MAD Cheatsheet

Function fitting

Linear Least squares

Data: $\{t_i, b_i\}_{i=1}^{N}$
Fitted function: $f(t) = \sum_{k=1}^{M} x_k \Phi_k(t)$
where $\Phi_k(t)$ are linear independent basisfunctions. The functions can be non linear (ex: $e^{\beta t}$) but the unknown factor x must enter linearly.

Cost function

$E^2 = \sum_{i=1}^{N} e_i^2 = \sum (b_i - f(t_i))^2 \quad e_i$: residuals
Matrix Formulation:
$E^2 = E^T E = (b - Ax)^T (b - Ax) = b^T b - 2x^T A^T b + x^T A^T A x$
Problem: Find x_k to minimize Cost $\implies$ derivate wrt. x and solve for x
$x = (A^T A)^{-1} A^T b$

Formulas for linear case

$x_1 + x_2 t_i \approx b_i$
$x_1 = \frac{\sum t_i^2 \sum b_i - \sum t_i \sum t_i b_i}{N \sum t_i^2 - (\sum t_i)^2}$
$x_2 = \frac{N \sum t_i b_i - \sum t_i \sum b_i}{N \sum t_i^2 - (\sum t_i)^2}$
$b = X_1 + X_2(t - \bar{t})$
$X_1 = \frac{\sum b_i}{N} \quad X_2 = \frac{\sum (t_i - \bar{t}) b_i}{\sum (t_i - \bar{t})^2}$

Properties

- The vector E is orthogonal wrt. to A
- x can be seen as the projection of b onto A
- The conditionnumber $\kappa(A) = \|A\| \|A^{-1}\| = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$ tells us how stable a fit is the smaller the number the better the result.
- By solving the normal equation the condition number gets squared due to A beeing twice in the inverse. A more stable solution would be to apply a QR or SVD decomposition where the inverse gets first normalized.
- $A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 & 0 \end{bmatrix}^T \quad x = R^{-1} Q^T b$
- $A = U \Sigma V^T \quad x = V \Sigma^+ U^T b$

Solving non-linear equations

	Bisection	Newton's Method	Secant method
rate	1	2	1.618/1.839
cost	cheap	expensive	middle
robust	yes	no	no

Conditioning

If the function is very horizontal at the roots position, the problem is ill conditioned. $\kappa = |f'(x^*)|^{-1}$

Bisection

Search for an interval with root. Then half the interval and continue with the new interval. The midpoint will converge towards the root.

Newton

$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$
Derivation with Taylor series:
$\underbrace{f(x^*)}_{0} = f(x_k) + f'(x_k)(x^* - x_k)$

Secant method

Approximate derivative numerically:
$x_{k+1} = x_k - f(x_k) * \frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})}$ Newtons method gets linear if the multiplicity of the root is m¿1. Also the if the derivative tends to zero the method gets unstable.

Convergence rate

$\lim_{k \to \infty} \frac{|E_{k+1}|}{|E_k|^r} = C$ Where r is the order of convercence and C is the convergence error constant.

Set of Equations

$x_{k+1} = x_k - J^{-1}(x_k) F(x_k)$ where J is the Jacobian We don't invert the Jacobian instead we solve for:
$J(x_k) y = -F(x_k) \quad x_{k+1} = x_k + y$ Cond. num.: $\kappa = \|J^{-1}\|$. Costs: build J: n^2 solve: n^3 Improvements: If J isn't changing rapidly keep it constant or adjust it with the step $J_{n+1} = J_n + \frac{(\Delta F - J \Delta x)(\Delta x)^T}{(\Delta x)^T (\Delta x)}$

Optimization

By finding roots of the derivative of the Cost/Error function we can optimize multivariable equations. Where F is replaced by $\nabla E(x)$ and J by $\nabla^2 E(x)$. To further increase performance we can further choose a combination between Gradient descent and Newtons (Levenberg-Marquardt Method) $\nabla^2 E + \lambda \mathbb{I} y = -\nabla E$ where we start with a large λ and decrease it at each iteration if the error decreases and increase it if the error increases.

LSQ

We can also use this method to fit non-linear functions to a set of datapoints. $\left(J^T J - L\right) y = J^T (b - F)$
Gradient descent: $y = J^T (b - F) \quad x_{k+1} = x_k + \eta y$

Interpolation

We want to have a function f that goes through our datapoints. $\sum \alpha_k \phi_k = y_i$

Lagrange interpolation

Approximate function with polynomials: $f(x) = \sum y_k \mathcal{L}_k$
Where $L_k = \frac{(x - x_1)(x - x_2) \ldots (x - x_{k-1})(x - x_{k+1}) \ldots (x - x_n)}{(x_k - x_1)(x_k - x_2) \ldots (x_k - x_{k-1})(x_k - x_{k+1}) \ldots (x_k - x_n)}$
The error is
$|y(x) - f(x)| = |\frac{y^{(n)}(\eta)}{n!} \Pi (x - x_k)| \quad y^{(n)}$: n-th derivative

Properties

- It is not recommended to use Lagrange interpolation for extrapolation.
- Lagrange interpolation will cause huge oscilations in the edge regions (Runge's phenomenon).
- If points are changed the whole polynomial has to be recomputed
- It is sensitive to noise

Cubic splines

We interpolate between two points with a third order polynomial and require C^1 and C^0 continuity.

Derivation

$f''(x) = f_i'' \frac{x_{i+1} - x}{\Delta_i} + f_{i+1}'' \frac{x - x_i}{\Delta_i}$
$f(x) = f_i'' \frac{(x_{i+1} - x)^3}{\Delta_i} + f_{i+1}'' \frac{(x - x_i)^3}{\Delta_i} + C_i (x - x_i) + D_i$ Solve for f_i'' ,f_{i+1}'' ,C_i and D_i by imposing boundry conditions:
$\frac{\Delta_{i-1}}{6} f_{i-1}'' + \left(\frac{\Delta_{i-1} + \Delta_i}{3}\right) f_i'' + \frac{\Delta_i}{6} f_{i+1}'' = \frac{y_{i+1} - y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}}$
For the boundries a lot of different conditions can be chosen:

- Natural Spline: $f_1'' = f_N'' = 0$
- Parabolic runout: Set $f_1'' = f_2''$ and $f_N'' = f_{N-1}''$
- Clamping: $f_1' = f_N' = 0$

Tridiagonal matrix

This equation can be written in matrix notation:

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_1 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix}$$

By eliminating the lower diagonal we get the formula:

$$x_i = \begin{cases} g_N & i = N \\ g_i - w_i x_{i+1} & i = N-1, N-2, \dots, 1 \end{cases}$$

Where: $w_i = \begin{cases} \frac{c_1}{b_1} & i = 1 \\ \frac{c_i}{b_i - a_i w_{i-1}} & i = 2, 3, \dots, N-1 \end{cases}$ and

$$g_i = \begin{cases} \frac{d_1}{b_1} & i = 1 \\ \frac{d_i - a_i g_{i-1}}{b_i - a_i w_{i-1}} & i = 2, 3, \dots, N \end{cases}$$

Orthogonal basis

If we choose for ϕ_i orthogonal functions, we can easily increase the degree of our polynomial, since the new parameter α_i will not affect the old ones since the new function is orthogonal to the others. We get orthogonal functions by using Gramm-Schmidt. The closed expression for the coefficients is

$\alpha_i = \langle y(x)\phi_i(x)\rangle_p = \int\limits_{-\infty}^{\infty} y(x)\phi_i(x)p(x)dx$ Where if $y(x)$ is discrete $p(x) = \frac{1}{N}\sum\limits_{n=1}^{N}\delta(x - x_n)$ and otherwise 1.

($\alpha_i = \frac{1}{N}\sum\limits_{n=1}^{N} y_n\phi_i(x_n)$ for discrete sets)

B-Splines

B-splines are a more general form of cubicsplines where the degree of continuity can be chosen freely.

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

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

The given knot vector and its associated entries $t_j = (j = 1, \dots, M + d + 1)$ define the form of the spline. The entries have to be order, but can be repeated to ensure that derivatives equal to zero (clamped: d+1, $f'(t_j) = 0$: d). The resulting spline is described by:

$S_{d,t}(x) = \sum\limits_{i=1}^{M} \alpha_i B_{i,d,t}(x) \quad \text{for} t_{d+1} \le x < t_M + 1$

α_i are free parameters that have to be determined with LSQ in case of $N \ge M$ or are y_i in case of $M = N$. Also the parameters are tied to each other. If we change someting the whole spline has to be recomputed.

NURBS

Non-Uniform-Rational B-Splines are used to generalize B-splines even further to interpolate curves instead of functions.

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

properties

- in case of $w_i = 1$ the equation simplifies to $R_{i,d,t} = B_{i,d,t}$
- Changing a point will only affect part of the whole curve.
- For computergraphic they often use cubic Bézier curves that are patched together.

Multivariable Interpolation

Gridded data

$z = f(x, y) = \sum\limits_i\sum\limits_j a_{i,j}\phi_i(x)\phi_J(y)$ We can either use Lagrange polynomials:
$f(x_p, y_q) = \sum\limits_i\sum\limits_j Z(x_i, y_j)l_i(x_p)l_j(y_q)$
Or NURBS: $\overrightarrow{p}(u, v) = \sum\limits_i\sum\limits_j R_{i,d_u,t_u}(u)R_{j,d_v,t_v}(v)\overrightarrow{p}_{i,j}$

Irregular Data

We can either use Shepard's method which is based on we choose the approximation function to be:
$f(x, y) = \frac{\sum_k z_k g(x - x_k), y - y_k)}{\sum_k g(x - x_k, y - y_k)}$
Where $g(x, y)$ is a radial symmetrical function with decaying magnitude from the datapoint (Radial Basis Function)as: $g(x, y) = \frac{1}{(x^2 + y^2)^{\mu/2}}$
Or we can use Coons patches where we linearly interpolate twice between four corners.

Neural Networks

Neural networks function similar to braincells. A node recieves a signal from the previous layer. In this node the input gets summed up using weights for each input into the node as well as a individual bias. The sum is then evaluated with a activation function such as tanh, Heavyside, ReLU or sigmoid. The output is then sent to the next layer until the output layer is reached. Since we know how our Output has been generated, we can adjust the weights and biases, that the output of our network matches our desired output. This is achieved with trainingdata and the use of backprobagation where we adjust the weights to our liking.
$\Delta w_{ij}^k = -\eta\frac{\partial E}{\partial w_{ij}^k} \quad E(w) = \frac{1}{2}\sum\limits_{i=1} N(\overrightarrow{y_i} - NN(w, \overrightarrow{x_i}))^2$
Usual problems are that the learningrate η has to be wisely choosen, such that the learning rate isn't to slow but still converges to a minimum. Also if the network learns to long with the training data the result on other data will diverge, since it will overfit to the trainingdata, which can contain noise and errors. A method to avoid overfitting is crossvalidation (k-fold). We split our data into k-groups, where we use all but one groups to train and the spare one to validate. Afterwards we change the roles and continue for each group. $r_i = \frac{K}{N}\sum(f_i(x) - y)^2$ and $R = \frac{1}{K}\sum\limits_{i=1}^{K} r_i$ This way we get an estimated error predition error.

Numerical Integration

We approximate integrals with sums.

Rectangle rule

- Quadrature:$f(x_i)\Delta_i$
- closed Expr:$\Delta_x\sum I$

Midpoint rule

- Quadrature:$f\left(\frac{x_i + x_{i+1}}{2}\right)\Delta_i$
- closed Expr:$\Delta_x\sum I$
- Error:$\frac{1}{24}f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5)$

Trapezoidal rule

- Quadrature:$\frac{f(x_i) + f(x_{i+1})}{2}\Delta_i$
- closed Expr:$\frac{\Delta_i}{2}\left(f(x_0) + f(x_N) + 2\sum\limits_{i=1}^{N-1} f(x_i)\right)$
- Error:$-\frac{1}{12}f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5)$

Simpson rule

- Quadrature:$\frac{f(x_i) + 4f((x_i + x_{i+1})/2) + f(x_{i+1})}{6}\Delta_i$
- closed Expr:$\frac{\Delta_x}{3}\left(f(x_0) + f(x_N) + 4\sum\limits_{i=\text{odd}} f(x_i) + 2\sum\limits_{i=\text{even}} f(x_i)\right)$
- Error:$O(\Delta_i^5)$

Attention: Error is reduced by one order if it is evaluated over a domain.

Newton-Cotes

Approximation of interpolant with Lagrange polynomial.
$I \approx \Delta_i\sum\limits_{k=0}^{M} C_k^M f(x_k) \quad C_k^M = \frac{1}{\Delta_i}\int\limits_{x_i}^{x_{i+1}} l_k^M(x)\,\mathrm{d}x$

Properties

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

Richardson extrapolation

The absolute value G is approximated with $G \approx G(h)$ which is dependant on discretization h. We approximate $G(h) = G(0) + c_1 h + c_2 h^2 + \dots$ With $G_1(h) = 2G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \dots$ we can reduce the error by one degree.
We obtain:
$G_n(h) = \frac{1}{2^n - 1}\left(2^n G_{n-1}(h/2) - G_{n-1}(h)\right) = G + O(h^{n+1})$

Error estimation

$\epsilon(h/2) \approx G(h/2) - G(h)$

Romberg integration

Richardson applied to trapezoidal:
$I_k^n = \frac{4^k I_{k-1}^{2n} - I_{k-1}^n}{4^k - 1}$

Adaptive quadrature

We can use Rhomberg integration and error estimation to evaluate locally if we want to evaluate the integral with more precision at points with sudden changes.

Gauss quadrature

$I = \int_a^b f(x)\,\mathrm{d}x \approx \sum_i c_i f(x_i)$ We choose c_i and x_i to minimize the error:
For two points we require the equation to exactly integrate a fourth order polynomial. We get:
$c1 = \frac{b-a}{2} = c_2 \quad x_{1,2} = \left(\frac{b-a}{2}\right)\left(\pm\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}$

Hermite Interpolation

For the n-point rule we interpolate its values, aswell as its derivatives. $f(x) = \sum_{k=1}^n U_k(x) y_k + \sum_{k=1}^n V_k(x) y_k'$
$U_k(x) = \left[1 - 2L_k'(x_k)(x - x_k)\right] L_k^2(x) \quad V_k(x) = (x - x_k) L_k^2(x)$

n-point Gauss rule

We integrate move our general integration domain a,b to -1,1. $\int_{-1}^1 f(x)\,\mathrm{d}x = \sum_{k=1}^n y_k \int_{-1}^1 U_k(x)\,\mathrm{d}x + \sum_{k=1}^n y_k' \int_{-1}^1 V_k(x)\,\mathrm{d}x$ We want that $v_k = 0$ to match our previous rules. We factor $L_k(x) = C_k F(x)/(x - x_k)$ where C_k is the denominator and $F(x)/(x - x_k)$ the nominator of L_k. Since C_k is nonzero, we can require: $0 = \int\limits_{-1}^{1} F(x) L_k(x)\,\mathrm{d}x$.
Polynomials $F(x)$ with this property are the Legendre polynomials. The roots of the n-th Legendre polynomial are therefore x_k and $u_k = \frac{2}{(1 - x_k^2)(P_n'(x_k))^2}$
The error with n abscissas is $\epsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\eta)$

Multidimensional Integrals

We can express multidimensional integrals as nested integrals:
$I \approx \sum_{\substack{i_1=1\\ \dots\\ i_d=1}}^{n} w_{i_1,\dots,i_d} f(x_{i_1}, \dots, x_{i_d})$ However the error of these integrals scales with:$I - I_S = O(M^{-1/d}$. (for Simpson $O^{-4/d}$) where M are the function evaluations and d the dimension.

Monte-Carlo Integration

With Monte-Carlo Integration we sample random points from our integration domain Ω and evaluate f at these points. Our Integral becomes $I = |\Omega|\langle f\rangle$ Where $\langle f\rangle = \frac{1}{|\Omega|}\int_\Omega f(x)dx$. Now we approximate $\langle f\rangle$ with M uniform random function evaluations
$\langle f\rangle \approx \langle f\rangle_M = \frac{1}{M}\sum\limits_{i=1}^{M} f(x_i)$. For sufficient enough evaluations the estimation will tend towards the exact value. The error of this Method is $\epsilon_M = O(M^{-1/2})$ which is independant of d. Therefore Monte Carlo is more efficient for $d > 8$.

Sampling

Usually we can sample in some form from an uniform distribution. But sometimes we want to sample from non-uniform distributions.

Inverse Transform Sampling

We want to transform a uniform distribution u to a non-uniform distribution x. Where $p_x(x)$ is the probability density function and $F_x(x)$ is the commulative density function. It holds that $F_u(u) = \int\limits_0^u p_u(s)\,\mathrm{d}s = u$ As for any cdf the value will continually grow which allows us to write $F_x = u$ Which allows us to write $x = F_x^{-1}(u)$ This can be used for distributions where its density can be calculated easily. But for example for a normal distribution where the cdf can not be calculated analytically we can sometimes use approximations such as Box-Muller:
$x_1 = \sqrt{-2\ln(u_1)}\cos(2\pi u_2) \quad x_2 = \sqrt{-2\ln(u_1)}\sin(2\pi u_2)$

Rejection Sampling

We have a simple distribution with pdf $h(x)$ from which we can sample and which bounds our desired distribution $p(x)$: $p(x) < \lambda h(x)$. If this hold we can evaluate: $u < \frac{p(x)}{\lambda h(x)}$ where u is sampled from a uniform distribution. If it holds we accept x and otherwise we reject it.

Importance Sampling

For some distributions we have to reject a lot of samples which will result in a lot of wasted effort. To circumvent this we draw samples x that are distributed as probability $w(x)$. To compensate for the bias we normalize $p(x)$ with the same function $w(x)$:
$\langle f\rangle_p = \frac{1}{b-a}\sum\limits_{i=1}^{M} f(x_i)\frac{p(x_i)}{w(x_i)}$

Probability

Basics

- There is an random variable X that has an associated probability $p(x)$
- $\sum\limits_i P(x_i) = 1$
- The cumulative distribution function (CDF) $F_x(x)$ is the probability P that a value chosen from the variables distribution is less or equal than some threshold.
- The probability density function (PDF) p is the derivative of the CDF. And integrates to one over the domain.
- $P(a \le X \le b) = \int\limits_a^b p(x)\,\mathrm{d}x$

Common distributions

Uniform distribution

$p_{\mathcal{U}}(x) = \frac{1}{b-a}$

Binomial distribution

$p(k) = \binom{n}{k} p^k (1-p)^{n-k}$

Normal distribution

$p_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\exp(-\frac{(x-\mu)^2}{2\sigma^2})$

Expected value

$\mathbb{E}[X] = \langle X\rangle = \int_\Omega x p(x)\,\mathrm{d}x$
In discrete settings (mean): $\mathbb{E}(x) = \sum\limits_i x_i P(x_i)$

Variance

$\mathrm{Var}[X] = \sigma^2[X] = \mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}[X^2] - E[X]^2$

Bayesian inference

Bayes' theorem

$P(A_j|B) = \frac{P(B|A_j)P(A_j)}{\sum\limits_{i=1}^{n} P(B|A_i)P(A_i)}$

Parameter estimation

$\underbrace{P(\theta|D,I)}_{\text{posterior}} = \frac{\overbrace{P(D|\theta,I)}^{\text{likelihood}}\overbrace{P(\theta|I)}^{\text{prior}}}{\underbrace{P(D|I)}_{\text{evidence}}} \quad P(I|D) = \int P(D|\theta,I)P(\theta|I)\,\mathrm{d}\theta$

- I: Information
- D: Data
- M: Model explaining data (I)

We look for parameters θ_0 that minimize the error of the model.

Laplace Approximation

Looking for maximum:

$\frac{\partial P}{\partial \theta}|_{\theta_0} = 0 \quad \frac{\partial^2 P}{\partial \theta^2}|_{\theta_0} < 0$

We can define L as $L(\theta) = \log(P(\theta))$ which conserves the location of the maximum. Then we approximate $L(\theta)$ with a Taylor expansion and substitute $P(\theta)$ back:

$$P(\theta|D,I) \approx A \exp\left(\frac{1}{2}\frac{\partial^2 L}{\partial \theta^2}|_{\theta_0}(\theta-\theta_0)^2\right)$$

We replace the second derivative with $-\sigma^{-2}$ which gets us a normal distribution around θ_0.

Coin example

$P = (H)^R(1-H)^{N-R}$

- H: Probability for Head
- N: Coin tosses
- R: # Heads in N

$$L(H) = C + R\log(H) + (N-R)\log(1-H)$$

$$\frac{\partial L}{\partial H} = \partial RN - \partial N - R1 - H \quad \frac{\partial^2 L}{\partial H^2} = -\frac{R}{H^2} - \frac{N-R}{(1-H)^2}$$

This implies $H_0 = \frac{R}{N}$ and $\sigma = \sqrt{\frac{H_0(1-H_0)}{N}}$

Model selection

We can use Bayes to compare models:

$$\frac{P(M_1|D}{P(M_2|D)} = \frac{P(D|M_1)P(M_1|I)}{P(D|M_2)P(M_2|I)}$$

We assume$P(M_i|I)$ are uniform distributions:

$$\frac{P(M_1|D}{P(M_2|D)} = \frac{P(D|M_1)(m_2^{\max} - m_2^{\min})}{P(D|M_2)(m_1^{\max} - m_1^{\min})}$$