error:

$$\max_{x \in [a,b]} |f(x) - v(x)| \leq \max_{t \in [a,b]} \left| f^{(IV)}(t) \right| h^4$$

$h = \max_{0 \leq i \leq N-1} h_i$

Catmull-Rom splines:

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

- C^1-continuity:
  - $s_i'(x_{i+1}) = s_{i+1}'(x_{i+1}) \quad i = 0, ..., N-2$
- $s_i'(x_{i+1}) = \frac{y_{i+2} - y_i}{h_i + h_{i+1}}$ (slope at a point between x_i and x_{i+1} is the same as the linear line between the two points)

3.2 Regression

3.2.1 Linear least squares

```
p = lsfit(x_data, y_data, n);
y_pred = polyval(flip(p), x_pred);
```

Residual vector $\underbrace{r}_{m \times 1} = \underbrace{y}_{m \times 1} - Ax$

Goal: $||r||^2$ to be minimal (least squares)

$$||r||_2^2 = ||y - Ax||_2^2 \to min$$

$$x^* = \underset{x}{\text{argmin}} ||y - Ax||^2 \qquad \boxed{A^T A x^* = A^T y}$$

$$x^* = (A^T A)^{-1} A^T y = A^+ y \qquad A^+ \equiv \underset{\text{pseudo inverse}}{\text{Moore-Penrose}}$$

- Symmetric: $(A^T A)^T = A^T A$
- Positive definite: $p^T(A^T A)p = ||Ap||^2 > 0$
  $\Rightarrow$ eigenvalues real and positive
  $\Rightarrow A^T A$ not singular and thus invertible
- Special case $m = n$: $A^+ = A^{-1}$

Generalized condition number: $A \in \mathbb{R}^{m \times n}$

$$\frac{||\delta x||}{||x||} \leq ||A|| \cdot ||A^+|| \frac{||\delta y||}{||y||} = \kappa(A) \frac{||\delta y||}{||y||}$$

$$\boxed{\kappa(A^T A) = \frac{\lambda_1}{\lambda_n} = (\frac{\sigma_1}{\sigma_n})^2 = \kappa^2(A)}$$

$\Rightarrow A^T A$ is much worse conditioned than A

3.2.2 Basis Matrix A in Least Squares Problems

The basis matrix A is constructed using the formula:

$$A_{ij} = \Phi_j(x_i),$$

where:

- $\Phi_j(x)$: Basis functions that linearly combine to approximate $y(x)$,
- x_i: Data points at which $\Phi_j(x)$ is evaluated,
- i: Index of data points, j: Index of basis functions.

Applicability: This formula applies when:

- The model is linear in the coefficients c_j: $y(x) \approx \sum_{j=1}^{n} c_j \Phi_j(x)$,
- Basis functions $\Phi_j(x)$ and data points x_i are explicitly defined.

Exceptions: For non-linear models (e.g., $y = e^{c_1 x}$) or systems with multiple variables, the construction of A may vary and may involve iterative methods or custom evaluations of the basis functions.

3.2.3 Least squares with SVD

```
[U, sigma, Vt] = svd(A);
```

$x = A^t y = V\Sigma^t U^t y \quad$ or $\quad (\Sigma V^T)x = U^T y$
$\Rightarrow \quad \kappa(\Sigma V^T) = \frac{\sigma_{max}}{\sigma_{min}} = \kappa(A)$

- Well conditioned
- SVD is expensive

3.2.4 Least squares with QR

1. QR decomposition of A (i.e. find Q, R_0) -> appendix
2. $d = Q^T y$
3. first n rows of $d \to d_0$
4. solve $R_0 x = d_0$ by back substitution

- $\kappa(R_0) = \kappa(A)$
- less expensive than SVD

3.2.5 Probabilistic view of lin. least squares

Noisy data: $y_i = \Gamma(t_i) + e_i \quad e_i \equiv$ noise
Residual components: $r_i(\vec{x}) = y_i - v(t_i; \vec{x}) = e_i + \underbrace{\Gamma(t_i) - v(t_i; \vec{x})}_{\text{approximation error}}$
We assume: $e_i = N(0, \sigma^2)$

$$P(\vec{x}) \underset{\text{MLE}}{\to} max \quad \rightleftharpoons \quad \|\vec{r}(\vec{x})\|^2 \underset{\text{Least squares}}{\to} min$$

4 Num. differentiation

$f(x_0 \pm h) \approx f(x_0) \pm h f'(x_0) + \frac{h^2}{2} f''(x_0) \pm \frac{h^3}{6} f'''(x_0) + ... + \frac{h^k}{k!} f^{(k)}(x_0)$ for small h

4.1 First derivative

- Forward difference (1st-order accurate): $\xi \in [x_0, x_0 + h]$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} - \frac{h}{2} f''(\zeta)$$

- Backward difference (1st-order accurate): $\xi \in [x_0 - h, x_0]$

$$f'(x_0) = \frac{f(x_0) - f(x_0 - h)}{h} + \frac{h}{2} f''(\zeta)$$

- Centered 3-point formula (2nd-order acc.):
  Derive by subtracting and rearranging $f(x_0 + h)$ and $f(x_0 - h)$.
  Will get $f'''(\xi_1) + f'''(\xi_1) = 2f'''(\xi)$

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} - \frac{h^2}{6} f'''(\zeta)$$

- Centered 5-point formula (4th-order acc.): You can derive the 5 point formula using Richardson extrapolation. Insert this here!!!

$$f'(x_0) = \frac{f(x_0 - 2h) - 8f(x_0 - h) + 8f(x_0 + h) - f(x_0 + 2h)}{12h} + \frac{h^4}{30} f^v(\zeta)$$

4.2 Second derivative

Derive by adding and rearranging $f(x_0 + h)$ and $f(x_0 - h)$.

$$f''(x_0) = \frac{f(x_0 - h) - 2f(x_0) + f(x_0 + h)}{h^2} - \frac{h^2}{12} f^{iv}(\zeta)$$

4.3 Errors

E.g. for forward difference formula

Discretization err.: $|f'(x_0) - D_h| = -\frac{h}{2}|f''(\zeta)|$

Roundoff error: $|D_h - \bar{D}_h| \leq 2\frac{\epsilon}{h}$

$$|D_h - \bar{D}_h| \leq \left| \frac{f(x_0 + h) - \bar{f}(x_0 + h)}{h} \right| + \left| \frac{f(x_0) - \bar{f}(x_0)}{h} \right|$$

Total error: $|f'(x_0) - \bar{D}_h| \leq \frac{h}{2}|f''(\zeta)| - 2\frac{\epsilon}{h}$

Rules of thumb:

- Roundoff error dominates up to 10^{-5} and discretization error dominates from ther on up. Optimal in discretiz. area.
- For a formula of order q, use $h \gg \eta^{\frac{1}{q+1}}$
- Use double precision!

4.4 Noisy data

- If a function is noisy, numerical differentiation magnifies it by $1/h$
- Computation of the k^{th} derivative will magnify noise by $1/h^k$

$\Rightarrow$ **always denoise** data befor deifferentiation!!!

5 Num. Integration

$$I = \int_a^b f(x)dx \approx \sum_{j=0}^{n} w_j f(x_j)$$

Precision or degree of accuracy of a quadrature rule = largest degree of polynomial integrated exactly

How to find error for quad. rule:

- 1. taylor of quadrature rule
- 2. integrate taylor of $f(x)$
- 3. calculate difference

Composite quad: Subdivide interval

$t_0 = a, t_i = a + ih, t_N = b \quad h = \frac{b-a}{N}$

$$\int_a^b f(x)dx = \sum_{i=1}^{N} \int_{t_{i-1}}^{t_i} f(x)dx$$

5.1 Basic quadrature rules

Idea: use low-order polynomial interp.

$f(x) \approx p_n(x) \Rightarrow \int_a^b f(x)dx \approx \int_a^b p_n(x)dx$

Midpoint rule/0-degree polynomial: Precision 1

$$\int_a^b f(x)dx \approx f(\frac{a+b}{2})(b-a)$$

Composite: $\int_a^b f(x)dx \approx h\sum_{i=1}^{N} f(\frac{t_{i-1}+t_i}{2})$

```
function I = I_midpoint(a, b, N, f)
    h = (b-a)/N; x = [a+h/2:h:b-h/2];
    I = h*sum(f(x)); end
```

Trapezoidal rule/1-degree polynomial: Precision 1

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}(b-a)$$

Composite: $\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{N}[f(t_{i-1})+f(t_i)]$

```
function I = I_trap_comp(a, b, N, f)
    h = (b-a)/N; x = [a+h:h:b-h];
    I = h/2 * (f(a) + 2*sum(f(x)) + f(b));
end
```

Simpson rule/2-degree polynomial: For example for 3 points if the middle point is the mid point-> x1=(x0+x2)/2 Precision 3

$$\int_a^b f(x)dx \approx \frac{b-a}{6}\left[f(a)+4f(\frac{a+b}{2})+f(b)\right]$$

Composite: for N even, $\frac{N}{2}$ sub-int. of length $2h$

$[t_{2k-2}, t_{2k}], k = 1, \ldots, \frac{N}{2}$

$\int_a^b f(x)dx \approx \frac{h}{3}[f(t_{2k-2})+4f(t_{2k-1})+f(t_{2k})]$

```
    h = (b-a)/N; x_odd = [a+h:2*h:b-h];
    x_even = [a+2*h:2*h:b-2*h];
    I = h/3 * (f(a) + 4*sum(f(x_odd)) + 2*sum(f
      (x_even)) + f(b));
```

n^{th}-degree polynomial: $L_j(x) = \Pi_{\substack{i=0 \\ i\neq j}}^{n} \frac{x-x_i}{x_j-x_i}$

$$\int_a^b f(x)dx \approx \sum_{j=0}^{n} f(x_j) \underbrace{\int_a^b L_j(x)dx}_{w_j}$$

Another way to find the wheights: Change of Variables to get from $[a,b]$ to $[-1,1] = \xi$

$$\int_a^b 1\,dx = \sum w_i, \qquad x(\xi) = \frac{b-a}{2}\xi + \frac{b-a}{2}$$

$$\int_a^b x\,dx = \sum w_i x_i, \rightarrow \frac{dx}{d\xi} = \frac{b-a}{2}$$

$$\int_a^b x^2\,dx = \sum w_i x_i^2. \int_a^b u(x)dx = \frac{b-a}{2}\int_{-1}^{1} u(x(\xi))d\xi$$

Smartest basic quadrature rule: Maximize precision for a given n_{qp} (quadrature points). How much more precision? Precision used to be $n = n_{qp} - 1$, chose n_{qp} location -> $2n_{qp} - 1$

•n_{qp} = number of quadrature points

Gauss-Legendre quad: Precision $2n_{qp} - 1$

Determine x_j, w_j ($2n_{qp}$ unknowns) ($j = 0, \ldots, n(= n_{qp} - 1)$) s.t.:

$$\int_{-1}^{1} x^k dx = \sum_{j=0}^{n_{qp}-1} w_j x_j^k \quad \text{for } k = 0, \ldots, 2n_{qp} - 1$$

$2n_{qp}$ equations. Assume x_j are symmetric around $x = 0$

All the polynomials with degree from 0 to $2n_{qp} - 1$ are approximated exactly.

$$\int_a^b f(\zeta)d\zeta \approx \frac{b-a}{2}\sum_{j=0}^{n_{qp}-1} w_j f(\frac{b-a}{2}x_j + \frac{b+a}{2})$$

Legendre Polynomials: $P_0(x) = 1, P_1(x) = x$

$P_{k+1}(x) = \frac{2k+1}{k+1}xP_k(x) - \frac{k}{k+1}P_{k-1}(x)$

Zeros of $P_{n_{qp}}(x)$ are the quadrature points

Vorgehen:

(1) find $P_{n_{qp}}(x)$,

(2) find zeros of $P_{n_{qp}}(x)$ in $(-1,1)$,

(3) find $L_j(x)$ which interpolate $f(x)$ at $x_0, x_1, \ldots, x_{n_{qp}-1}$ (n=$n_{qp} - 1$)

(4) $w_j = \int_{-1}^{1} L_j(x)dx$

Symetrie trick: • add

Quadrature	Rule	Error
Midpoint	$(b-a)f(\frac{a+b}{2})$	$\frac{f''(\xi_1)}{24}(b-a)^3$
Trapezoidal	$\frac{b-a}{2}[f(a)+f(b)]$	$-\frac{f''(\xi_2)}{12}(b-a)^3$
Simpson	$\frac{b-a}{6}[f(a)+4f(\frac{b+a}{2})+f(b)]$	$-\frac{f'''(\xi_3)}{90}(\frac{b-a}{2})^5$

6 Roots of non-linear equations

Goal: find x^* s.t. $f(x^*) = 0$

Approach: start with initial guess x_0, generate $x_1, \ldots, x_k, \ldots \rightarrow x^*$, stop when meeting one or more termination criteria, e.g.

- $|x_n - x_{n-1}| < tol_{abs}$
- $\frac{|x_n - x_{n-1}|}{|x_n|} < tol_{rel}$
- $|f(x_n)| < tol_f$

-Convergence order:

Linear convergence: $|x_{kh} - x^*| <= \varphi|x_k - x^*|$ with some $\varphi <$ 1 and all k suffi. large

Quadratic convergence: $|x_{kh} - x^*| <= M|x_k - x^*|^2$ with some M > 0 and all k suffi. large

Bisection method: **Assume:** $f(a) > 0, f(b) < 0$ -> $x^* \in [a,b]$

Start with $x_0 = \frac{a+b}{2}$, if $f(x_0) > 0$ -> root in $[x_0, b]$,If is other way around than in assumption root would be in [a,x0]

then take $x_1 = \frac{x_0+b}{2}$ and continue untill term. crit. is reached.

• **General algorithm:** $x_k = \frac{a+b}{2}$, if $f(a) * f(x_k) < 0$ $b = x_k$, else $a = x_k$

· Solution exists theoretically and can be approx. found nummerically.

· $|x_n - x^*| <= \frac{b-a}{2}\frac{1}{2^n} = tol\_abs$ -> n to achieve tol_abs.

-Asympt. we reach linear convergence with φ = 1/2

- Robustnes
- Minimal info
- Minimal smoothnes
- Efficiency
- Generalization

Fixed-point method: Start with x_0,

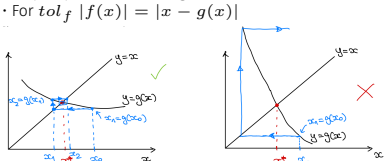
Rewrite $f(x) = 0$ as $x = g(x)$

Solve iteratively, compute: $x_1 = g(x_0)$, ..., $x_{k+1} = g(x_k)$ until a term. crit. is met.

· Exists theoretically? if $\exists x^*$ s.t. $f(x^*) = 0$, then $\exists x^*$ s.t. $x^* = g(x^*)$

· Can we approx. numerically? -> depends on g. Can chose g(x) in many ways, we want: x = g(x) <-> f(x) = 0

· For tol_f $|f(x)| = |x - g(x)|$



Fixed point theorem: · If $g \in \rho[a,b], g(a) >= a$ and $g(b) <= b$, then $\exists x^*$ s.t. $x^* = g(x^*)$

· if $\exists g'$ and $\exists \rho < 1$ s.t. $|g'(x)| <= \rho$ for all $x \in [a,b]$ then then fixed point is unique in [a,b]

Convergence: · If $\exists g'(x)$ and $\exists \rho < 1$ s.t. $|g'(x)| \leq \rho$ in $[a,b]$, convergence to the solution (if it exists)

$|x_{k+1} - x^*| = |g(x_k) - g(x^*)| = |g'(\xi)(x_k - x^*)| \leq |g'(\xi)||x_k - x^*| \leq \rho|x_k - x^*|$

$x_{k+1} = g(x_k), \quad x^* = g(x^*), \quad g(x_k) = g(x^*) + g'(\xi)(x_k - x^*), \quad \xi$ between x_k and x^*

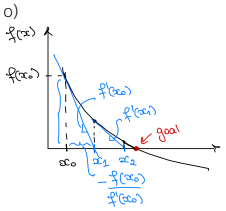
· The smaller $\rho < 1$, the faster the convergence (not always linear. If $g'(x^*) = 0$ convergence may be faster than linear!

- Minimal info
- Minimal smoothnes
- Generalization
- Efficiency
- Robustnes (But + if we ensure $|g'(x)| < 1$

```
sol = fixpt(g, x0, tol_f, k_max);
sol1 = sol1_k(end)
```

Newton-Raphson's method: Start with x_0,

Solve iteratively with $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$, k=0, 1, ..., (f'(x) not = 0)



We drop the quadratic term -> convergence is quadratic!

· If there is a solution x^* s.t. $f(x^*) = 0$ and $f'(x) not = 0$, in the vicinity of the solution x^* (local convergence), newtons method has quadratic convergence.

- Efficiency
- Generalization
- Robustness (need good initial guess)
- Minimal info
- Non minimal smoothness

If f' and f have a zero at the same point, the problem becomes ill-conditioned. The method may still converge, but convergence becomes linear.

Quasi-Newton's method Start with x_0, x_1, (two initial guesses)

$$x_{k+1} = x_k - \frac{f(x_k)(x_k - x_{k-1})}{f(x_k) - f(x_{k-1})}$$

Convergence is faster than linear, but slower than quadratic -> Superlinear convergence.

- Requires no derivative
- Generalizable to systems of equations
- Not robust (choice of initial guess)
- Fast (superlinear convergence)

6.1 System of non-linear equations

Solve $\underline{f}(\underline{x}) = 0$ where $\underline{x} = [x1x2....xn]^T$ and $\underline{f}(\underline{x}) = [f1(\underline{x})f2(\underline{x})...]^T$. n eqs., n unknowns fi(x) are non linear and we have no grantee that a solution exists and is unique. A solution will be $\underline{x}^*$ (root or zero)

• Scalar case, no closed form solution. Iterate with initial guess $\underline{x}^{(0)}$ and generate until $\underline{x}^*$ for $k \rightarrow \infty$. Stop when termination criteria is met.

• Termination criteria:

$||\underline{x}^{(k)} - \underline{x}^{(k-1)}|| < tol_{abs}$, $\frac{||\underline{x}^{(k)} - \underline{x}^{(k-1)}||}{||\underline{x}^{(k)}||} < tol_{rel}$ $(||\underline{x}^{(k)}|| \neq 0)$, $|f(\underline{x}^{(k)})| < tol_f$

• Convergence order:

Linear convergence: $||\underline{x}^{(k+1)} - \underline{x}^*|| \leq \rho||\underline{x}^{(k)} - \underline{x}^*||, \rho < 1$

Quadratic convergence: $||\underline{x}^{(k+1)} - \underline{x}^*|| \leq M||\underline{x}^{(k)} - \underline{x}^*||^2, M > 0$

$\vec{f}(\vec{x}) = \vec{0}$ n equations and n unknowns

Newton's: $\vec{x}^{(k+1)} = \vec{x}^{(k)} - J^{-1}(\vec{x}^{(k)})\vec{f}(\vec{x}^{(k)})$

Solve in two stages: $\vec{P}^{(k)} = -J^{-1}(\vec{x}^{(k)})\vec{f}(\vec{x}^{(k)})$, i.e. solve $J(\vec{x}^{(k)})\vec{P}^{(k)} = -\vec{f}(\vec{x}^{(k)})$, for $\vec{P}^{(k)}$

$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{p}^{(k)}$

Jacobian Matrix

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

6.2 (Unconstrained) optimization

• Goal find: $\min_{\vec{x}\in\mathbb{R}^n} \phi(\vec{x})$ where Φ is the Objective function/loss function.

$\max_{\vec{x}\in\mathbb{R}^n} \phi(\vec{x}) \rightleftharpoons \min_{\vec{x}\in\mathbb{R}^n} -\phi(\vec{x})$

• Often $\phi(\vec{x})$ has many local minima, the one with the smallest values of ϕ is the global minimum.

Conditions for a (local) minimum:

For n = 1:

necessary: $\phi'(\vec{x}^*) = 0$, with x^* being crit. point point

sufficient: $\phi''(\vec{x}^*) > 0$ with x^* being min point

For n > 1: necessary: $\nabla\phi(\vec{x}^*) = \vec{0}$, sufficient: $\nabla^2\phi(\vec{x}^*)$ positive definite

Hessian matrix: Symetric and nxn

$$H = \begin{bmatrix} \frac{\partial^2\phi}{\partial x_1^2} & \frac{\partial^2\phi}{\partial x_1\partial x_2} & \cdots & \frac{\partial^2\phi}{\partial x_1\partial x_n} \\ \frac{\partial^2\phi}{\partial x_2\partial x_1} & \frac{\partial^2\phi}{\partial x_2^2} & \cdots & \frac{\partial^2\phi}{\partial x_2\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2\phi}{\partial x_n\partial x_1} & \frac{\partial^2\phi}{\partial x_n\partial x_2} & \cdots & \frac{\partial^2\phi}{\partial x_n^2} \end{bmatrix}$$

λ_i: eigenvalues of $\nabla^2\phi(\vec{x})$. $\nabla^2\phi(\vec{x})$: Positive def. if all λ_i > 0 (local min), negative def. if all λ_i < 0 (local max), indefinite if some bigger, some smaller then 0.

Newton's method for unconstrained optimization:

Recall: ($J = \nabla^2\phi$)

- Solve $(J(\vec{x}^{(k)})\vec{p}^{(k)} = -\vec{f}(\vec{x}^{(k)}))$ for $(\vec{p}^{(k)}) \rightarrow \vec{P}^{(k)} = -J(\vec{x}^{(k)})^{-1}\vec{f}(\vec{x}^{(k)}) = -[\nabla^2\phi(\vec{x}^{(k)})]^{-1}\nabla\phi(x^{(k)})$
- Set$(\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{p}^{(k)})$

$\nabla^2\phi(\vec{x}^{(k)})\vec{p}^{(k)} = -\nabla\phi(\vec{x}^{(k)})$

$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{p}^{(k)}$

• Notes: +quadratic convergence, - need to compute J of f = $\nabla^2\phi(x)$at each iteration, - n ~ solve a lin. system of eqs., - may or may not converge to a min, in general converges to a crit point.

6.2.1 Link between the two:

If $\vec{f} = \nabla\phi$: $\nabla\phi(\vec{x}) = 0 \rightleftharpoons \vec{f}(\vec{x}) = \vec{0}$

$$\nabla^2\phi(\vec{x}) = \nabla\vec{f}(\vec{x}) = J(\vec{x})$$

6.2.2 Uncosntrained optimization methods:

Most methods use: $\vec{p}^{(k)} = -B_k^{-1}\nabla\phi(\vec{x}^{(k)})$

, $\vec{x}^{(k+1)} = \vec{x}^{(k)} + \alpha^{(k)}\vec{p}^{(k)}$

In newtons method: $B_k = \nabla^2\phi(\vec{x}^{(k)})$

If B_k is symm. positive definite:

every $\vec{p}^{(k)}$ is a descent direction

Newton: $\alpha = 1$, Gradient descent: $\alpha < 1$

•Alternatives: Insert the theory of this from the last lecture?

7 Appendix

Algorithm Time: $t_\backslash < t_{QR} < t_{LU} << t_{SVD}$

7.1 Orhtogonalprojektion

Zwei Vektoren sind orthogonal, falls $\langle x, y\rangle = 0$.
Die Orthogonalprojektion p des Vektors x auf Vektor y ist:

$$p = \frac{\langle x, y\rangle}{\langle y, y\rangle} \cdot y$$

7.2 Orthogonalisierungsverfahren

Orthonormalbasis zu erzeugen.

- to find the orhtonormal vector to two known unity vectors, just do the cross prodcut: $e_3 = \vec{e_1} \times \vec{e_2}$

Bei einer Orthonormalbasis sind alle Basisvektoren:

- orthogonal zueinander: $\langle b_i, b_j\rangle = 0$
- Einheitsvektoren: $||b_i|| = \sqrt{\langle b_i, b_i\rangle} = 1$

1 Wähle beliebigen ersten Basisvektor b_1 und normiere mit von Skalarprodukt induzierter Norm.

$$e_1 = \frac{b_1}{||b_1||} = \frac{b_1}{\sqrt{\langle b_1, b_1\rangle}}$$

2 Wähle zweiten Basisvektor b_2. Zuerst zu b_1 parallelen Teil abziehen, und dann normieren.

$$e_2' = b_2 - \langle b_2, e_1\rangle \cdot e_1$$
$$e_2 = \frac{e_2'}{||e_2'||} = \frac{e_2'}{\sqrt{\langle e_2', e_2'\rangle}}$$

3 Wiederhole für jeden weiteren Basisvektor b_i:

$$e_i' = b_i - \langle b_i, e_1\rangle \cdot e_1 - \langle b_i, e_2\rangle \cdot e_2 - \ldots - \langle b_i, e_{i-1}\rangle \cdot e_{i-1}$$
$$e_i = \frac{e_i'}{||e_i'||} = \frac{e_i'}{\sqrt{\langle e_i', e_i'\rangle}}$$

7.3 QR-Zerlegung

Kleinste Quadrate mit QR-Zerlegung
Löst man ein Optimierungsproblem mit dem Computer, liefert das in 10.1 beschriebene Verfahren ungenaue Lösungen (da numerisch instabil). Das Lösungsverfahren mittels QR-Zerlegung ist besser. **In Aufgabe nur machen, wenn explizit verlangt!**

Vorgehen:

1 Man bestimme A und c wie bei 10.1.

Bsp: $A = \begin{pmatrix}1 & 0\\ 0 & 1\\ 1 & 1\end{pmatrix}$, $c = \begin{pmatrix}1\\0\\1\end{pmatrix}$

2 Man führe die QR-Zerlegung durch $A = QR$

Bsp: $Q = \begin{pmatrix}1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3}\\ 0 & \sqrt{2}/\sqrt{3} & -1/\sqrt{3}\\ -1/\sqrt{2} & 1/\sqrt{6} & 1/\sqrt{3}\end{pmatrix}$, $R = \begin{pmatrix}\sqrt{2} & 1/\sqrt{2}\\ 0 & \sqrt{3}/\sqrt{2}\\ 0 & 0\end{pmatrix}$

3 Man berechne $d = Q^T \cdot c$

Bsp: $d = Q^T \cdot c = \begin{pmatrix}0\\ \sqrt{2}/\sqrt{3}\\ 2/\sqrt{3}\end{pmatrix}$

4 Man berechne löse das Gleichungssystem $R_0 \cdot x = d_0$, wobei R_0 die extrahierte Dreiecksmatrix aus R ist und d_0 die dazugehörigen oberen Einträge von d

Bsp: $R_0 = \begin{pmatrix}\sqrt{2} & 1/\sqrt{2}\\ 0 & \sqrt{3}/\sqrt{2}\end{pmatrix}$, $d_0 = \begin{pmatrix}0\\ \sqrt{2}/\sqrt{3}\end{pmatrix}$

7.4 QR Zerlegung für square matrices (nxn) $\rightarrow \mathcal{O}(m^3)$

Consider the matrix

$$A = \begin{bmatrix}1 & 1 & 0\\ 1 & 0 & 1\\ 0 & 1 & 1\end{bmatrix}, \quad \Leftrightarrow \quad A = Q \cdot R$$

with the vectors

$$\mathbf{a}_1 = \begin{bmatrix}1\\1\\0\end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix}1\\0\\1\end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix}0\\1\\1\end{bmatrix}.$$

Find the vectors $e_1^\rightarrow, e_2^\rightarrow$.... by using the Gram Schmidt Orhtonormalisierungsverfahren. The matrix Q is given by:

$$Q = [\mathbf{e}_1, \quad \mathbf{e}_2, \quad \cdots \quad \mathbf{e}_n]$$

The matrix R is given by:

$$R = \begin{bmatrix}\mathbf{a}_1 \cdot \mathbf{e}_1 & \mathbf{a}_2 \cdot \mathbf{e}_1 & \mathbf{a}_3 \cdot \mathbf{e}_1\\ 0 & \mathbf{a}_2 \cdot \mathbf{e}_2 & \mathbf{a}_3 \cdot \mathbf{e}_2\\ 0 & 0 & \mathbf{a}_3 \cdot \mathbf{e}_3\end{bmatrix}$$

7.5 QR applied

Calculate QR decomposition of A

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

a_i = i-th column
Gram-Schmidt algorithm for QR decomposition

$$q_1 = \frac{a_1}{||a_1||} = \begin{bmatrix}1/3\\ 2/3\\ -2/3\end{bmatrix}$$

$$q_2 = \frac{a_2 - (a_2^T q_1)q_1}{||a_2 - (a_2^T q_1)q_1||} = \begin{bmatrix}2/3\\ 1/3\\ 2/3\end{bmatrix}$$

$$\Rightarrow Q = [q_1 \quad q_2]$$
$$R_{ij} = q_i^T \cdot a_j$$

R is **ALWAYS** and upper triangular matrix

7.6 Singular Values Decomposition SVD step by step

Pleas validate that this works, its from google and chat gpt, i tried it on a example and it didnt work properly, might have to do some changes to it or how do you calculate svd Might need to add a minus sometimes or so?
Given: Matrix A (This procedure was also used in ex class 6 Ex 1 C ?! Does this work for all matrices?
1. Calculate the eigenvalues and eigenvectors of AA^T and A^TA. Calculate their absolute values in ascending order.
2. Find the singular values: $\sigma = \sqrt{\lambda}$ (sqrt of absolute values of eigenvalues)
3. Form the Σ matrix by placing the singular values in the diagonal ordered from largest to smallest.
4. Form U and V matrix. Normalize the eigenvectors to form the columns of the U and V matrices. U is formed using the eigenvectors of the AA^T matrix, and V is formed using the eigenvectors of the A^TA matrix.
5. To ensure the SVD is correct, check that $A = U\Sigma V^T$. Now we will perform these steps for
For A not nxn dimensions: U: mxm, Σ: mxn, V: nxn

Notes: In SVD U and V Rotate, and Σ scales the coordinate system

7.7 LU Decompostion $\rightarrow \mathcal{O}(m^3)$

LU: compute LU(A^TA)

7.8 How to pick n for Linear Regression

- Data : $(t_1, y_1), (t_2, y_2), ..(t_m, y_m)$
  m= #Data, n= Order of the model = #unknown params = #DoF
- Model: $p_n(x) = \sum_{i=1}^{n} \phi_i x_i$
- $n \leq m \begin{cases} n = m & \textbf{Interpolation} \\ & \Rightarrow x_i \text{ so that } r = y - Ax = 0 \\ n < m & \textbf{Regression} \\ & \Rightarrow x_i \text{ so that } ||r||_2^2 = ||y - Ax||_2^2 \rightarrow min \end{cases}$
- Measure Goodness of fit (very common: use MSE)

$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - p_n(x_i))^2$$

8 Pseudo Code

8.1 Pseudo code free boundary condition spline

We do cubic spline interpolation that doest just go from left to right, thus parametrise with t. pseudocode to compute values of $x(t)$ and $y(t)$ from interpolating spline using free boundary conditions over parameter interval $[a, b]$ (in this example $[1, 13]$) with parameter step of $\Delta t (= 0.01)$ 13 Points $\rightarrow n = 13 - 1 = 12$ subintervals.
$s_i(t) = a_i + b_i * t + e_i * t^2 + d_i * t^3, i = 0, \ldots, n-1 = 11$

pp.coefs = $\begin{bmatrix} d_0 & c_0 & b_0 & a_0 \\ d_1 & c_1 & b_1 & a_1 \\ \vdots & \vdots & \vdots & \vdots \\ d_{n-1} & c_{n-1} & b_{n-1} & a_{n-1} \end{bmatrix}$

Pseudo Code:

```
x = ... %from table
y = ... %from table
delta_t = 0.01
t_tilde = 1 : delta_t : 13 %query point vector
pp_x = csape (t,x,'variational')
x_tilde = ppval (pp_x, t_tilde)
pp_y = csape (t, y, 'variational')
y_tilde = ppval (pp_y, t_tilde)
plot(x_tilde, y_tilde)
```

func	meaning
csape	returns cubic spline interpolation with data x/y
ppval(pp,xq)	eval. the piecewise polynomial at query points xq

8.2 Code for monomial interpolation

```
x = [1,2,4];
y = [1,3,3];
x_tilde = 0:0.01:5; %query points
n = length(x)-1; %degree of the poly. interp.
j = 0 : n;
X = x'.^j;
c = X\y';
% EVALUATION
m = length(x_tilde); %number of query points
y_tilde = zeros(m,1);
for k = 1 : m
    v_k = x_tilde(k).^j;
    % evaluation at query point
    y_tilde(k) = c'*v_k';
end
```

' to transpose

8.3 System solve with perturbation

System $Ax = b$ $A = \begin{bmatrix}2 & -1\\ -1 & 2\end{bmatrix}$, $b = \begin{bmatrix}\sqrt{3}/2\\ \sqrt{2}/2\end{bmatrix}$
$b + \delta b$ where $\delta b_i \in [0, 2]$
Pseudo code for δx such that $A(x+\delta x) = b + \delta b$

```
A=[2, -1 ; -1,2]
B=[sqrt(2)/2 ; -sqrt(2)/2]
X = A\b;
delta_b = 2 * rand(2,1)
b_plus_bdelta = b + delta_b
x_plus_xdelta = A\b_plus_bdelta
delta_x = x_plus_xdelta - X
```

8.4 Matlab functions

- ./ : Pointwise division used when a variable is a vector.
- a = linspace(x1, x2, n) : Equidistant points (a = (0:n-1)/(n-1)) also works
- for i = 2:n : loops from 2 to n (inclusive)
- norm(x, #) #-norm, cond = cond(A,#) # condition number
- eig(A) : calculates the eigenvalues
- [V, D] = eig(A) : Creates eigenvector and eigenvalue matrix.
- X\Y to solve system in matlab
- prod(vector) : To calculate product for example in Lagrange poly.

8.5 gram schmidt, from chat gpt

```
function Q = gram_schmidt(A)
    % Input:  A - an m x n matrix (columns are
      vectors to be orthogonalized)
    % Output: Q - an m x n matrix with
      orthonormal columns
    [m, n] = size(A);  % Get dimensions of A
    Q = zeros(m, n);   % Initialize Q with
      zeros
    for i = 1:n
        v = A(:, i);  % Take the i-th column of
       A
        for j = 1:i-1
            v = v - (Q(:, j)' * v) * Q(:, j);
      % Subtract projection onto Q(:, j)
        end
        Q(:, i) = v / norm(v);  % Normalize v
      to make it unit length
    end
end
```

8.6 Example questions

Ex. obtain a,b,c for $u''(0) \approx \frac{2u_0 + au_{-h} + 4u_{-2h} + bu_{-3h}}{ch^2}$ (1)

1) Taylor expansions:
$u(-h) = u_0 - hu_0' + \frac{h^2}{2}u_0'' - \frac{h^3}{6}u_0^{(3)} + \frac{h^4}{24}u_0^{(4)} \ldots$
$u(-2h) = u_0 - 2hu_0' + \frac{4h^2}{2}u_0'' - \frac{8h^3}{6}u_0^{(3)} + \frac{16h^4}{24}u_0^{(4)} \ldots$
$u(-3h) = u_0 - 3hu_0' + \frac{9h^2}{2}u_0'' - \frac{27h^3}{6}u_0^{(3)} + \frac{81h^4}{24}u_0^{(4)} \ldots$

2) Substitute them into formula (1) & rearrange for u_0, u_0', u_0'' etc parts

$u''(0) = \underbrace{\frac{(2+a+b)u_0}{ch^2}}_{=0} - \underbrace{h\frac{(a+8+3b)u_0'}{ch^2}}_{=0} + \underbrace{h^2\frac{(a+16+9b)u_0''}{2ch^2}}_{=1} + \mathcal{O}(h)$

$\left[-h^3\frac{(a+32+27b)u_0^{(3)}}{6ch^2} + h^4\frac{(a+64+81b)u_0^{(4)}}{24ch^2} + \ldots\right]$

3) Set u_0, u_0' parts to zero, $u_0'' = 1$ for the formula to be true
to we need 3 formulas for 3 unknowns
This leads to:

$$\begin{cases} a + b + 6 = 0 \\ a + 3b + 8 = 0 \\ \frac{a + 9b + 16}{2c} = 1 \end{cases} \rightarrow \begin{matrix} a = -5 \\ b = -1 \\ c = 1 \end{matrix} \rightarrow u''(0) = \frac{2u_0 - 5u_h + 4u_{-2h} - u_{-3h}}{h^2} \quad (I)$$

Truncation error & order of accuracy?
For blue part insert a,b & c $\rightarrow |D_h| = |E_h| = \frac{11}{12}h^2 u^{(4)}(\xi)$ $(\xi \in [-3h, 0])$
$\hookrightarrow u^{(3)}$ term becomes 0, $\rightarrow$ use $u_0^{(4)}$ term for the error

Roundoff error $\tilde{E} \leq \frac{|2\epsilon| + |-5\epsilon| + |4\epsilon| + |-\epsilon|}{h^2}$ from (I) each term has roundoff error ϵ

Total error: $\tilde{D}_h = D_h + \tilde{D} = E_h + \tilde{E} \leq \ldots$
In matlab E_obs = abs(dudx - true_dudx)

Using linear least squares to approximate points $(x_0, f_0), (x_1, f_1), (x_2, f_2)$ with $p(x) = a + bx$
1) Calculate a & b
$p(x) = a + bx$, $f_i = a + bx_i$ v_i $\rightarrow$ $\begin{bmatrix}1 & x_0\\ 1 & x_1\\ 1 & x_2\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \begin{bmatrix}f_0\\ f_1\\ f_2\end{bmatrix}$
$A \quad \theta \quad y$
$\rightarrow$ Normal equations: $A^TA\theta = A^Ty$

$\begin{bmatrix}1 & 1 & 1\\ x_0 & x_1 & x_2\end{bmatrix}\begin{bmatrix}1 & x_0\\ 1 & x_1\\ 1 & x_2\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \begin{bmatrix}1 & 1 & 1\\ x_0 & x_1 & x_2\end{bmatrix}\begin{bmatrix}f_0\\ f_1\\ f_2\end{bmatrix}$ $(\sum x_i \cdot \sum x_i x_j \neq \sum x_i^2)$

(I) $\sum x_i \left(\begin{bmatrix}3 & \sum x_i\\ \sum x_i & \sum x_i^2\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \begin{bmatrix}\sum f_i\\ \sum f_i x_i\end{bmatrix}\right)$ (II) $-$ (I) to get b
(II) 3. $\left(\begin{bmatrix}3 & \sum x_i\\ \sum x_i & \sum x_i^2\end{bmatrix}\begin{bmatrix}a\\ b\end{bmatrix} = \begin{bmatrix}\sum f_i\\ \sum f_i x_i\end{bmatrix}\right)$ do similar to get a.

$b = \frac{3\sum f_i x_i - \sum f_i \sum x_i}{3\sum x_i^2 - (\sum x_i)(\sum x_i)}$ $a = \frac{(\sum f_i)(\sum x_i^2) - (\sum x_i f_i)(\sum x_i)}{3\sum x_i^2 - (\sum x_i)(\sum x_i)}$

or: $[f_0, f_1, f_2]^T$

or
$x^* = \begin{bmatrix}a\\ b\end{bmatrix} = (A^TA)^{-1}A^Ty$

$\begin{bmatrix}a\\ b\end{bmatrix} = \begin{pmatrix}3 & \sum x_i\\ \sum x_i & \sum x_i^2\end{pmatrix}^{-1}\begin{pmatrix}\sum f_i\\ \sum f_i x_i\end{pmatrix} = \frac{1}{3\sum x_i^2 - (\sum x_i)^2}\begin{pmatrix}\sum x_i^2 & -\sum x_i\\ -\sum x_i & 3\end{pmatrix}\begin{pmatrix}\sum f_i\\ \sum f_i x_i\end{pmatrix}$

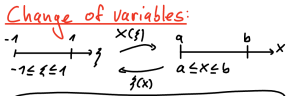
$\begin{bmatrix}a\\ b\end{bmatrix} = \frac{1}{3\sum x_i^2 - (\sum x_i)(\sum x_i)}\begin{pmatrix}(\sum x_i^2)(\sum f_i) - (\sum x_i)(\sum f_i x_i)\\ -(\sum x_i)(\sum f_i) + 3\sum f_i x_i\end{pmatrix}$

2) Approx function $p(x) = \sin(x)$ using lin approx & eval@ $x_0 = 0$, $x_1 = \frac{\pi}{2}$, $x_2 = \pi$
Replace f_i & x_i in the formula with $p(x) = \sin(x)$ & the three points
$a = \frac{(\frac{\pi^2}{2} + \pi^2)(\sin(0) + \sin(\frac{\pi}{2}) + \sin(\pi)) - (0 + \frac{\pi}{2} + \pi)(\sin(0)\cdot 0 + \sin(\frac{\pi}{2})\frac{\pi}{2} + \sin(\pi)\cdot\pi)}{3(0^2 + (\frac{\pi}{2})^2 + \pi^2) - (0 + \pi/2 + \pi)(0 + \frac{\pi}{2} + \pi)}$

$a = \frac{1}{3}$, $b = 0$

3) Pseudocode to get coefficients:
$X = [0, \frac{\pi}{2}, \pi]$
$Y = \sin(X)$
$P = \text{polifit}(X, Y, 1)$ Degree 1 from $P(x) = a + bx$
or (sfit(X,Y,1) ?
$\rightarrow p = [b, a]$ in descending order
X_pred = polyval(P, linspace(0,π,100)) 100 points
or Y_pred = polyval(flip(P), linspace(0,π,100) ?

Change of variables:



$x(\xi) = m\xi + u$

$x(-1) = a \Rightarrow -m + u = a$ (1)

$x(1) = b \Rightarrow m + u = b$ (2)

(2)-(1): $2m = b - a$

(1)+(2): $2u = b + a$

$$\Rightarrow x(\xi) = \frac{b-a}{2}\xi + \frac{b+a}{2} \qquad \frac{\partial x}{\partial \xi} = \frac{b-a}{2}$$

$$\int_a^b f(x)\,dx = \int_{-1}^{1} f\left(\frac{b-a}{2}\xi + \frac{b+a}{2}\right)\frac{\partial x}{\partial \xi}\,d\xi = \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}\xi + \frac{b+a}{2}\right)d\xi$$

$$\simeq \frac{b-a}{2}\sum_{j=0}^{n_p-1} w_j f\left(\frac{b-a}{2}\xi_j + \frac{b+a}{2}\right)$$