

MAD - Formelsammlung

Jorit Geurts jgeurts@student.ethz.ch

Version: 26. Juni 2021

1 Linear Least Squares

Given a Set of $\{(x_i, y_i)\}_{i=1}^N$ data fit a **Function** $f(x)$ to the data. $f(x)$ can be expressed as a **linear combination of M linearly independent functions** $\varphi_k(x)$.

$$f(x; \mathbf{w}) = \sum_{k=1}^N w_k \varphi_k(x) \quad (1)$$

$\mathbf{w} = (w_1, \dots, w_M)$ are unknown weights we have to find.

$\varphi_k(x)$ are the basis functions where typically $M \ll N$.

Some typical basis functions:

$$\begin{aligned} \varphi_k(x) &= x^{k-1}, & \varphi_k(x) &= \cos((k-1)x) \\ \varphi_k(x) &= e^{\beta k x}, & \varphi_k(x) &= 1 - \frac{x - x_k}{\delta} \end{aligned} \quad (2)$$

The name **Linear Least Squares** comes from the fact that the unknown parameter w_k come in linearly and not that we fit a linear function.

We want to find w_k with $k = 1, \dots, M$ such that the error function $E(\mathbf{w})$ is **minimised**.

$$E(\mathbf{w}) = \|\mathbf{e}\|_2^2 = \sum_{i=1}^N e_i^2 = \sum_{i=1}^N (y_i - f(x_i))^2 \quad (3)$$

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} E(\mathbf{w}) \quad (4)$$

1.1 Matrix Formulation

We want to solve the following equation.

$$A\mathbf{w} = \mathbf{y} \quad (5)$$

where

$$\begin{matrix} A \in \mathbb{R}^{N \times M}, \text{ Regression or Least Square Matrix} & \mathbf{w} \in \mathbb{R}^M & \mathbf{y} \in \mathbb{R}^N \\ \begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_M(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \dots & \varphi_M(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_1(x_N) & \varphi_2(x_N) & \dots & \varphi_M(x_N) \end{bmatrix} & \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix} & = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \end{matrix}$$

We need to solve the above equation for \mathbf{w} . We distinguish between three different cases:

- **M = N:** (Easy case)
The Matrix A is square. We get the solution $\mathbf{w} = A^{-1}\mathbf{y}$
- **M > N:** (Not interesting)
The system is undetermined and has infinite solutions.
- **M < N:** (Most of the time its this case)
The system is overdetermined. We can seek an approximate solution $A\mathbf{w} \approx \mathbf{y}$ with the least squares method by requiring that $E(\mathbf{w}) = \|\mathbf{y} - A\mathbf{w}\|_2^2$

1.1.1 Solution for M < N

Via derivation of the Error function we get the normal equation.

Normal equation:

$$A^T A \mathbf{w} = A^T \mathbf{y}. \quad (6)$$

Since the φ_k are linearly independent the matrix $A^T A$ is symmetric, we get a solution for \mathbf{w} .

Solution for w:

$$\mathbf{w}^* = (A^T A)^{-1} A^T \mathbf{y} \quad (7)$$

1.1.2 Special case orthonormal case

When the columns \mathbf{a}_i of the matrix $A \in \mathbb{R}^{N \times M}$ are orthonormal.

$$\mathbf{a}_i \cdot \mathbf{a}_j = \delta_{ij} \quad (8)$$

Then $A^T A = I$ and we get the following solution for \mathbf{w} :

$$\mathbf{w}^* = A^T \mathbf{y} \quad (9)$$

1.1.3 Solution for Linear Function

Fit data to $f(x) = w_1 + w_2 x$:

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

Normal case:

$$w_1^* = \frac{\sum_{i=1}^N x_i^2 \sum_{i=1}^N y_i - \sum_{i=1}^N x_i \sum_{i=1}^N x_i y_i}{N \sum_{i=1}^N x_i^2 - (\sum_{i=1}^N x_i)^2}$$

$$w_2^* = \frac{N \sum_{i=1}^N x_i y_i - \sum_{i=1}^N x_i \sum_{i=1}^N y_i}{N \sum_{i=1}^N x_i^2 - (\sum_{i=1}^N x_i)^2}$$

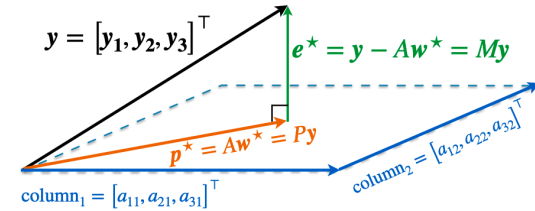
Orthonormal case:

Rewrite $y = w_1 + w_2 x$ as $y = w_1 + w_2(x - \bar{x})$. LSQ yields

$$w_1^* = \frac{\sum y_i}{N}; \quad w_2^* = \frac{\sum (x_i - \bar{x}) y_i}{\sum (x_i - \bar{x})^2}$$

1.2 Geometric interpretation

The columns \mathbf{a}_i of $A \in \mathbb{R}^{N \times M}$ create a M-dimensional space. The solution $A\mathbf{w}^*$ is a projection of \mathbf{y} onto that space spanned by A. The residual error is perpendicular to that space.



The projected Vector:

$$\mathbf{p}^* = A\mathbf{w}^* = A(A^T A)^{-1} A^T \mathbf{y} = P\mathbf{y}$$

Projection Matrix:

$$P = A(A^T A)^{-1} A^T \quad (10)$$

Properties of the projection matrix:

- It is symmetric: $P = P^T$
- It is idempotent: $P = P^2$

Same goes for the error:

$$\mathbf{e}^* = (I - A(A^T A)^{-1} A^T) \mathbf{y} = M\mathbf{y} \quad (11)$$

We see that:

$$P + M = I, \quad PM = 0, \quad PA = A, \quad MA = 0 \quad (12)$$

We know from linear algebra that:

$$\mathbf{y} = \mathbf{y}_r + \mathbf{y}_n = (P + M)\mathbf{y} \quad (13)$$

1.3 Numerical Solutions

Computer can't calculate exact numbers, so there will be rounding errors: $\delta \mathbf{w} = \mathbf{w} - \tilde{\mathbf{w}}$ and $\delta \mathbf{y} = \mathbf{y} - \tilde{\mathbf{y}}$

Condition Number:

$$\frac{\|\delta \mathbf{w}\|}{\|\mathbf{w}\|} = \|A\| \|A^{-1}\| \frac{\|\delta \mathbf{y}\|}{\|\mathbf{y}\|} = \kappa(A) \frac{\|\delta \mathbf{y}\|}{\|\mathbf{y}\|} \quad (14)$$

$$\kappa(A) = \|A\| \|A^{-1}\|, \quad \kappa(A) \in [1, \infty) \quad (15)$$

A problem is well conditioned if $\kappa(A)$ is not too large.

$$\|A\|_2 = \sqrt{\lambda(A^T A)} \quad (16)$$

QR-Decomposition: $A = QR$

$$\mathbf{w}^* = R^{-1} Q^T \mathbf{y} \quad (17)$$

SVD: $A = U \Sigma V^T$

$$\mathbf{w}^* = V \Sigma^+ U^T \mathbf{y} \quad (18)$$

$$\Sigma = \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \rightarrow \Sigma^+ = \begin{bmatrix} S^{-1} & 0 \\ 0 & 0 \end{bmatrix} \quad (19)$$

2 Non Linear Systems

Goal, find root of a function:

$$f(x^*) = 0 \quad (20)$$

Sensitivity: If $|f(\tilde{x})| \approx 0$, does this mean that $|\tilde{x} - x^*| \approx 0$?

Definition:

- Well Conditioned: small change in input causes small change in output
- Ill Conditioned: small change in input causes big change in output

Condition Number:

$$\kappa = \frac{1}{|f'(x^*)|} \quad (21)$$

Note for $f'(x^*) = 0$ the problem is ill conditioned and corresponds to roots of multiplicity $m > 1$.

k -th error: $e_k = x_k - x^*$

We define r and C so that there exists a limit wich follows:

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^r} = C \quad (22)$$

$$\text{Convergence rate: } r, r \approx \frac{\log \frac{|e_{k+2}|}{|e_{k+1}|}}{\log \frac{|e_{k+1}|}{|e_k|}}$$

$$\text{Order of convergence: } C, C = \lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^r}$$

- $r = 1$: if $C \in (0, 1)$ linear convergence. If $C = 0$ superlinear, $C = 1$ sublinear
- $r = 2$: quadratic convergence

2.1 Bisection Method

Algorithm:

1. Choose interval $[a, b]$, such that $f(a) \cdot f(b) < 0$
2. Half interval ($x_k = \frac{a+b}{2}$), if $\text{sign}(f(x_k)) = \text{sign}(f(a))$ then $a = x_k$ else $b = x_k$
3. repeat until satisfied. ($|b - a| < \text{tol}$)

Facts:

- $C = 1$ and $r = \frac{1}{2}$.
- (+) $f(x)$ doesn't have to be differentiable
- (+) certain to find a solution
- (-) but slow. $k = \log_2 \left(\frac{b-a}{\text{tol}} \right)$
- (-) interval $[a, b]$ may be hard to find.

2.2 Newtons Method (Tangent Method)

Algorithm: Tangent at the point x_k

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)} \quad x_{k+1} = x_k - m \underbrace{\frac{f(x_k)}{f'(x_k)}}_{\text{roots with multiplicity } m > 1} \quad (23)$$

Convergence: $r = 2$

$$\lim_{k \rightarrow \infty} \frac{|e_{k+1}|}{|e_k|^2} = \lim_{k \rightarrow \infty} \frac{|f''(\xi_k)|}{2|f'(x_k)|} = \frac{|f''(x^*)|}{2|f'(x^*)|} = C < \infty$$

Facts:

- (+) Quadratic convergence
- (-) convergence not guaranteed
- (-) if $f'(x_k) = 0$ it breaks
- (-) requires $f(x_k)$ and $f'(x_k)$
- (-) $f(x)$ needs to be differentiable
- linear convergence if root has multiplicity $m > 1$.

2.3 Secant Method

Algorithm: secante through x_k and x_{k-1}

$$x_{k+1} = x_k - f(x_k) \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \quad (24)$$

Convergence: $r = \varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$

Facts:

- (+) only $f(x)$ is needed, one per iteration
- (-) not quadratic convergence
- (-) two first approximations are needed

2.4 Set of Equations

Find the root of N non linear functions $f_i(\mathbf{x})$.

N = Number of Equations (f_i), M = Number of Variables (x_i)

$$F(\mathbf{x}^*) = \begin{pmatrix} f_1(\mathbf{x}^*) \\ f_2(\mathbf{x}^*) \\ \vdots \\ f_N(\mathbf{x}^*) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{O} \quad (25)$$

Taylor Expansion for Matrices and Vectors:

$$F(\mathbf{x} + \mathbf{y}) = F(\mathbf{x}) + J(\mathbf{x})\mathbf{y} + O(\|\mathbf{y}\|^2) \quad (26)$$

The Jacobian Matrix:

$$J(\mathbf{x}) = \begin{bmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_1(\mathbf{x})}{\partial x_M} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_2(\mathbf{x})}{\partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(\mathbf{x})}{\partial x_1} & \frac{\partial f_N(\mathbf{x})}{\partial x_2} & \dots & \frac{\partial f_N(\mathbf{x})}{\partial x_M} \end{bmatrix} \quad (27)$$

Condition Number:

$$\kappa = \|J^{-1}(\mathbf{x}^*)\| \quad (28)$$

2.4.1 Newtons Method

Algorithm:

$$J(\mathbf{x}_k)(\mathbf{x}_{k+1} - \mathbf{x}_k) = -F(\mathbf{x}_k), \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z} \quad (29)$$

$$A\mathbf{z} = \mathbf{b}, \quad A = J(\mathbf{x}_k) \quad \mathbf{b} = -F(\mathbf{x}_k) \quad (30)$$

Newton-Raphson method: ($N = M$)

$$\mathbf{x}_{k+1} = \mathbf{x}_k - J^{-1}(\mathbf{x}_k) F(\mathbf{x}_k) \quad (31)$$

In Practise we don't invert J and instead solve the following equation.

$$J(\mathbf{x}_k)\mathbf{z} = -F(\mathbf{x}_k), \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{z} \quad (32)$$

Facts:

- (+) Convergence is quadratic ($r = 2$) if J is not singular
- (-) cost is substantial. $O(N^2)$ for building J and $O(N^3)$ for solving the linear system.

Pseudo-Newton method: ($M \neq N$)

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J^{-1}(\boldsymbol{x}_k)F(\boldsymbol{x}_k) \tag{33}$$

Moore Penrose pseudo inverse matrix: J^{+}

$$J^{+} = \begin{cases} (J^T J)^{-1} J^T & \text{for } M > N \\ J^T (J J^T)^{-1} & \text{for } M < N \end{cases} \tag{34}$$

Modified Newton Method Instead of computing J every iteration we only use one.

$$J_1 \boldsymbol{z} = -F(\boldsymbol{x}_k), \quad J_0 = J(\boldsymbol{x}_0) \tag{35}$$

Gets rid of $O(N^3)$ and leaves us with $O(N^2)$. Only good if J doesn't change too rapidly.

2.5 Non Linear Optimization

We want to solve a minimization problem:

$$\boldsymbol{x}^* = \arg \min_{\boldsymbol{x}} E(\boldsymbol{x}) \tag{36}$$

$\boldsymbol{x} = (x_1, \dots, x_M)^T$ and $E: \mathbb{R} \rightarrow \mathbb{R}$

Maximisation equal to minimisation of $-E(\boldsymbol{x})$.

Sufficient Condition:

$$F(\boldsymbol{x}) = \nabla E(\boldsymbol{x}) = \left(\frac{\partial E}{\partial x_1}(\boldsymbol{x}^*), \dots, \frac{\partial E}{\partial x_M}(\boldsymbol{x}^*) \right)^T = \boldsymbol{0} \tag{37}$$

Critical Condition: Hessian matrix has to be positive definite!

$\nabla^2 E(\boldsymbol{x}) = H(\boldsymbol{x})$

$$H(\boldsymbol{x}^*) = \begin{pmatrix} \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1 \partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_M \partial x_1} & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_M \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_M^2} \end{pmatrix}$$

2.5.1 Newtons Method

Algorithm: $J(\boldsymbol{x}) = \nabla^2 E(\boldsymbol{x})$

$$\nabla^2 E(\boldsymbol{x}_k) \boldsymbol{z} = -\nabla E(\boldsymbol{x}), \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z} \tag{38}$$

A lot of computations and not guaranteed to converge.

Steepest Descent/Gradient Descent:

$$\boldsymbol{z} = -\nabla E(\boldsymbol{x}_k), \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \eta \boldsymbol{z} \tag{39}$$

Simpler, but slower than Newton. May only find local minimum. **Levenberg-Marquardt Method:**

$$\left(\nabla^2 E(\boldsymbol{x}_k) + \lambda I \right) \boldsymbol{z} = -\nabla E(\boldsymbol{x}), \quad \boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z} \tag{40}$$

Start with large λ (Gradient Descent), then decrease λ (Newtons Method). Often used for Non Linear Least Squares.

2.5.2 Non Linear Least Squares

We have data $\{(x_i, y_i)\}_{i=1}^N$ and want to fit a function $f(x)$.

$$f(x) = \sum_{k=1}^N \varphi_k(x; \boldsymbol{w}), \quad \boldsymbol{w} \in \mathbb{R}^M \quad \text{possible that: } M > K$$

Cost function:

$$E(\boldsymbol{w}) = \sum_{i=1}^N (y_i - f(x_i; \boldsymbol{w}))^2 = \|\boldsymbol{y} - F(\boldsymbol{x}; \boldsymbol{w})\|_2^2 \tag{41}$$

Gauss-Newton Method:

$$\boldsymbol{r}_i = y_i - f(x_i; \boldsymbol{w}) \tag{42}$$

$$D_{ij}(\boldsymbol{w}) = \frac{\partial f(x_i; \boldsymbol{w})}{\partial w_j} \tag{43}$$

$$D(\boldsymbol{w}_k)^T D(\boldsymbol{w}_k) \boldsymbol{z} = D(\boldsymbol{w}_k)^T \boldsymbol{r}(\boldsymbol{w}_k) \tag{44}$$

$$\boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \boldsymbol{z}$$

Steepest Descent Method:

$$\boldsymbol{z} = D(\boldsymbol{w}_k)^T \boldsymbol{r}(\boldsymbol{w}_k), \quad \boldsymbol{w}_{k+1} = \boldsymbol{w}_k + \nabla \boldsymbol{z} \tag{45}$$

3 Interpolation and Splines

3.1 Lagrange Interpolation

Fit a function of degree $N - 1$ to N data points. Therefore we need the **Lagrange Polynomial:**

$$\ell_k(x) = \prod_{\substack{i=1 \\ i \neq k}}^N \frac{x - x_i}{x_k - x_i} \tag{46}$$

Lagrange interpolation function:

$$f(x) = \sum_{i=1}^N y_k \ell_k(x) \tag{47}$$

Facts:

- Polynomials with degree $N - 1$ for N data points
- Interpolate data not extrapolate data
- Analytic expression from data points
- (-) Sensitive to noise
- (-) Predictability issues
- (-) High degrees give rise to huge oscillations
- (-) they are global and can't represent the local behaviour
- (-) small fluctuations in the data end in remodelling of the whole function

3.2 Cubic Splines

Locally defined cubic functions to represent the data. Given data $\{(x_i, y_i)\}_{i=0, \dots, N}$ with $x_i < x_{i+1}$. In every interval $[x_{i-1}, x_i]$, $i = 1, \dots, N$ we define a cubic function:

$$f_i(x) = \alpha_i x^3 + \beta_i x^2 + \gamma_i x + \delta_i, \quad i = 1, \dots, N \tag{48}$$

$4N$ unknowns \rightarrow **4 Constraints:**

- $f_i(x_{i-1}) = y_{i-1}$, ($i = 1, \dots, N$)
- $f_i(x_i) = y_i$, ($i = 1, \dots, N$)
- $f'_i(x_i) = f'_{i+1}(x_i)$, ($i = 1, \dots, N - 1$)
- $f''_i(x_i) = f''_{i+1}(x_i)$, ($i = 1, \dots, N - 1$)

We have $4N - 2$ constraints, we need 2 more.

Possible Conditions:

- Natural spline: Set $f''_1(x_0) = f''_N(x_N) = 0$
- Parabolic runout: Set $f''_1(x_0) = f''_1(x_1)$ and $f''_N(x_N) = f''_N(x_{N-1})$
- Clamping: Set $f'_1(x_0) = f'_N(x_N) = 0$

We can now solve the problem the following way:

$f''_i(x_i) = f''_{i+1}(x_i)$ gives us: ($i = 1, \dots, N - 1$)

$$f''_i(x) = \frac{a_{i-1}}{\Delta x_i} (x_i - x) + \frac{a_i}{\Delta x_i} (x - x_{i-1}) \tag{49}$$

$a_i = f''(x_i) = f''_i$ and $\Delta x_i = x_i - x_{i-1}$

From integrating the last constraint we get: ($i = 1, \dots, N$)

$$f_i(x) = a_{i-1} \frac{(x_i - x)^3}{6\Delta x_i} + a_i \frac{(x - x_{i-1})^3}{6\Delta x_i} + B_i(x - x_{i-1}) + C_i$$

Equations to be solved: ($i = 1, \dots, N - 1$)

$$\Delta x_i a_{i-1} + 2(\Delta x_i + \Delta x_{i+1}) a_i + \Delta x_{i+1} a_{i+1} = 6 \frac{\Delta y_{i+1}}{\Delta x_{i+1}} - 6 \frac{\Delta y_i}{\Delta x_i}$$

This ends up as a Matrix Equation where we want to find the vector \boldsymbol{a} :

$$\begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots \\ a_2 & b_2 & c_2 & 0 & \cdots \\ 0 & a_3 & b_3 & c_3 & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots \\ 0 & 0 & 0 & a_{N-1} & b_{N-1} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{N-1} \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{N-1} \end{bmatrix}$$

For $i = 0$ (a_0) and $i = N$ (a_N) we need to use the special conditions!

4 Numerical Integration

4.1 Numerical Quadrature

We split an interval $[a, b]$ into N Intervals $[x_i, x_{i+1}]$ of length

$\Delta_i = x_{i+1} - x_i$

Rectangle Rule:

$$I_{R_i} = f(x_i) \Delta_i \tag{50}$$

Midpoint Rule:

$$I_{M_i} = f\left(\frac{x_i + x_{i+1}}{2}\right) \Delta_i \tag{51}$$

Trapezoidal Rule:

$$I_{T_i} = \frac{f(x_i) + f(x_{i+1})}{2} \Delta_i \tag{52}$$

Simpsons Rule:

$$I_{S_i} = \frac{f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1})}{6} \Delta_i \tag{53}$$

4.2 Total Integrals

Rectangle Rules:

$$I \approx \Delta_i \sum_{i=0}^{N-1} f(x_i) \tag{54}$$

Midpoint Rule:

$$I \approx \Delta_i \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2}\right) \tag{55}$$

Trapezoidal Rule:

$$I \approx \frac{\Delta_i}{2} \left(f(x_0) + 2 \sum_{i=1}^{N-1} f(x_i) + f(x_N) \right) \tag{56}$$

Simpsons Rule:

$$I \approx \frac{\Delta_i}{3} \left(f(x_0) + 4 \sum_{\substack{i=1 \\ i=\text{odd}}}^{N-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=\text{even}}}^{N-2} f(x_i) + f(x_N) \right)$$

4.3 Newton Cotes Formula

We use $M + 1$ equidistant points in $[x_i, x_{i+1}]$ ($x_k = x_i + k h$) , ($k \in [0, M]$) and Lagrange Interpolations.

The Integrals:

$$I_i \approx \Delta_i \sum_{k=0}^M C_k^M f(x_k), \quad C_k^M = \frac{1}{\Delta_i} \int_{x_i}^{x_{i+1}} l_k^M(x) dx \tag{57}$$

$$l_k^M(x) = \prod_{\substack{i=0 \\ i \neq k}}^M \frac{(x - x_i)}{(x_k - x_i)} \tag{58}$$

Properties:

- $C_k^M = C_{M-k}^M$
- $\sum_{k=0}^M C_k^M = 1$

4.4 Error Analysis

Find an upper bound for our Integral. $E_{\text{rule}, i} = I_i - I_{\text{rule}, i}$

- **Rectangle Rule:** Second Order Accurate

$$E_{R_i} = \frac{1}{2} f'(x_i) \Delta_i^2 + \frac{1}{6} f''(x_i) \Delta_i^3 + \frac{1}{24} f'''(x_i) \Delta_i^4 + O(\Delta_i^5)$$

- **Midpoint Rule:** Third Order Accurate

$$E_{M_i} = \frac{1}{24} f''(x_{i+1/2}) \Delta_i^3 + O(\Delta_i^5) + \dots$$

- **Trapezoidal Rule:** Third Order Accurate

$$E_{T_i} = -\frac{1}{12} f''(x_{i+1/2}) \Delta_i^3 + O(\Delta_i^5) + \dots$$

- **Simpsons Rule:** Fifth Order Accurate ($I_{S_i} = \frac{2}{3} I_{M_i} + \frac{1}{3} I_{T_i}$)

$$E_{S_i} = -\frac{1}{90} f^{(4)} \Delta_i^5 \approx O(\Delta_i)^5$$

5 Richardson and Romberg

5.1 Richardson Extrapolation

A quantity of interest G is discretized by some grid spacing h : $G \approx G(h)$
For $h \rightarrow 0$ we should obtain the exact value G .

Expanding with a Taylor Series we get: (with $G(0) = G$)

$$G(h) = G(0) + c_1 h + c_2 h^2 + \dots \tag{59}$$

Halving h gives us $G(h/2)$:

$$G(h/2) = G + \frac{1}{2} c_1 h + \frac{1}{4} c_2 h^2 + \dots \tag{60}$$

Subtracting the two equations gives us:

$$G_1(h) = 2G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \dots \tag{61}$$

We increased the exponent of the leading error term, by just one subtraction.
General Case:

$$G_n(h) = \frac{1}{2^n - 1} (2^n G_{n-1}(h/2) - G_{n-1}(h)) = G + O(h^{n+1})$$

Error Estimation:

$$\epsilon(h/2) = G(h/2) - G(h) \tag{62}$$

For small h the estimation is good.

5.2 Romberg Integration

Improve an inaccurate, but "cheap" method, and improve it by using Richardson Extrapolation: $I_0^1, I_0^2, I_0^3, \dots$

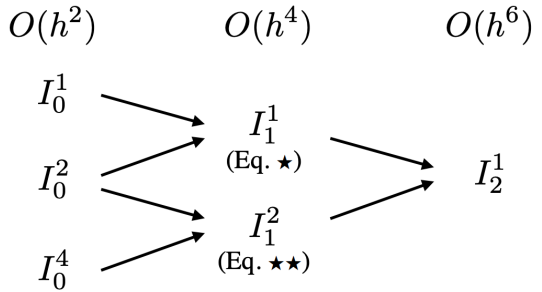
We start by using the Trapezoidal Rule and improve it.

Resulting Integral: (Trapezoidal Rule)

$$I_k^n = \frac{4^k I_{k-1}^{2n} - I_{k-1}^n}{4^k - 1} \tag{63}$$

Resulting Integral: (Simpsons Rule)

$$I_k^n = \frac{4^{k+1} I_{k-1}^{2n} - I_{k-1}^n}{4^{k+1} - 1} \tag{64}$$



6 Adaptive Quadrature

The main idea is to only subdivide the interval non-uniformly.

Algorithm 3 Adaptive integration.

Steps:

Subdivide the interval of the integration into sub-intervals
for all sub-intervals do
 Compute sub-integral, estimate the error with Richardson procedure described earlier.
 if accuracy is worse than desired **then**
 Subdivide the interval
 else
 Leave the interval untouched
 end if
end for

6.1 Gauss-Quadrature

Main Idea:

$$I = \int_a^b f(x) dx \approx \sum_i c_i \cdot f(x_i) \tag{65}$$

We want to choose c_i and x_i to minimize the error.

Method of undetermined coefficients: (only exact for a straight line)

$$I = \int_a^b f(x) dx = \int_a^b (a_0 + a_1 x) dx \approx c_1 f(a) + c_2 f(b)$$

Integration and comparing the coefficients we get $c_1 = c_2 = \frac{b-a}{2}$ (Trapezoidal Rule).

2-point Gauss Quadrature:

Same as above, but with variable function evaluation points.

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$$

(66)

with $f(x) = a_0 + a_1x + a_2x^2 + a_3x^3$.
Solving it the same way as above we get:

$$I \approx \frac{b-a}{2} \cdot f\left[\left(\frac{b-a}{2}\right)\left(\frac{-1}{\sqrt{3}}\right) + \frac{b+a}{2}\right]$$
$$+ \frac{b-a}{2} \cdot f\left[\left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}\right]$$

For higher order Gauss Quadrature, we need Hermite Interpolation and Legendre Polynomials.

6.2 Hermite Interpolation

Interpolate the values y_k and the derivatives of the data/function y_k' .

$$f(x) = \sum_{k=1}^n U_k(x)y_k + \sum_{k=1}^n V_k(x)y_k'$$

(67)

U_k and V_k are polynomials of degree $2n-1$ with the following properties.

$$U_k(x_j) = \delta_{jk}, \quad U_k'(x_j) = 0, \quad V_k(x_j) = 0, \quad V_k'(x_j) = \delta_{jk}$$

In Terms of the Lagrange Polynomial we get the followoing functions:

$$U_k(x) = \left[1 - 2L_k'(x_k)(x - x_k)\right] L_k^2(x)$$

(68)

$$V_k(x) = (x - x_k)L_k^2(x)$$

(69)

6.3 n-point Gauss Quadrature

We move the interval $[a, b]$ to $[-1, 1]$. ($x \in [a, b] \rightarrow z \in [-1, 1]$)

$$z = \frac{2x - (a + b)}{b - a}$$

(70)

We then get the following integral: (rearrange the above equation)

$$I = \int_a^b f(x)dx = \int_{-1}^1 \frac{b-a}{2} f\left(\frac{b-a}{2}(z-1) + b\right) dz$$

(71)

Approximating a general function $f(x)$ with Hermite Polynomials we get:

$$\int_{-1}^1 f(x)dx = \sum_{k=1}^n y_k \int_{-1}^1 U_k(x)dx + \sum_{k=1}^n y_k' \int_{-1}^1 V_k(x)dx$$

(72)

$$\int_{-1}^1 f(x)dx = \sum_{k=1}^n u_k y_k + \sum_{k=1}^n v_k y_k'$$

(73)

with $u_k = \int_{-1}^1 U_k(x)dx$ and $v_k = \int_{-1}^1 V_k(x)dx = 0 \forall k$

Resulting Integral: (u_k is tabulated)

$$I = \int_{-1}^1 f(x)dx = \sum_{k=1}^n u_k f(x_k)$$

(74)

$$u_k = \frac{2}{(1-x_k^2)(P_n'(x_k))^2}$$

(75)

In Practice we get: ($z \in [-1, 1]$)

$$I = \int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^n w_i f\left(\frac{b-a}{2}(z_i-1) + b\right)$$

(76)

w_i is tabulated

Error with n abscissas:

$$\varepsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\xi)$$

(77)

7 Monte Carlo

7.1 Multidimensional Quadrature

Until now we only looked at one dimensional functions. If we use functions with an arbitrary dimension $f: \mathbb{R}^d \rightarrow \mathbb{R}$, $I = \int_{\Omega} f(x^{(1)}, \dots, x^{(d)}) dx$

$$I \stackrel{\text{Fubinis theorem}}{=} \int_{\Omega} \dots \int_{\Omega_d} f(x^{(1)}, \dots, x^{(d)}) dx^{(d)} \dots dx^{(1)}$$
$$\approx \sum_{i_1=1}^{N_1} \dots \sum_{i_d=1}^{N_d} \underbrace{w_{i_1} \dots w_{i_N}}_{= \underline{\underline{W}}_{i_1, \dots, i_d}} f(x_{i_1}^{(1)}, \dots, x_{i_d}^{(d)})$$

(78)

$\underline{\underline{W}} = \vec{w} \vec{w}^T$ (2D) where \vec{w} is the $N \times 1$ dimensional weight vector as specified by section ??
In general, $\underline{\underline{W}} \in N_1 \times \dots \times N_d$

7.2 Curse of dimensionality

The error of that Integral will scale as follows:

$$\mathcal{O}(M^{-s/d})$$

(79)

with $M = n^d$