1 Lagrange and Least Squares

1.1 Function Fitting

Fitting function $f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x)$

With possible basis functions: $\boxed{\begin{array}{ll} \phi_k(x) = x^{k-1} & \phi_k(x) = \cos[(k-1)x] \\ \phi_k(x) = e^{\beta_k x} & \phi_k(x) = 1 - \frac{|x-x_k|}{\Delta} \end{array}}$

1.2 Linear Interpolation $N = M$

Problem statement: $f(x_i) = \sum_{k=1}^{M} \phi_k(x_i)\alpha_k = y_i, \quad i = 1 \dots N$
To determine the coefficients, we solve the following system equations:

$$\Phi\vec{\alpha} = \vec{y}$$

$$\begin{pmatrix} \phi_1(x_1) & \phi_2(x_1) & \dots & \phi_M(x_1) \\ \phi_1(x_2) & \phi_2(x_2) & \dots & \phi_M(x_2) \\ \vdots & & & \vdots \\ \phi_1(x_N) & \phi_2(x_N) & \dots & \phi_M(x_N) \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_M \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

1.2.1 Lagrange Interpolation

Idea: Create basis functions such that the matrix above becomes the identity matrix.

- Polynomials of same degree as the number of data points
- Sensitive to noise
- Predictability issues
- Passes through all the points
- Lagrange polynomials form a basis of P_{N-1}
- Choose x_n to be the roots of the Chebychev polynomials.

Construct N polynomials of degree $N-1$: $\{l_k(x),\ k = 1, \dots, N\}$, such that $l_k(x_i) = \delta_{ki}$.

$$\boxed{l_k(x) = \prod_{\substack{1 \leqslant i \leqslant N \\ i \neq k}} \frac{x - x_i}{x_k - x_i} \quad \rightarrow \quad f(x) = \sum_{k=1}^{N} y_k l_k(x)}$$

$$\boxed{l_k(x,y) = \prod_{\substack{1 \leqslant i \leqslant N \\ i \neq k}} \frac{(x-x_i)^2 + (y-y_i)^2}{(x_k-x_i)^2 + (y_k-y_i)^2}} \text{ Lagrange in 3D}$$

$$\boxed{|y(x) - f(x)| = \left| \frac{y^{(n)}(\xi)}{n!} \Pi_{k=1}^{n}(x - x_k) \right|, \ x_1 \leq \xi \leq x_N} \text{ Error}$$

1.3 Least Squares $N < M$

- Low order model of data
- Less sensitive to noise
- Higher computational complexity

Equation system, N data points, M unknown parameters: $A\vec{x} = \vec{b}$
If the system is inconsistent, $\vec{b} \notin \text{span}(A)$. We are looking for $\vec{x}$ which minimizes the error: $E = \|A\vec{x} - \vec{b}\|$.

$\rightarrow A\vec{x} - b$ has to be perpendicular to the column space of A: $A\vec{y}$.
$(A\vec{y})^T(A\vec{x} - b) = 0 \Rightarrow y^T[A^TA\vec{x} - A^T\vec{b}] = 0$

$$\boxed{A^TA\vec{x} = A^T\vec{b} \qquad \Rightarrow \vec{x} = (A^TA)^{-1}A^Tb}$$

1.3.1 Least Squares Fitting of 3-D Data

This example can easily be applied to the 2-D data case.

Given set of N data $\{x_i, y_i, z_i\}_{i=1}^{N}$, we wish to fit to a linear function of 3 parameters A, B and C given by:

$$A + Bx_i + Cy_i \approx z_i \quad \text{or} \quad \begin{pmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & y_N \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} \approx \begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_N \end{pmatrix}$$

$A^TA\bar{x} = A^T\vec{b}$ leads to the final system of equation (2D, $z_i \rightarrow y_i$):

$$\begin{pmatrix} N & \sum x_i & \sum y_i \\ \sum x_i & \sum x_i^2 & \sum x_i y_i \\ \sum y_i & \sum x_i y_i & \sum y_i^2 \end{pmatrix} \begin{pmatrix} A \\ B \\ C \end{pmatrix} = \begin{pmatrix} \sum z_i \\ \sum z_i x_i \\ \sum z_i y_i \end{pmatrix}$$

1.3.2 The Projection Matrix P

Closest point to $\vec{b}$ is:

$$A\bar{x} = \vec{p} = A(A^TA)^{-1}A^T\vec{b} \quad \rightarrow \quad P = A(A^TA)^{-1}A^T$$

The matrix P is idemptotent: $P^2 = P$ and symmetric: $P = P^T$.
Orthogonal case: $A^TA = \mathbb{I}$

2 Splines

2.1 Cubic Splines

We try to fit a cubic function $f_i(x)$ between two data points with $f_i' = f_{i+1}'$ and $f_i'' = f_{i+1}''$ at each node $\{x_i, y_i\}$. $(i = 1 \cdots N)$

2_{nd} derivative is a piecewise linear function $\rightarrow$ integrate twice:

$$f(x) = f_i'' \frac{(x_{i+1} - x)^3}{6\Delta_i} + f_{i+1}'' \frac{(x - x_i)^3}{6\Delta_i} + C_i(x - x_i) + D_i$$

evaluate at $\{x_i, y_i = f(x_i)\}$ and $\{x_{i+1}, y_{i+1} = f(x_{i+1})\}$ to get:

$$C_i = \left(\frac{y_{i+1} - y_i}{\Delta_i} - (f_{i+1}'' - f_i'')\frac{\Delta_i}{6} \right) \qquad D_i = \left(y_i - f_i'' \frac{\Delta_i^2}{6} \right)$$

for $x_i \leq x \leq x_{i+1}$:

$$f'(x) = f_{i+1}'' \left[\frac{(x - x_i)^2}{2\Delta_i} - \frac{\Delta_i}{6} \right] - f_i'' \left[\frac{(x_{i+1} - x)^2}{2\Delta_i} - \frac{\Delta_i}{6} \right] + \frac{y_{i+1} - y_i}{\Delta_i}$$

for $x_{i-1} \leq x \leq x_i$:

$$f'(x) = f_i'' \left[\frac{(x - x_{i-1})^2}{2\Delta_{i-1}} - \frac{\Delta_{i-1}}{6} \right] - f_{i-1}'' \left[\frac{(x_i - x)^2}{2\Delta_{i-1}} - \frac{\Delta_{i-1}}{6} \right] + \frac{y_i - y_{i-1}}{\Delta_{i-1}}$$

With $\Delta_i = x_{i+1} - x_i$, $f''(x_i) = f_i''$ and the condition that $f'(x)$ must be continuous $f'(x_i) = f'(x_i)$ the equation that must be solved is:

$$\frac{\Delta_{i-1}}{6} f_{i-1}'' + \left(\frac{\Delta_{i-1} + \Delta_i}{3}\right) f_i'' + \frac{\Delta_i}{6} f_{i+1}'' = \frac{y_{i+1} - y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}}$$

$$\Rightarrow \quad A_i f_{i-1}'' + B_i f_i'' + C_i f_{i+1}'' = D_i$$

Boundary conditions:

- Natural spline: $f_1'' = f_N'' = 0$
- Splines with clamped boundaries: $f_1' = f_N' = 0$
- Parabolic runout: $f_1'' = f_2''$ and $f_N'' = f_{N-1}''$
- General equation not valid for end points, except for periodic case

Example of system with $N = 4$ and clamped boundaries:

$$\begin{pmatrix} B_1 & C_1 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 \\ 0 & A_3 & B_3 & C_3 \\ 0 & 0 & A_4 & B_4 \end{pmatrix} \begin{pmatrix} f_1'' \\ f_2'' \\ f_3'' \\ f_4'' \end{pmatrix} = \begin{pmatrix} D_1 \\ D_2 \\ D_3 \\ D_4 \end{pmatrix}$$

- Goes through all control points.
- Only uses lower order polynomials.
- Continuous 2_{nd} derivative.

2.2 B-splines and NURBS

General "B-spline approach" with free parameters α_i to be determined:

$$S_{d,t}(x) = \sum_{i=1}^{M} \alpha_i B_{i,d,t}(x) \qquad \text{Find } \alpha_i \text{ with linear least squares.}$$

N=M forces $S_{d,t}(x)$ to go through all points. Change one point $\rightarrow$ recompute all coefficients.
The B-spline $B_{i,d,t}(x)$ is itself constructed as piecewise polynomials of degree d which is only non-zero for the range $t_i \leq x \leq t_{i+d+1}$. N data points lead to N splines $(i = 1, \dots, N)$ with continuous derivatives up to degree d-1. The size of the knot vector t is therefore $N + d + 1$.

$$B_{i,0,t}(x) = \begin{cases} 1 & \text{if } t_i \leq x \leq t_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$B_{i,d,t}(x) = \frac{x - t_i}{t_{i+d} - t_i} B_{i,d-1,t}(x) + \frac{t_{i+d+1} - x}{t_{i+d+1} - t_{i+1}} B_{i+1,d-1,t}(x)$$

NURBS (Non-Uniform Rational B-Splines)

N data points $\vec{p}_i = \{x_i, y_i\}, (i = 1, \dots, N)$ each weighted by w_i. The curve is parametrized as $\vec{p}(s) = \{x(s), y(s)\}$ for $t_{d+1} \leq s \leq t_{N+1}$ as follows:

$$\vec{p}(s) = \sum_{i=1}^{N} R_{i,d,t}(s)\vec{p}_i \quad \text{with} \quad R_{i,d,t}(s) = \frac{B_{i,d,t}(s)w_i}{\sum_{j=1}^{N} B_{j,d,t}(s)w_j}$$

If B-splines with **"clamped"** knots are used, the curve is guaranted to start at the first control point and end at the final one.

$$t = \{\underbrace{t_1, \dots, t_{d+1}}_{\substack{\text{d+1} \\ \text{equal knots}}}, \underbrace{t_{d+2}, \dots, t_N}_{\text{N-d knots}}, \underbrace{t_{N+1}, \dots, t_{N+d+1}}_{\substack{\text{d+1} \\ \text{equal knots}}}\}$$

Interval between two equal nodes is trivial. Does not produce a new spline.

2.3 Multivariate Interpolation

Given $z_k = Z(x_k, y_k)$ for $k = 1, \dots, N$ we must find a reasonable function f(x,y) such that $z_k = f(x_k, y_k)$

2.3.1 Gridded Data

In the case of gridded data we can use as functions the tensor products of functions in 1-dimension.

$$f(x_p, y_q) = Z(x_p, y_q) = \sum_i \sum_j \alpha_{ij} \cdot \phi_i(x_p) \cdot \phi_j(y_q)$$

Using Lagrange polynomials:

$$f(x_p, y_q) = \sum_i \sum_j Z(x_i, y_j) \cdot l_i(x_p) \cdot l_j(y_q)$$

Using NURBS surfaces: We define a "grid" of $N \times M$ control points $\vec{p}_{ij} = \{x_{ij}, y_{ij}, z_{ij}\}$. The two dimensions of the grid of control points correspond to two parametric dimensions u and v. For each parametric dimension we fix the degree to be d_U and d_V, and define knot vectors t_U and t_V.

$$\vec{p}(u, v) = \sum_{i=1}^{N} \sum_{j=1}^{M} R_{i,d_U,t_U}(u) \cdot R_{j,d_V,t_V}(v) \cdot \vec{p}_{ij}$$

3 Choosing basis functions

3.1 Orthogonal Functions

Definition: $\langle \phi_i \phi_j \rangle = \int_{-\infty}^{\infty} \phi_i \phi_j dx = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & \text{otherwise} \end{cases}$

Advantage: enable to add additional basis functions without recomputing previous parameters. Functional representation of $y(x)$ is given by

$$y(x) = \sum_{i=1}^{M} \alpha_i \phi_i(x)$$

with the coefficients α_i and orthonormal basis functions ϕ_i:

$$\alpha_i = \int_{-\infty}^{\infty} y(x) \phi_i(x) dx$$

With a set of **experimental observations** $\{x_i, y_i\}_{i=1,\dots,N}$ the basis functions must be orthonormal with respect of the probability density $p(x)$:

$$p(x) \approx \frac{1}{N} \sum_{n=1}^{N} \delta(x - x_n)$$

$$\rightarrow \langle \phi_i(x), \phi_j(x) \rangle = \int_{-\infty}^{\infty} \phi_i(x) \phi_j(x) p(x) dx = \delta_{ij}$$

Which leads to a new formula for computing the coefficients and to the general discrete inner product:

$$\alpha_i \approx \frac{1}{N} \sum_{n=1}^{N} y_n \phi_i(x_n)$$

$$\langle f(x), g(x) \rangle = \frac{1}{N} \sum_{i=1}^{N} f(x_i) g(x_i)$$

Possible basis functions:

- Hermite polynomials, orthogonal with respect to e^{-x^2}
- Laguerre polynomials, orthogonal with respect to e^{-x}
- Chebyshev polynomials, orthogonal with respect to $(1 - x^2)^{-\frac{1}{2}}$

3.1.1 Gram-Schmidt process

- Normalize first basis function: $\phi_1 = \frac{g_1}{\langle g_1(x), g_1(x) \rangle^{1/2}}$
- Find orthogonal function $\tilde{\phi}_2$ and normalize it to ϕ_2

  $\tilde{\phi}_2 = g_2(x) - \phi_1(x) \frac{1}{N} \sum_{n=1}^{N} \phi_1(x_n) g_2(x_n)$

  $\phi_2 = \frac{\tilde{\phi}_2}{\langle \tilde{\phi}_2(x), \tilde{\phi}_2(x) \rangle^{1/2}}$
- Find third orthogonal function etc.

  $\tilde{\phi}_3 = g_3(x) - \phi_1(x) \frac{1}{N} \sum_{n=1}^{N} \phi_1(x_n) g_3(x_n) - \phi_2(x) \frac{1}{N} \sum_{n=1}^{N} \phi_2(x_n) g_3(x_n)$

3.2 Radial Basis Functions

Advantages:

- Extra terms added without increased divergence (since all basis functions are identical)
- Centers can be placed where needed

We choose a set of identical basis functions ϕ, which depend on the distance from a set of "centers" c_i and on parameters α_i:

$$y(x) = \sum_{i=1}^{N} \phi(|x - c_i|, \alpha_i) \xrightarrow{\text{if linear}} y(x) = \sum_{i=1}^{N} \alpha_i \phi(|x - c_i|)$$

Issues:

- Choice of ϕ, for example:

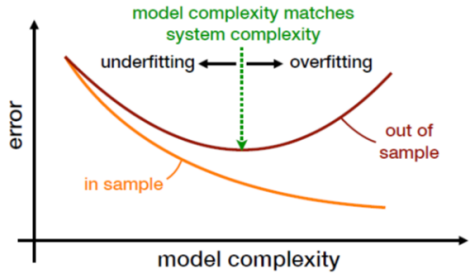
  $$\phi(r) = r \qquad \phi(r) = r^n \qquad \phi(r) = e^{-r^2}$$

  3-D Gaussian RBF:

  $$\phi_i(x, y) = \frac{1}{2\pi\sigma_x\sigma_y} \exp\left\{ -\frac{1}{2} \left(\frac{(x - c_{xi})^2}{\sigma_x^2} + \frac{(y - c_{yi})^2}{\sigma_y^2} \right) \right\}$$
- Choice of c_i: Randomly, Uniformly or Data based?
- Linear or non-linear coefficients: For non-linear coefficients, iterations are necessary. For linear we need to solve a system as we did for Least Squares. Example:

$$\begin{pmatrix} |x_1 - c_1|^3 & |x_1 - c_2|^3 & \dots & |x_1 - c_M|^3 \\ |x_2 - c_1|^3 & |x_2 - c_2|^3 & \dots & |x_2 - c_M|^3 \\ \vdots & & & \vdots \\ |x_N - c_1|^3 & |x_N - c_2|^3 & \dots & |x_N - c_M|^3 \end{pmatrix} \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_M \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix}$$

3.3 Overfitting / Underfitting



Using too many or too few parameters → we need a method to select the right model → Cross validation.

3.4 Cross Validation

- Divide the training data $Z = \{x_i, y_i\}_{i=1,\dots,N}$ to k disjoint samples of roughly equal size $(N/K), Z_1, \dots, Z_K$
- For each validation sample Z_i
  - Use the remaing data $Z_l = \cup_{j \neq i} Z_i$ to construct an estimate of the model f_i
  - For the estimated model f_i, average the square errors for the data in Z_i:

    $$r_i = \frac{K}{N} \sum_{Z_i} (f_i(x) - y)^2$$
- Compute the estimate for the prediction error by averaging the r_i for $Z_1, \dots, Z_k$

  $$R = \frac{1}{K} \sum_{i=1}^{K} r_i$$

If K=N we do leave one out cross validation.

4 Non-Linear Equations

General approach: 1. Bring the non-linear equation to the form $f(x) = 0$, where x is the unknown parameter. 2. Find the root of $f(\vec{x})$.

4.1 Conditioning for Root finding problems

If $\frac{1}{|f'(x^*)|}$ is small, the problem is well conditioned. Thus if $f(\tilde{x}) \approx 0$, $\tilde{x}$ is sure to be close to x^*.

$\tilde{x}$ calculated root, x^* true root

4.2 Convergence rate

Many schemes proceed by iteratively improving an estimate $x^{(k)} \approx x^*$.

Error: $E^{(k)} = x^{(k)} - x^*$

Sequence of $x^{(k)}$ converges with rate r if $\lim_{k \to \infty} \frac{\|E^{(k+1)}\|}{\|E^{(k)}\|^r} = C$

- $r = 1$: linear
- $r > 1$: superlinear
- $r = 2$: quadratic

4.3 Bisection Method

The method assumes, when $f(a)$ is negative and $f(b)$ positive, the root of $f(x)$ must be in the range $[a, b]$. With this starting interval $[a, b]$ after k iterations, the interval is $(b-a)/2^k$, so achieving an error tolerance of tol requires $m = \log_2((b-a)/tol)$ steps, regardless of f.

Algorithm: Bisection Method

while $(b-a) > tol$ **do**
  $m \leftarrow (a+b)/2$
  if $sign(f(a)) = sign(f(m))$ **then**
    $a \leftarrow m$
  else
    $b \leftarrow m$
  end if
end while

- Certain to converge but slow.
- Interval is cut in half every time.
- Linear convergence (r=1), C=0.5.
- Achieving error tolerance of tol: steps=$\log_2((b-a)/tol)$.

4.4 Newton's Method

Take a first guess for $x^{(k)}$ so that $f(x^{(k)})$ is "somewhere" around 0. Iterate several times with

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} \qquad \epsilon_k = \|x_k - x_{k-1}\|$$

simple roots: $(x-1)^1 \to r = 2$, higher multiplicity: $(x-1)^2 + \to r = 1$
until $f(x^{(k+1)})$ goes below a certain tolerance. This method is derived by the taylor series $f(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)})$. Since evaluating the derivation may be expensive or inconvenient, we can replace it with a finite difference approximation, the **Secant Method**.

$$f'(x^{(k)}) = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \qquad r = 1.618$$

4.4.1 Solving systems of non-linear equations

$$\vec{F}(\vec{x}) = \begin{pmatrix} f_1(\vec{x}) \\ \vdots \\ f_N(\vec{x}) \end{pmatrix} = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} = \vec{0}$$

The derivation $f'(x)$ is replaced by the Jacobian matrix J which is a $N \times N$ matrix with elements $J(\vec{x})_{ij} = \partial f_i(\vec{x})/\partial x_j$

$$J(\vec{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1}(\vec{x}) & \dots & \frac{\partial f_1}{\partial x_N}(\vec{x}) \\ \vdots & \ddots & \vdots \\ \frac{\partial f_N}{\partial x_1}(\vec{x}) & \dots & \frac{\partial f_N}{\partial x_N}(\vec{x}) \end{pmatrix}$$

So for the iteration process we get the equation:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} - J^{-1}(\vec{x}^{(k)})\vec{F}(\vec{x}^{(k)})$$

For not computing the inverse of J we can solve

$$J(\vec{x}^{(k)})\vec{y}^k = -\vec{F}(\vec{x}^{(k)}) \quad \xrightarrow{\text{update}} \quad \vec{x}^{(k+1)} = \vec{x}^{(k)} + \vec{y}^k$$

Algorithm: Newton's Method

Input:
$\vec{x}^{(0)}$, {vector of length N with initial approximation}
tol, {tolerance: stop if $\|\vec{x}^{(k)} - \vec{x}^{(k-1)}\| < tol$ }
k_{max}, {maximal number of iterations: stop if $k > k_{max}$}
Output:
$\vec{x}^{(k)}$, {solution of $\vec{F}(x^{(k)}) = \vec{0}$ within tolerance tol}
Steps:
$k \leftarrow 1$
while $k \leq k_{max}$ **do**
  Calculate $\vec{F}(\vec{x}^{(k-1)})$ and $N \times N$ matrix $J(\vec{x}^{(k-1)})$
  Solve the $N \times N$ linear system $J(\vec{x}^{(k-1)})\vec{y} = -\vec{F}(\vec{x}^{(k-1)})$
  $x^{(k)} \leftarrow \vec{x}^{(k-1)} + \vec{y}$
  if $\|\vec{y}\| < tol$ **then**
    break
  end if
  $k \leftarrow k + 1$
end while

4.4.2 Simplifications to reduce cost

- **Modified Newton Method:** Compute $J^{(0)} = J(x^{(0)})$ only once and solve $J^{(0)}\vec{y}^k = -\vec{F}(\vec{x}^{(k)})$. This method can only succeed if J is not changing rapidly.

- **Quasi Newton Method:** J gets updated by using information from the steps. After the first step we know: $\Delta\vec{x} = \vec{x}^{(1)} - \vec{x}^{(0)}$ and $\Delta\vec{F} = \vec{F}(\vec{x}^{(1)}) - \vec{F}(\vec{x}^{(0)})$. So derivatives of the f_i are in the direction of $\Delta\vec{x}$. Then the next $J^{(1)}$ is adjusted to satisfy:

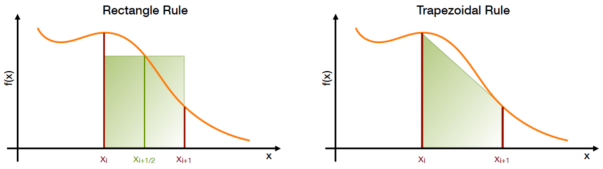
$$J^{(1)}\Delta\vec{x} = \Delta\vec{F}$$

$$J^{(1)} = J^{(0)} + \frac{(\Delta\vec{F} - J^{(0)}\Delta\vec{x})(\Delta\vec{x})^T}{(\Delta\vec{x})^T(\Delta\vec{x})}$$

5 Numerical Integration

If integral cannot be solved analytically, the function is known as a set of points.

$$I = \int_a^b f(x)dx \quad = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx \quad \approx \sum_{i=0}^{N-1} I_i$$



Rectangle Rule: Uses 2 intervals

$$I_{R_i} = f(\frac{x_i + x_{i+1}}{2})\Delta_i \quad \text{with } \Delta_i = x_{i+1} - x_i$$

Trapezoidal Rule:

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

Simpson's Rule: Uses 2 intervals

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

With constant Δ_x all three formulas can be written as a weighted sum of f_i: $I \approx \sum_{i=0}^{N} w_i f_i$.

$$I_R \approx 2\Delta_x \sum_{\substack{i=1 \\ i=\text{odd}}}^{N-1} f_i \qquad I_T \approx \frac{\Delta_x}{2}\left(f_0 + 2\sum_{i=1}^{N-1} f_i + f_N\right)$$

$$I_S \approx \frac{\Delta_x}{3}\left(f_0 + 4\sum_{\substack{i=1 \\ i=\text{odd}}}^{N-1} f_i + 2\sum_{\substack{i=1 \\ i=\text{even}}}^{N-2} f_i + f_N\right)$$

Newton-Cotes formulas: approximate the function using Lagrange polynomials, with n + 1 equidistant points in $[a, b]$

$$I \approx (b-a)\sum_{k=0}^{n} C_k^n f(x_k) \quad \text{with } C_k^n = \frac{1}{b-a}\int_a^b l_k^n(x)dx$$

$\sum_{k=0}^{n} C_k^n = 1 \qquad C_k^n = C_{n-k}^n$ Properties of C_k^n

Because Lagrange polynomials fit $f(x) = 1$ perfectly $\to \frac{I}{(b-a)} = 1 = \sum C_k \cdot 1$ for n=2 we get Simpsons rule

5.1 Error analysis

Taylor's series around $x_{i+0.5}$:
$f(x) = f(x_{i+0.5}) + (x - x_{i+0.5})f'(x_{i+0.5}) + \frac{1}{2}(x - x_{i+0.5})^2 f''(x_{i+0.5}) + \frac{1}{6}(x - x_{i+0.5})^3 f'''(x_{i+0.5}) + \dots$
$I_i = \int_{x_i}^{x_{i+1}} f(x)dx = \underbrace{f(x_{i+1/2})\Delta_i}_{I_{R_i}} + \underbrace{0 + \frac{1}{24}f''(x_{i+1/2})\Delta_i^3 + 0 + O(\Delta_i^5)}_{\text{error of rectangle rule}}$
$I_{T_i} = \frac{f(x_i)+f(x_{i+1}}{2}\Delta_i = \Delta_i\left(f(x_{i+1/2}) + \frac{1}{8}f''(x_{i+1/2})\Delta_i^2 + \cdots\right)$

When we consider the entire Taylor series in our scheme of integration, we can evaluate the error for the different methods:

$$I_{R_i} = I_i - \frac{1}{24}f''(x_{i+0.5})\Delta_i^3 + \mathcal{O}(\Delta_i^5) + \dots$$
$$I_{T_i} = I_i + \frac{1}{12}f''(x_{i+0.5})\Delta_i^3 + \mathcal{O}(\Delta_i^5) + \dots$$
$$I_{S_i} = \frac{2}{3}I_{R_i} + \frac{1}{3}I_{T_i} = I_i + \mathcal{O}(\Delta_i^5) + \dots$$

For the whole domain: where $N = (b-a)/\Delta_x$

$$\sum_{i=0}^{N-1} I_{Ri} = \sum_{i=0}^{N-1}\left(I_i - \frac{1}{24}f''(x_{i+1/2})\Delta_x^3 + O(\Delta_x^5) + \cdots\right)$$
$$\left|\sum_{i=0}^{N-1} I_{Ri} - I\right| < N\frac{1}{24}\max_i\left(|f''(x_{i+1/2})|\right)\Delta_x^3 + NO(\Delta_x^5) + \cdots$$
$$= \frac{b-a}{24}\max_i\left(|f''(x_{i+1/2})|\right)\Delta_x^2 + O(\Delta_x^4) + \cdots$$

5.2 Richardson Extrapolation and Error Estimation

Suppose we have a method of approximating a quantity G. The approximations depend on a parameter h so that we write $G \approx G(h)$, which can be expressed in terms of a Taylor series for h:

$$G(h) = G + c_1 h + c_2 h^2 + \dots$$
$$G(h/2) = G + \frac{1}{2}c_1 h + \frac{1}{4}c_2 h^2 + \dots$$
$$\Rightarrow G_1(h) = 2G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \dots$$

In general: $G_n(h) = \frac{1}{2^n - 1}\left(2^n G_{n-1}(h/2) - G_{n-1}(h)\right) = G + \mathcal{O}(h^{n+1})$

$$\epsilon(h/2) = G(0) - G(h/2) = \quad -\tfrac{1}{2}c_1 h - \tfrac{1}{4}c_2 h^2 + O(h^3)$$
this is similar to $G(h/2) - G(h) = \quad -\tfrac{1}{2}c_1 h - \tfrac{3}{4}c_2 h^2 + O(h^3)$
$\rightarrow$ to 1_{st} order $\epsilon(h/2) \approx G(h/2) - G(h)$

If h is small, this will be a good estimate of the error. If the error is not small enough, then this tells the user to keep subdividing.

5.3 Romberg integration

Improve inaccurate integration methods by using Richardson's extrapolation. Using a set of trapezoidal approximations I_0^n for $n = 1, 2, 4, 8, \dots$ (n intervals), the method goes as follows:

$$I_0^n = \frac{b-a}{2n}\left(f(a) + f(b) + 2\sum_{j=1}^{n-1} f\left(a + j\frac{b-a}{n}\right)\right)$$

Then we recursively calculate the higher order approximations according to the following expression:

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

Note: 1. How many initial integrals I_0^k we need, depends on the accuracy we want to obtain. If we want to drop the first order $O(h^2)$ term for instance, we only have to compute two initial approximations I_0^1 and I_0^2.
2. The function values of $f(x)$ must not be recomputed in every step. Think of the highest n you will need and calculate all values for this particular case. Then store them for the next computations of $I_0^{n_{\max}/2}, I_0^{n_{\max}/4}, \dots, I_0^1$. **3.** Every iteration the accuracy increases by a factor of 2.

5.4 Adaptive Quadrature

Algorithm: Adaptive integration

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

Algorithm: Adaptive integration using recursion and Simpson

function adaptiveSimpson(a, b)
  apply Simson's rule in interval $[a, b]$
  subdivide the interval into $[a, m]$ and $[m, b]$ with $m = (a+b)/2$
  apply Simpson's rule in intervals $[a, m]$ and $[m, b]$
  estimate error in $[a, b]$ using Richardson's extrapolation
  if accuracy is worse than desired **then**
    return adaptiveSimpson(a, m) + adaptiveSimpson(m, b)
  else
    return value of Simpson's rule (the accurate one)
  end if
end function

5.5 Hermite Interpolation

Enhancement of Lagrance polynomials $\rightarrow$ interpolate derivative too. Given y_i, y_i' find $f(x)$ such that $f(x_i) = y_i$ and $f'(x_i) = y_i'$.

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

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

solution: $U_k(x) = \quad [1 - 2L_k'(x_k)(x - x_k)]L_k^2(x)$, $V_k(x) = \quad (x - x_k)L_k^2(x)$

5.6 Gauss Quadrature

Good for smooth functions, if integrand is non-smooth divide in smooth parts.
Optimal integration points $\rightarrow$ high accuracy
Method of indeterminate coefficients: $\int_a^b f(x)dx \approx \sum \omega_i f(x_i)$
Thus n abscissas and n weights adjustable. Polynomials of degree $2n-1$ should be exactly integrable. How to find those?

- Change the boundary of the integral: $z = \frac{2x}{b-a} + 1 - \frac{2b}{b-a}$.
- If z_i and ω_i given proceed with:

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

  otherwise they are found as follows (proof):
- $\int_{-1}^{1} f(x)dx = \int_{-1}^{1} H_I(x)\omega(x)dx = \sum_{k=1}^{n} u_k f(x_k) + \sum_{k=1}^{n} v_k f'(x_k)$
  $u_k = \int_{-1}^{1} \omega(x) U_k(x)dx \qquad v_k = \int_{-1}^{1} \omega(x) V_k(x)dx$
- $v_k \stackrel{!}{=} 0$ to get the form $I = \int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} \omega_i f(x_i)$ otherwise y_i' which are generally unavailable would be needed.
- $L_k = \frac{C_k F(x)}{(x - x_k)} \rightarrow v_k = C_k \int_{-1}^{1} F(x) L_k dx = 0$
  $C_k = \prod_{i=1}^{n} \frac{1}{(x_i - x_k)} \qquad F(x) = \prod_{i=1}^{n} (x - x_i)$
  Therefore F(x) must be orthogonal to any L_k. The only solution to that is given by the Legendre polynomials $P_n(x)$.

$$\boxed{\int_{-1}^{1} P_n(x)P_m(x)dx = \tfrac{2}{2n+1}\delta_{nm}} \text{ Orthogonality}$$

- x_k are the zeros of $P_n(x)$. $u_k = \frac{2}{(1 - x_k^2)(P_n'(x_k))^2}$
  u_k are symmetric, since x_k are symmetric to the y-axis and $P_n(x)$, $P_n'(x)$ have either even or odd symmetry.
- Error with n abscissas: $\epsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\xi)$

5.6.1 Curse of dimensionality

If we increase the dimensions of our integral, the order of accuracy decreases. Example with Simpson's rule where we have n quadrature points and d dimensions (leading to $M = n^d$ function evaluations):

One dimension ($d = 1, M = n$): $I - I_s = \mathcal{O}(\Delta x^4)$
d dimensions: $I - I_s = \mathcal{O}(M^{-4/d})$

5.6.2 2 Point Gauss-rule

Exact result for 3_{rd} order polynomial.
The method of indeterminate coefficients can be used to compute the integral of a 2n-1 polynomial: $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$ with $\int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2)$.
Comparing the coefficients of a_i leads to the 2 point Gauss-rule:
$\int_a^b f(x)dx \approx$
$\frac{b-a}{2} f\left(\left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}\right) + \frac{b-a}{2} f\left(\left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}\right)$

5.7 Monte Carlo Quadrature

The integral I is defined as follows, with M points x_i (samples) chosen randomly in Ω.

$$I = |\Omega|\langle f \rangle$$
$$\text{with } \langle f \rangle = \frac{1}{|\Omega|}\int_\Omega f(\vec{x})d\vec{x} \text{ and } |\Omega| = \int_\Omega 1 d\vec{x}$$
$$\langle f \rangle \approx \langle f \rangle_M = \frac{1}{M}\sum_{i=1}^{M} f(x_i)$$

Error estimation: The error does not depend on dimension d

$$\epsilon_M = \sqrt{\mathrm{Var}[\langle f \rangle_M - \langle f \rangle]} = \sqrt{\frac{\mathrm{Var}[f]}{M}} = \mathcal{O}(M^{-1/2})$$

$$\mathrm{Var}[X] = \langle X^2 \rangle - \langle X \rangle^2 = \langle (X - \langle X \rangle)^2 \rangle$$

$$|\langle f \rangle - \langle f \rangle_M| < \begin{cases} \epsilon_M, & \text{with probability of } 68\% \\ 2\epsilon_M, & \text{with probability of } 95\% \\ 3\epsilon_M, & \text{with probability of } 99\% \end{cases}$$

Remark: To reduce error by factor n, one needs n^2 more samples.

Monte Carlo recipe

- Draw random points x_i and evaluate the integrand f to get random variables $f(x_i)$
- Store the number of samples M, the sum of values, and the sum of squares
- Compute the mean as the estimate of the expectation (normalized integral) $\langle f \rangle_M$
- Estimate the variance from the mean of squares and the error

$$\mathrm{Var}_M[f] = \frac{M}{M-1}\left(\langle f^2 \rangle_M - \langle f \rangle_M^2\right) \quad \rightarrow \quad \epsilon_M \approx \sqrt{\frac{\mathrm{Var}_M[f]}{M}}$$