Interpolation

- Goal: Find a function $v(x)$ which can be used to estimate sampled function for $x \neq x_i$
- Interpolation is the reverse of evaluation
  - Evaluation: given a polynomial, evaluate a y-value for a given x-value
  - Interpolation: given these points, compute a polynomial that can generate them

Lagrange interpolation

Given n data points $(x_1, y_1), \dots, (x_n, y_n)$, the polynomial that interpolates the points is:

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

$$L_k(x) = \frac{(x-x_1)\dots(x-x_{k-1})(x_{k+1})\dots(x-x_n)}{(x_k-x_1)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)}$$

Degree of $P(x) = n-1$ or less

📌 Theorem. Let $(x_1, y_1), \dots, (x_n, y_n)$ be n points in a plane with distinct x_i.
Then there exists one and only one polynomial P of degree n-1 or less thats satisfies $P(x_i) - y_i$ for $i = 1, \dots n$

Newton's divided difference

- Given n points, the polynomial P will in this form:
  $P(x) = c_0 + c_1(x-x_1) + c_2(x-x_1)(x-x_2) \dots + c_{n-1}(x-x_1)\dots(x-x_{n-1})$
- Denoted by $f[x_1 \dots x_n]$ the coefficient of the x^{n-1} term, $f[x_1 \dots x_n] \equiv c_{n-1}$
  - k-th divided difference : $f[x_i \dots x_{i+k}] = f[x_i \dots x_{i+k}] = \frac{f[x_{i+1} \dots x_{i+k}] - f[x_i \dots x_{i+k-1}]}{x_{i+k} - x_i}$
  - e.g. $f[x_1, x_2, x_3, x_4] = \frac{f[x_2,x_3,x_4] - f[x_1,x_2,x_3,x_4]}{x_4 - x_1}$

Interpolation Error

The interpolation error at x is $f(x) - P(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_n)}{n!} f^{(n)}(c)$,
c lies between the smallest and largest of x_i

The error will be smaller close to the center of the interpolation interval

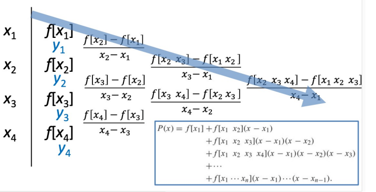
Runge phenomenon

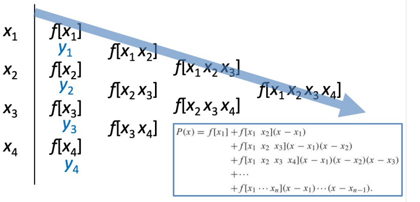
- Polynomial wiggle near the ends of the interpolation interval
- e.g. $f(x) = \frac{1}{(1+12x^2)}$
- 高階多項式通常不適合用於插值，
  多項式的階數增高時插值誤差甚至會趨向無限大

Interpolation: The new value $x \neq x_i$ is inside the range of the interpolation points $x_0, x_1, \dots, x_n$

Extrapolation: The new value z is outside the range

Approximation: Some norm $||v - y||$ of the difference of the vector $v = [v(x_0), \dots, v(x_n)]$ and $y = [y_0, \dots, y_n]$ is minimized





QR Factorization

- Avoid computing A^TA (To solve the ill-conditioned)
- $A = QR$
  - Q is orthogonal → $Q^{-1} = Q^T$
  - R is upper triangular

$$Ax = b$$
$$\to QRx = b$$
$$\to Q^{-1}QRx = Q^{-1}b = Q^Tb$$
$$\to Rx = Q^Tb$$

- **Gram-Schmidt method**
  - *Orthogonalizes a set of vectors*
  - Input : *n* linearly independent input vectors (*Ai*)
  - Output : *n* mutually perpendicular unit vectors spanning the same space (q_i)
    1. 1st: $y_1 = A_1,\ q_1 = \frac{y_1}{||y_1||^2}$
    2. 2nd: $y_2 = A_2 - q_1(q_1^TA_2),\ q_2 = \frac{y_2}{||y_2||^2}$
    3. j st: $y_j = A_j - q_1(q_1^TA_j) - q_2(q_2^TA_j) - \dots - q_{j-1}(q_{j-1}^TA_j)$,
       $q_j = \frac{y_j}{||y_j||^2}$
- Full QR Factorization
  - Add a 3rd vector arbitrary

$$\text{error magnification factor} = \frac{\text{relative forward error}}{\text{relative backward error}} = \text{cond(A)}$$

$$= \|A\| \times \|A^{-1}\|$$

- Orthogonal set
  - A set of vectors in which $v_i^Tv_j = 0$ whenever $i \neq j$.
  - Example: $\{[1, 1, 1]^T, [2, 1, -3]^T, [4, -5, 1]^T\}$
- Orthonormal set
  - An orthogonal set of ***unit vectors***. a vector of length 1
  - $\|v\|_2 = \sqrt{v_1^2 + v_2^2 + \dots + v_n^2} = 1$
  - Example: $\{[0, 0, 1]^T, [0, 1, 0]^T, [1, 0, 0]^T\}$
  - Normalizing a vector? $u/\|u\|_2$
- Orthogonal matrix
  - The column vectors form an orthonormal set.
  - $Q^{-1} = Q^T$

$$A = \begin{bmatrix} 1 & -4 \\ 2 & 3 \\ 2 & 2 \end{bmatrix} = \begin{bmatrix} 1/3 & -14/15 \\ 2/3 & 1/3 \\ 2/3 & 2/15 \end{bmatrix} \begin{bmatrix} 3 & 2 \\ 0 & 5 \end{bmatrix} = QR$$

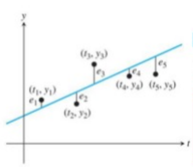
q_1 q_2 R

$$r_{11} = ||y_1||_2 = \sqrt{1^2 + 2^2 + 2^2} = 3,$$
$$r_{22} = ||y_2||_2 = 5$$
$$r_{12} = q_1^TA_2 = 2$$

$$\boxed{\begin{aligned} r_{jj} &= \|y\|_2 \\ r_{ij} &= q_i^TA_j \end{aligned}}$$

Least Squares Approximation

- Find the closest *x*
- Fitting model to data



linear model
$y = at + b$

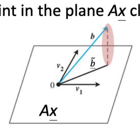
$y_2 = at_2 + b$
$y_3 = at_3 + b$
$y_4 = at_4 + b$
$y_5 = at_5 + b$

Find a, b such that
$e_1^2 + e_2^2 + e_3^2 + e_4^2 + e_5^2$
is minimized!

Normal equations

Solving an inconsistent system (a system has no solution)

- Any $m \times n$ system $A\underline{x} = \underline{b}$ can be view as a vector equation $x_1v_1 + x_2v_2 + \dots + x_nv_n = b$
- b is a linear combination of the column v_i of A , with coefficient $x_1 \dots x_n$
- Solution
  - Yes → b lies on the plane
  - No → Find the closest instead (Least squares solution)
- Find a point in the plane $A\bar{x}$ closest to $\bar{b}$



closest solution

- $\tilde{b} = A\tilde{x}$
- Residual vector $b - \tilde{b} = b - A\tilde{x}$

$\tilde{b} = A\tilde{x}$
→ Residual vector $r = \underline{b} - \tilde{b} = \underline{b} - A\tilde{x}$
$\underline{b} - A\tilde{x} \perp A\underline{x}$
→ $A\underline{x}^T(\underline{b} - A\tilde{x}) = 0$
→ $\underline{x}^TA^T(\underline{b} - A\tilde{x}) = 0$
→ $A^T(\underline{b} - A\tilde{x}) = 0$
→ $A^TA\tilde{x} = A^T\underline{b}$ → Normal Equations
→ $(A^TA)\tilde{x} = (A^T\underline{b})$

he solution $\tilde{x}$ is the **least squares solution** of the system

Fitting data

1. Choose a model (e.g. Linear, Parabola, Periodic...)
2. Force the model to fit the data
3. Solve the normal equations

✏️ Data linearization
Notice: It changes the least squares problem
(Errors in log space)
- Exponential model
- Power law model

- Euclidean length (2-norm)

$$\|\underline{r}\|_2 = \sqrt{r_1^2 + \dots + r_m^2}$$

- Squared error

$$\text{SE} = r_1^2 + \dots + r_m^2$$

- Root mean squared error

$$\text{RMSE} = \sqrt{\text{SE}/m} = \sqrt{(r_1^2 + \dots + r_m^2)/m}$$

Given an inconsistent system

$$\underset{m*n}{A}\underline{x} = \underset{m*1}{\underline{b}},$$

solve

$$\underset{n*n}{(A^TA)}\tilde{\underline{x}} = \underset{n*1}{A^T\underline{b}}$$

$$\underline{v}_k = -\left(D_r(\underline{x}_k)^TD_r(\underline{x}_k)\right)^{-1}D_r(\underline{x}_k)^Tr(\underline{x}_k)$$

$$\underbrace{\left(D_r(\underline{x}_k)^TD_r(\underline{x}_k)\right)}_{n\times m \quad m\times n}\underbrace{\underline{v}_k}_{n\times 1} = \underbrace{-D_r(\underline{x}_k)^Tr(\underline{x}_k)}_{n\times m \quad m\times 1}$$

$$\begin{bmatrix} \frac{\partial r_1}{\partial x_1} & \cdots & \frac{\partial r_1}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial r_m}{\partial x_1} & \cdots & \frac{\partial r_m}{\partial x_n} \end{bmatrix} m \times n$$

Gauss-Newton

$\underline{x}_0$ = initial vector
$\underline{x}_{k+1} = \underline{x}_k - \left(D_r(\underline{x}_k)^TD_r(\underline{x}_k)\right)^{-1}D_r(\underline{x}_k)^Tr(\underline{x}_k)$ for $k = 0,1,2,\dots$

$\underline{x}_{k+1} = \underline{x}_k + \underline{v}_k$ $\underline{v}_k$

Gauss-Newton Method (Nonlinear Least Squares)

- Multivariate Newton's method + Normal equations
- Consider the system of m equations in b unknowns (m < n)
  - $r_1(x_1, \dots x_n) = 0$
  - $r_m(x_1, \dots, x_n) = 0$
  - Find a solution $\underline{x}$ that minimizes the sum $r_1(\underline{x})^2 + r_2(\underline{x})^2 + \dots + r_m(\underline{x})^2$
  - Steps
    1. Set x_0 = initial vector
    2. for k = 0, 1, 2...

$$A = D_r(\underline{x}^k)$$
$$A^TA\underline{v}^k = -A^Tr(\underline{x}^k)$$
$$\underline{x}^{k+1} = \underline{x}^k + \underline{v}^k$$

Optimization

- Finds the values of *n*-variables $(x_1, x_2, x_3, \dots, x_n)$ that minimize (or maximize) some objective function *f*(*x*)

$$\underline{x}^* = argmin\ f(x)$$

Gradient descent

- Algorithm
  1. Pick an initial point x_0
  2. Iterate until convergence: $x_{t+1} = x_t - \eta\nabla f(x_t)$
     - η : step size / learning rate
     - Iterate until $||\nabla f(x_t)|| \leq \epsilon$

⚠️ **Gradient descent is a greedy method!**
It may return a local minimum!