CLASS NOTES

Models, Algorithms and Data: Introduction to computing

2019

Petros Koumoutsakos, Jens Honore Walther, Julija Zavadlav

(Last update: July 17, 2019)

IMPORTANT DISCLAIMERS

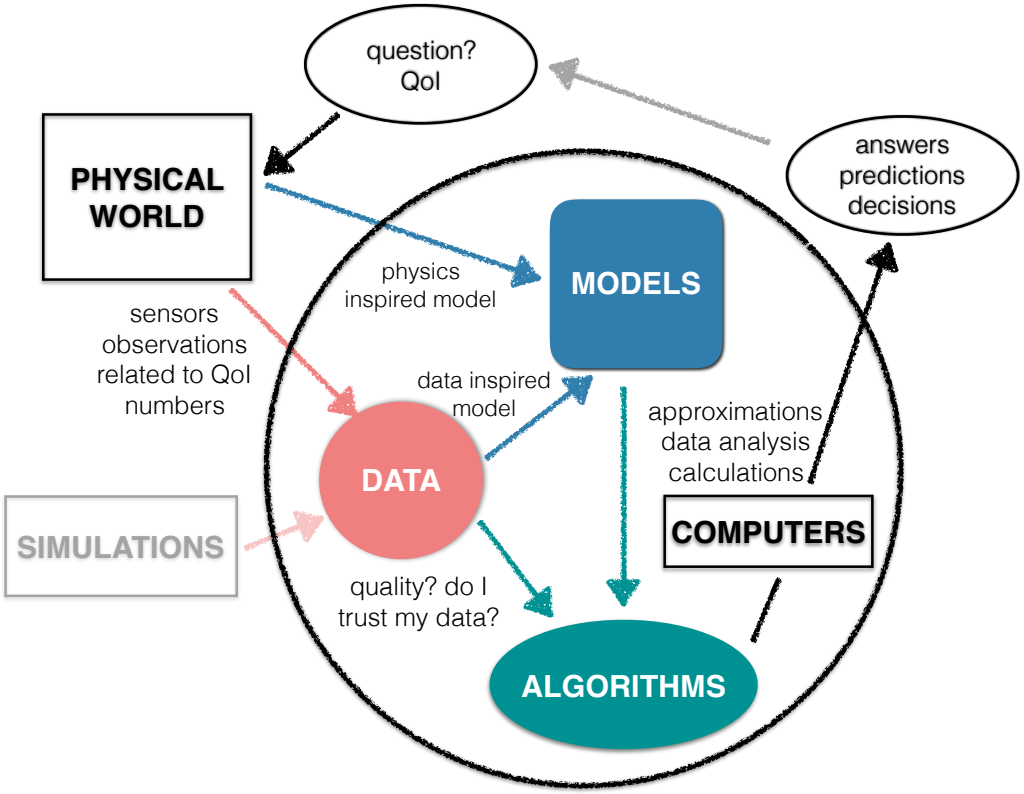
Much of the material (ideas, definitions, concepts, examples, etc) in these notes is taken for teaching purposes, from several references:

- Numerical Analysis by R. L. Burden and J. D. Faires
- The Nature of Mathematical Modeling by N. Gershenfeld
- A First Course in Numerical methods by U. M. Ascher and C. Greif

These notes are only informally distributed and intended ONLY as study aid for the final exam of ETHZ students that were registered for the course Models, Algorithms and Data (MAD): Introduction to computing 2019. The notes have been checked, however they may still contain errors so use with care.

Contents

The MAD World



Suppose we have some question about the physical world that is we are interested in some quantity of interest (QoI). By performing an experiment (observing the physical world), the sensors give us some **DATA** (numbers) related to our QoI.

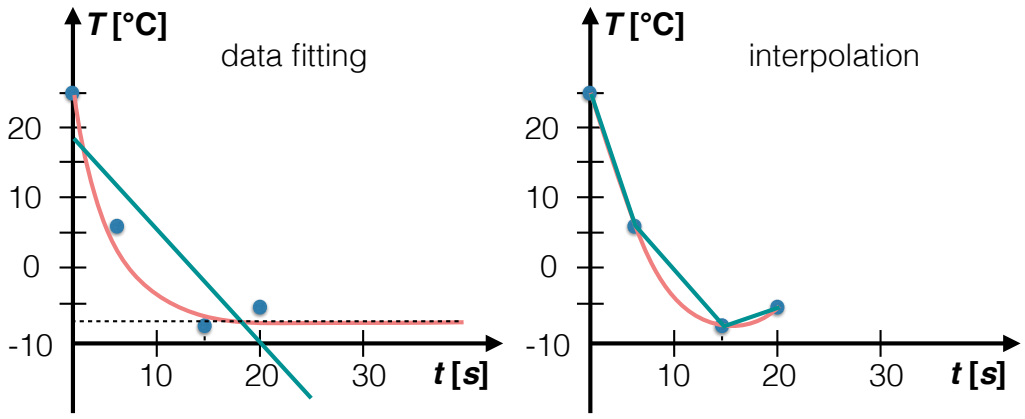


Figure 1: Example on shaking (see http://www.cookingissues.com/index.html%3Fp=1527.html): we would like to know how long one should shake a cocktail. We insert a thermostat into the shaker and measure the temperature at some time instances during the shaking of the cocktail.

To answer questions about our QoI we construct a **MODEL** that describes the data. Models are "architectures of assumptions". These assumptions can be inspired by

- **data**, e.g. we see patterns in the data and therefore assume some underlying functional form.
- **physical laws**, e.g., we may know that the relationship is following some physical law.

Depending on the assumptions about the data, we can construct a model that will either:

- **fit** the data: we construct a low order model that approximates the data. Our model ("functions") does not pass through all the data points. We perform data fitting, when *we assume that the observed data has errors.*
- **interpolate/extrapolate** the data: we construct a model such that the function goes through all the data points. We perform data interpolation, when *we assume that the data points are accurate.* Typically, when data points are hard to evaluate and we find a function that is easier to evaluate and can interpolate between (extrapolate beyond) the data points. Here, the model is specified by the data. (Lectures 4-6)

For data fitting, we further distinguish between

- **parametric** fitting where we are able to specify a particular function form of the model which has a computable number of adjustable parameters. For example, the model is $f(x) = ax + b$ and a, b are the parameters of the model. (Lecture 1)
- **nonparametric** fitting, where the model has so many parameters that the model can describe any functional form. This is the area of Machine learning, e.g., Neural Networks. (Lecture 7)

In both cases, the goal is to obtain the values of the parameters such that the model will "best" capture the data. This implies **optimization** in terms of a minimizing or maximizing a cost function. Such tasks are be performed by **ALGORITHMS**. In this course, we will also discuss: how to **solve nonlinear equations** (Lectures 2 and 3), how to perform numerical **integration** (Lectures 8-11). Note that algorithms depend on the assumed model architecture, i.e., some algorithms are only applicable for some models.

Lastly, in function fitting we assume a model. But how can we know whether the assumed model was good or bad? Or given models A and B can we establish which one is better? We will see that we can answer this question with Uncertainty Quantification (Lectures 12 and 13).

LECTURE 1

Modeling Data: Linear Least Squares

1.1 Parametric Function Fitting

Given a set of data $\{(t_i, b_i)\}_{i=1,\ldots,N}$ the aim of function fitting is to find the parameters of a function $f(t)$ such that $f(t)$ will "best" describe the data, i. e., $b_i \approx f(t_i)$. Assuming there is a function $g(t)$ describing exactly all the data then $f(t)$ is an approximation of $g(t)$.

In function fitting, we need to specify:

- **model architecture:** the functional form of function $f(t)$. We can choose to fit our data with a straight line (linear function), a polynomial function, an exponential function etc.
- **measure of "best":** the cost function. The parameters of f can be found by minimizing the $\sum_{i=1}^{N} \left[b_i - f(t_i)\right]^2$ (squared L2-norm) or we can choose to minimize $\sum_{i=1}^{N} \left|b_i - f(t_i)\right|$ (L1-norm), or $\max_i \left|b_i - f(t_i)\right|$ (L∞-norm). In the following we will denote $\|\boldsymbol{x} - \boldsymbol{y}\|_k = \sqrt[k]{\sum_{i=1}^{n} (x_i - y_i)^k}$ for $x, y \in \mathbb{R}^n$, which is the Lk-norm.

Different choices will lead to different results as shown in Figure 1.1 where the same data points have

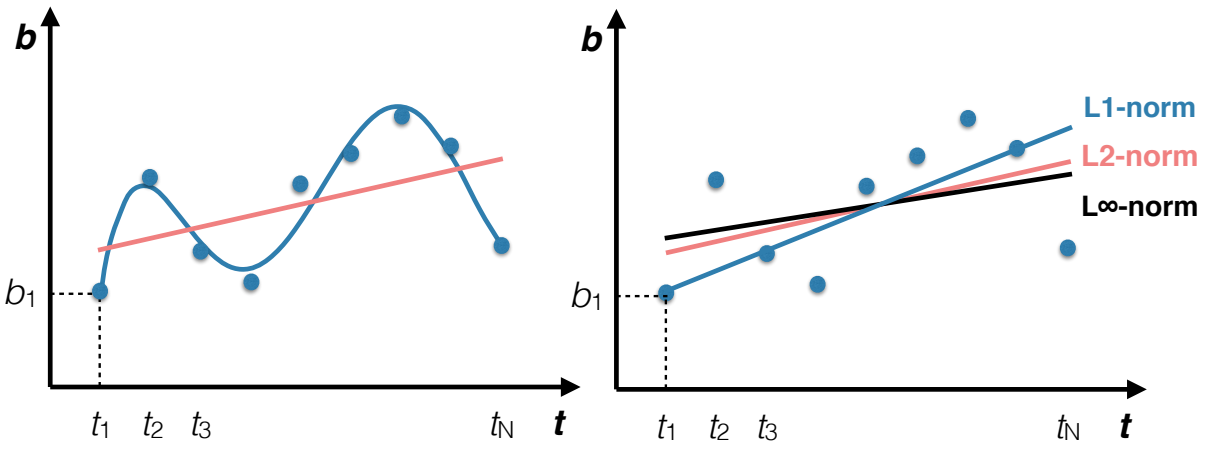


Figure 1.1: Example showing function fitting of given data points with two possible model architectures (left): straight and curved lines. In the right plot, we show three different linear approximations to the data obtained with different norm measures.

been fitted with 2 model architectures (Figure 1.1 left): a straight line (a first order polynomial) and a higher order polynomial and using 3 different measures of "best" (Figure 1.1 right).

Typically, we have more data points N than the total number of free parameters M of function f, which is why the fitted function in general does not pass through all the data points. If $N = M$, then we talk about data interpolation/extrapolation. A topic that will be discussed in subsequent lectures.

1.2 Linear Least Squares

We choose to fit the data $\{(t_i, b_i)\}$, $i = 1, \ldots, N$ with a fitting function $f(t)$ that can be expressed as a linear combination of M linearly independent functions $\phi_k(t)$, i.e.,

$$f(t) = \sum_{k=1}^{M} x_k \phi_k(t) \tag{1.2.1}$$

The functions $\phi_k(x)$ are called basis functions where typically $M \ll N$. Some examples of functions that can be used as basis functions ϕ_k are given below:

$$\phi_k(t) = t^{k-1}, \qquad \phi_k(t) = \cos[(k-1)t],$$
$$\phi_k(t) = e^{\beta_k t}, \qquad \phi_k(t) = 1 - \frac{|t - t_k|}{\Delta}.$$

Note the following: The functions ϕ can be nonlinear, i.e., linear least squares doesn't mean we are fitting a linear function to the data. *"Linear" in the Linear Least Squares method refers to the fact that the unknown parameters x_k enter linearly.* The functions ϕ should not be linearly dependent otherwise some parameters are redundant.

We want to find the unknown parameters x_k with $k = 1, ..., M$ such that the cost function

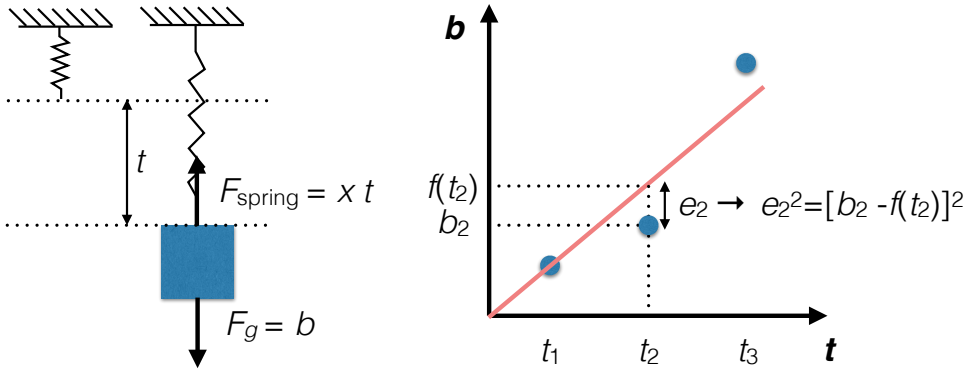
$$E^2 = \sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} \left[b_i - f(t_i) \right]^2 \tag{1.2.2}$$

is **minimised**. *"Least Squares" refers to the fact that the "best" fit is the one that minimizes the sum of squared L2-norm of the residuals e_i.* Residual is a distance between the observed value b_i and the fitted value $f(t_i)$ provided by the model. Note that minimizing E^2 is equivalent to minimizing E, i.e., both cost functions give the same result. We choose E^2 so we do not have to deal with square root.

Example 1.1: A linear spring

We measure the elastic constant of a linear spring by imposing different weights b on the spring and measuring the displacement t. From physics we know that the force of gravity is balanced with the spring force, i.e., $b = xt$. Therefore, it is reasonable to fit the data with $f(t) = x\phi(t)$, where $\phi(t) = t$. We measure data points $\{t_1, b_1\}$, $\{t_2, b_2\}$, $\{t_3, b_3\}$ and the only unknown is the

elastic constant of the spring x. In this case $M = 1$ and $N = 3$.



We have the following equations:

$$t_1 x = b_1$$
$$t_2 x = b_2$$
$$t_3 x = b_3$$

In order to solve this problem, we seek to find a solution (Eq. (1.2.2)) by minimising

$$E^2 = (b_1 - t_1 x)^2 + (b_2 - t_2 x)^2 + (b_3 - t_3 x)^2$$

$$\frac{dE^2}{dx} = 0 \Longrightarrow -2\left[(b_1 - t_1 x)\,t_1 + (b_2 - t_2 x)\,t_2 + (b_3 - t_3 x)\,t_3\right] = 0$$

$$\bar{x} = \frac{t_1 b_1 + t_2 b_2 + t_3 b_3}{t_1^2 + t_2^2 + t_3^2},$$

where we have denoted with $\bar{x}$ the least squares solution to the problem.

1.3 Matrix formulation and the Normal equations

The linear least squares problem is more easily solved if we write the system of N equations with M unknowns in the matrix formulation. We are trying to find the unknown parameter vector $\boldsymbol{x}$ such that

$$A\boldsymbol{x} = \boldsymbol{b} \tag{1.3.1}$$

where $A \in \mathbb{R}^{N\times M}$, $\boldsymbol{x} \in \mathbb{R}^M$ and $\boldsymbol{b} \in \mathbb{R}^N$. In components this reads

$$\overbrace{\begin{bmatrix} \phi_1(t_1) & \phi_2(t_1) & \dots & \phi_M(t_1) \\ \phi_1(t_2) & \phi_2(t_2) & \dots & \phi_M(t_2) \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \phi_1(t_N) & \phi_2(t_N) & \dots & \phi_M(t_N) \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}}^{\boldsymbol{x}} = \overbrace{\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ \vdots \\ b_N \end{bmatrix}}^{\boldsymbol{b}} \tag{1.3.2}$$

To determine $\boldsymbol{x}$ we need to solve the above system of equations. We distinguish between different cases:

M=N Matrix A is square. Since functions $\Phi_k(t)$ are linearly independent there exists a unique solution given by $\boldsymbol{x} = A^{-1}\boldsymbol{b}$. Remember from linear algebra that the following statements are equivalent: columns/rows of A are linearly independent $\Leftrightarrow A$ is not singular $\Leftrightarrow$ there exists A^{-1} such that $A^{-1}A = AA^{-1} = I \Leftrightarrow \det A \neq 0 \Leftrightarrow \text{rank}(A) = N \Leftrightarrow \text{range}(A) \equiv \text{im}(A) = \mathbb{R}^N \Leftrightarrow \text{null}(A) \equiv \ker(A) = 0$. Numerical methods to solve such systems are: Gauss elimination, pivoting strategies, LU decomposition, Cholesky decomposition, etc.

M>N We have an underdetermined system with infinitely many solutions.

M<N We have an overdetermined system and we can seek an approximate solution $A\boldsymbol{x} \approx \boldsymbol{b}$ with least squares method by requiring that $E^2 = \|\boldsymbol{b} - A\boldsymbol{x}\|_2^2$ is minimal.

For the last case, we therefore have

$$\begin{aligned} E^2 = E^T E =& (\boldsymbol{b} - A\boldsymbol{x})^T(\boldsymbol{b} - A\boldsymbol{x}) \\ =& (\boldsymbol{b}^T - \boldsymbol{x}^T A^T)(\boldsymbol{b} - A\boldsymbol{x}) \\ =& \boldsymbol{b}^T\boldsymbol{b} - \boldsymbol{b}^T A\boldsymbol{x} - \boldsymbol{x}^T A^T\boldsymbol{b} + \boldsymbol{x}^T A^T A\boldsymbol{x} \\ =& \boldsymbol{b}^T\boldsymbol{b} - 2\boldsymbol{x}^T A^T\boldsymbol{b} + \boldsymbol{x}^T A^T A\boldsymbol{x} \end{aligned}$$

In the last line we have used the fact that $\boldsymbol{b}^T A\boldsymbol{x}$ is a $(1\times N)(N\times M)(M\times 1)$ so (1×1) matrix which is always symmetric. In order to obtain a minimum, we compute the derivative with respect to $\boldsymbol{x}$[1]:

$$\frac{dE^2}{d\boldsymbol{x}} = 0 \Longrightarrow -2\boldsymbol{b}^T A + 2\boldsymbol{x}^T A^T A = 0 \quad transpose \quad -2A^T\boldsymbol{b} + 2A^T A\boldsymbol{x} = 0$$

[1] In order to review matrix differentiation see https://atmos.washington.edu/%7Edennis/MatrixCalculus.pdf

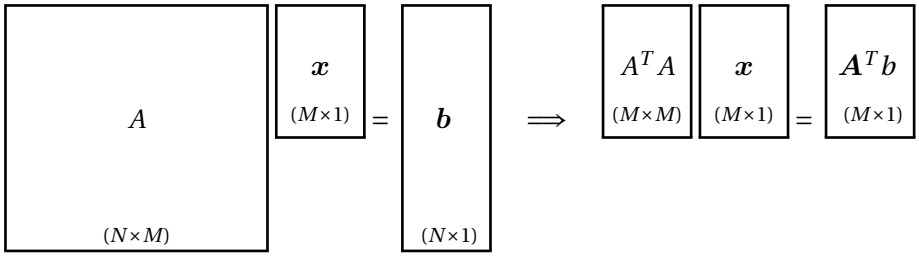
We obtain the so-called normal equations

$$A^T A\boldsymbol{x} = A^T \boldsymbol{b}. \tag{1.3.3}$$

Since we assume that our functions ϕ_k are linearly independent $A^T A$ is a symmetric, positiv definite matrix[2]. Thus it can be inverted and we can write the least squares solution $\bar{x}$ as

$$\boxed{\bar{x} = (A^T A)^{-1} A^T \boldsymbol{b}} \tag{1.3.4}$$

Note that the normal equations transform the problem:



Is the obtained solution truly a minimum? To see this we write the Taylor series expansion around $\bar{x}$:

$$\begin{aligned} E^2(\bar{x}+\boldsymbol{y}) =& E^2(\bar{x}) + \sum_{j=1}^{M} y_j \frac{\partial E^2(\bar{x})}{\partial x_j} + \frac{1}{2}\sum_{j=1}^{M}\sum_{k=1}^{M} y_j y_k \frac{\partial^2 E^2(\bar{x})}{\partial x_j \partial x_k} \\ =& E^2(\bar{x}) + y^T A^T A y > E^2(\bar{x}) \end{aligned}$$

We used the fact that $\frac{dE^2(\bar{x})}{d\boldsymbol{x}} = 0$ and $\frac{d^2E^2(\bar{x})}{d^2\boldsymbol{x}} = 2A^T A$. As already mentioned the matrix $A^T A$ is symmetric positive definite and therefore we conclude that the least square solution $\bar{x}$ is a minimum since adding a small vector $\boldsymbol{y}$ will increase E.

Example 1.2: A classic: Fitting a line

Consider a set of N experimental results $\{t_i, b_i\}$, which we wish to fit to

$$x_1 + x_2 t_i \approx b_i$$

According to the notation in Eq. (4.1.1), this means that we use $\phi_k(t_i) = t_i^{k-1}$ with $k = 1,2$ and $M = 2$. We can rewrite the problem in matrix form as

$$\begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \approx \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix}$$

[2] Symmetry is straight forward $(AA^T)^T = (A^T)^T A^T = AA^T$. For the other part first recall that a matrix B is positive definite if for any nonzero vector $\boldsymbol{y}$: $\boldsymbol{y}^T B\boldsymbol{y} > 0$. To see that this is true for matrix $A^T A$ we write: $\boldsymbol{y}^T A^T A\boldsymbol{y} = (A\boldsymbol{y})^T (A\boldsymbol{y}) = \|A\boldsymbol{y}\|_2^2 > 0$.

According to Eq. (1.3.3), we then need to solve the normal equations

$$\begin{bmatrix} N & \sum_{i=1}^N t_i \\ \sum_{i=1}^N t_i & \sum_{i=1}^N t_i^2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N b_i \\ \sum_{i=1}^N t_i b_i \end{bmatrix} \tag{1.3.5}$$

This can be solved as it is a square (2x2) matrix.The solution is given as:

$$\begin{aligned} \overline{x}_1 &= \frac{\left(\sum_{i=1}^N t_i^2\right)\left(\sum_{i=1}^N b_i\right) - \left(\sum_{i=1}^N t_i\right)\left(\sum_{i=1}^N t_i b_i\right)}{N\left(\sum_{i=1}^N t_i^2\right) - \left(\sum_{i=1}^N t_i\right)^2} \\ \overline{x}_2 &= \frac{N\left(\sum_{i=1}^N t_i b_i\right) - \left(\sum_{i=1}^N t_i\right)\left(\sum_{i=1}^N b_i\right)}{N\left(\sum_{i=1}^N t_i^2\right) - \left(\sum_{i=1}^N t_i\right)^2}. \end{aligned} \tag{1.3.6}$$

1.4 Geometric interpretation

Example 1.3: A linear spring: continued

Let's look again at the linear spring example. In matrix notation, the problem is given by

$$\begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} x = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

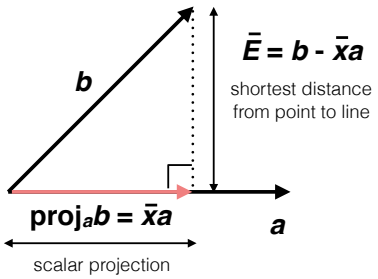
and the solution was

$$\bar{x} = \frac{t_1 b_1 + t_2 b_2 + t_3 b_3}{t_1^2 + t_2^2 + t_3^2} = \frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|_2^2},$$

which we rewrote using $\boldsymbol{a} = (t_1, t_2, t_3)$ and $\boldsymbol{b} = (b_1, b_2, b_3)$. From geometry we know that a projection of vector $\boldsymbol{b}$ onto vector $\boldsymbol{a}$ is given by

$$\text{proj}_{\boldsymbol{a}} \boldsymbol{b} = \frac{\boldsymbol{a}\boldsymbol{b}}{\|\boldsymbol{a}\|} \frac{\boldsymbol{a}}{\|\boldsymbol{a}\|},$$

where $\frac{\boldsymbol{a}\boldsymbol{b}}{\|\boldsymbol{a}\|}$ is called a scalar projection which is equal to the length of the vector projection.



This means that linear least squares gives us a solution $\bar{x}$ such that $\boldsymbol{p} = \bar{x}\boldsymbol{a}$ is a projection of $\boldsymbol{b}$ onto $\boldsymbol{a}$, which is the point on the line through $\boldsymbol{a}$ closest to $\boldsymbol{b}$. Therefore, the residual $\bar{E} = \boldsymbol{b} - \bar{x}\boldsymbol{a}$ is perpendicular to vector $\boldsymbol{a}$.

The geometrical properties of least squares solutions that we have observed in example 1.3 are true in general. *Since the basis functions $\phi(t)$ are chosen such that they are linearly independent, the column vectors of matrix A span a M-dimensional space. The least squares method finds the solution $\bar{x}$ such that $A\bar{x}$ is the projection of vector $\boldsymbol{b}$ on the column space of A. The residual or error $\bar{E} = \boldsymbol{b} - A\bar{x}$ is for least squares solution perpendicular to that space and has minimal length which means that $E = \|\boldsymbol{b} - A\boldsymbol{x}\|$ is minimal for $\bar{x}$*. Note that the equation $A\boldsymbol{x} = \boldsymbol{b}$ has a solution only if $\boldsymbol{b}$ is in the column space of A in which case $E = 0$.

To see that $\bar{E}$ is perpendicular to the column space of A we write

$$\begin{aligned} A^T A\bar{x} &= A^T \boldsymbol{b} \\ 0 &= A^T(\boldsymbol{b} - A\bar{x}) \\ 0 &= A^T \bar{E} \end{aligned}$$

Suppose that the fitting function has 2 basis functions $M = 2$ and we have $N = 3$ data points. Then the geometric interpretation can be nicely visualised (Figure 1.2) in 3D space, where the 2 columns of matrix A span a plane. For an inconsistent system of equations, vector $\boldsymbol{b}$ lies outside of that plane

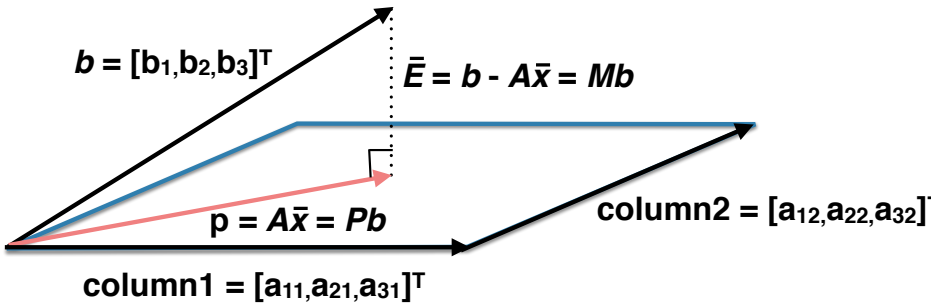


Figure 1.2: Geometric interpretation of the least squares solution with $N = 3$ and $M = 2$.

and the least squares method finds the vector $\boldsymbol{p} = A\bar{x}$ in the column space of $A\boldsymbol{x}$ that is closest to the given $\boldsymbol{b}$. Searching for the least squares solution $\bar{x}$, which will minimise the error E, is the same as locating the point $\boldsymbol{p} = A\bar{x}$ that is closer to $\boldsymbol{b}$ than any other point in the column space.

1.4.1 The Projection Matrix P

We have shown that the closest point to $\boldsymbol{b}$ is

$$\boldsymbol{p} = A\bar{x} = A(A^T A)^{-1} A^T \boldsymbol{b} = P\boldsymbol{b},$$

where matrix P given by

$$\boxed{P = A(A^T A)^{-1} A^T} \tag{1.4.1}$$

is called the **projection matrix**. Any projection matrix has two basic properties:

1. It is symmetric: $P = P^T$
2. It is idemptotent: $P^2 = P$

On the other hand, we can write the residual as

$$\bar{E} = \boldsymbol{b} - A\bar{x} = \boldsymbol{b} - P\boldsymbol{b} = \boldsymbol{b} - A(A^T A)^{-1} A^T \boldsymbol{b} = (I - A(A^T A)^{-1} A^T)\boldsymbol{b} = Mb,$$

where

$$M = I - A(A^T A)^{-1} A^T$$

which is also a projection matrix ($M = M^T$ and $M^2 = M$). It can be easily seen that $P + M = I$, $PM = 0$, $PA = A$ and $MA = 0$. The matrix P projects a vector onto the column space of A, while the matrix M is the projection onto the space orthogonal to the column space of A.

1.5 Special case: The Orthonormal Case

Consider A as an $M \times N$ matrix and suppose that its columns $A_i = A\hat{e}_i$ are **orthonormal**.

$$A_i^T A_j = \begin{cases} 0, & i \neq j \\ 1, & i = j \end{cases}$$

A square matrix with orthonormal columns is an **orthogonal matrix**.

Properties of orthogonal matrices:

$$\begin{aligned} & A^T A = I = AA^T, \quad A^T = A^{-1}, \\ \Leftrightarrow \quad & \|A\boldsymbol{x}\| = \|\boldsymbol{x}\|, \quad (A\boldsymbol{x})^T (A\boldsymbol{y}) = \boldsymbol{x}^T \boldsymbol{y}, \text{ for all } \boldsymbol{x}, \boldsymbol{y} \end{aligned} \tag{1.5.1}$$

Again $\bar{x} = (A^T A)^{-1} A^T \boldsymbol{b}$ is the least squares solution and $P = A(A^T A)^{-1} A^T$ is the projection matrix. In the case where columns of A are orthonormal the matrix $A^T A$ is the identity matrix, $P = AA^T$ and

$$\boxed{\bar{x} = A^T b.} \tag{1.5.2}$$

Example 1.4: A matrix with orthogonal columns

Fitting a straight line leads to orthogonal (but not orthonormal) columns when the measurements are taken at times which average to zero. We can always do this transformation: $\bar{t} =$

$(t_1 + \ldots + t_m)/m$. Instead of working with $b = x_1 + x_2 t$, we work with $b = X_1 + X_2(t - \bar{t})$.

$$\begin{bmatrix} 1 & t_1 - \bar{t} \\ \vdots & \vdots \\ 1 & t_N - \bar{t} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix}$$

If we insert this in Eq. (1.3.5), we observe that we obtain a diagonal matrix on the left hand side

$$\begin{bmatrix} N & 0 \\ 0 & \sum_{i=1}^N (t_i - \bar{t})^2 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N b_i \\ \sum_{i=1}^N (t_i - \bar{t}) b_i \end{bmatrix}.$$

We can easily solve the system to get

$$X_1 = \frac{\boldsymbol{a}_1^T \boldsymbol{b}}{\boldsymbol{a}_1^T \boldsymbol{a}_1} = \frac{\sum b_i}{N}, \quad X_2 = \frac{\boldsymbol{a}_2^T \boldsymbol{b}}{\boldsymbol{a}_2^T \boldsymbol{a}_2} = \frac{\sum (t_i - \bar{t}) b_i}{\sum (t_i - \bar{t})^2},$$

where $\boldsymbol{a}_1, \boldsymbol{a}_2$ are the columns of the matrix $A = [\boldsymbol{a}_1, \boldsymbol{a}_2]$. This is an example of the Gram-Schmidt process for orthogonalisation.

1.6 Numerical solutions

In this section, we discus how least squares problem is solved numerically. Generally, we do not solve the normal equations $A^T A\bar{x} = A^T b$ because they are ill conditioned as we will see below.

1.6.1 Condition number

In computer, the numbers are stored with limited digits. Thus, our numerical solution will be impacted by the round off errors. Suppose we have obtained a numerical solution $\bar{x}$ of $A\boldsymbol{x} = \boldsymbol{b}$, where $\boldsymbol{x}$ is a true solution. For now, we assume A is square and invertible. The residual of the numerical solution is non-zero due to roundoff errors:

$$E = \boldsymbol{b} - A\bar{x} \neq 0.$$

If $\|E\|$ is small, does this mean that $\|\boldsymbol{x} - \hat{x}\|$ is also small? The answer is NO. Now, we seek a relation between these two quantities:

$$\begin{aligned} E = \boldsymbol{b} - A\bar{x} &= A\boldsymbol{x} - A\bar{x} = A(\boldsymbol{x} - \bar{x}) \\ \boldsymbol{x} - \bar{x} &= A^{-1}E \\ \|\boldsymbol{x} - \bar{x}\| &= \|A^{-1}E\| \leq \|A^{-1}\|\|E\| \\ \frac{\|\boldsymbol{x} - \bar{x}\|}{\|\boldsymbol{x}\|} &\leq \|A^{-1}\|\|\hat{E}\| \frac{\|A\|}{\|\boldsymbol{b}\|} = \kappa(A) \frac{\|\hat{E}\|}{\|\boldsymbol{b}\|} \end{aligned}$$

where we used in the last line the relation $\|\boldsymbol{b}\| \leq \|A\|\|\boldsymbol{x}\|$ and defined the **condition number** of A as

$$\boxed{\kappa(A) = \|A\|\|A^{-1}\|} \tag{1.6.1}$$

The condition number of a matrix takes values $1 \leq \kappa(A) < \infty$. We call a problem well-conditioned, if the value $\kappa(A)$ is not too large. In such cases the obtained solution will be close to the true solution. Furthermore the condition number can be interpret as the a measure of how close a matrix is to be singular.

Let us regard some special cases:

orthogonal matrix Q In this case $\|Q\|_2 = 1$, $\|Q\|_2^{-1} = 1$ and $\kappa_2(A) = 1$, where we used subscript 2 to denote the L2-norm. Orthogonal matrix are ideally conditioned.

symmetric positive definite matrix A Here we know that A has real and positive eigenvalues $0 < \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_M$. The L2-norm for regular matrices is the biggest eigenvalue $\|A\|_2 = \lambda_M$. The A^{-1} is also symmetric positive definite matrix with eigenvalues $0 < 1/\lambda_M \leq 1/\lambda_{M-1} \leq \cdots \leq 1/\lambda_1$ and $\|A^{-1}\|_2 = 1/\lambda_1$. Therefore, $\kappa(A) = \frac{\lambda_M}{\lambda_1}$.

Example 1.5: Numerical singularity

Consider a matrix

$$A = \begin{bmatrix} 1 & 1 \\ \epsilon & 0 \\ 0 & \epsilon \end{bmatrix} \quad \rightarrow \quad A^T A = \begin{bmatrix} 1+\epsilon^2 & 1 \\ 1 & 1+\epsilon^2 \end{bmatrix} \underset{\text{double precision}}{\overset{\epsilon<\sqrt{\eta}}{\longrightarrow}} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

Forming $A^T A$ results in a loss of information if $\epsilon < \sqrt{\eta}$, where η is machine precision, because $A^T A$ becomes numerically singular.

If we now return to the problem of least squares, we note that the normal equations $A^T A\boldsymbol{x} = A^T\boldsymbol{b}$ can be rewritten as $B\boldsymbol{x} = \boldsymbol{c}$, where $B = A^T A$ is a symmetric positive definite matrix. As we have seen above that for $\lambda_1 \ll \lambda_M$ such a matrix can have a very large condition number. Generally one can say that the relative error in the solution using normal equations is bounded by

$$\kappa(A^T A) \approx [\kappa(A)]^2.$$

Now, we would like to find algorithms such that the error is bounded by $\kappa(A)$ and not its square.

1.6.2 QR decomposition

Any matrix $A \in \mathbb{R}^{N\times M}$ with linearly independent columns (full column rank) can be factored into

$$A = QR = \begin{pmatrix} Q_1 & Q_2 \end{pmatrix} \begin{pmatrix} R_1 \\ 0 \end{pmatrix} = Q_1 R_1 \tag{1.6.2}$$

where $Q \in \mathbb{R}^{N\times N}$ is an orthogonal matrix. In the later decomposition $Q_1 \in \mathbb{R}^{N\times M}$ is a matrix with orthogonal columns, and $R_1 \in \mathbb{R}^{M\times M}$ is upper triangular and invertible. Using $A = Q_1 R_1$ the least squares solution is given by

$$\bar{x} = R_1^{-1} Q_1^T \boldsymbol{b}. \tag{1.6.3}$$

Solving least square problem via QR decomposition is more stable because $\kappa(A) = \kappa(R_1)$.

1.6.3 Singular Value Decomposition (SVD)

Any $A \in \mathbb{R}^{N\times M}$ can be factored into

$$A = U\Sigma V^\top, \tag{1.6.4}$$

where $U \in \mathbb{R}^{N\times N}$ and $V \in \mathbb{R}^{M\times M}$ are orthogonal matrices. $\Sigma \in \mathbb{R}^{N\times M}$ is a diagonal matrix with non-negative diagonal entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_M \geq 0$. The numbers σ are called singular values of A. Using this we now define A^+ as the **pseudoinverse** of A, which is given by

$$A^+ = V\Sigma^+ U^\top$$

For rank deficient matrix ($\text{rank}(\Sigma) = r$), we define a More-Penrose pseudoinverse of Σ as follows

$$\Sigma = \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_r & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix} \longrightarrow \Sigma^+ = \begin{bmatrix} \sigma_1^{-1} & & & & \\ & \ddots & & & \\ & & \sigma_r^{-1} & & \\ & & & \ddots & \\ & & & & 0 \end{bmatrix}$$

Note: The Matrix Σ^+ has the following properties: $\text{rank}(\Sigma) = \text{rank}(\Sigma^+) = r$, and $(\Sigma^+)^+ = \Sigma$.

The SVD solution of $A\boldsymbol{x} = \boldsymbol{b}$ is given by:

$$\bar{x} = V\Sigma^+ U^\top \boldsymbol{b} \tag{1.6.5}$$

The columns of U associated with the non-zero singular values ($\sigma_i \neq 0$) form an orthonormal basis for the range of A and the columns of V associated with $\sigma_i = 0$ form an orthonormal basis for the nullspace of A. The singular values σ_i give the lengths of the principal axes of the hyper-ellipsoid defined by $A\boldsymbol{x}$, when $\boldsymbol{x}$ lies on the hypersphere $\|x\|^2 = 1$.

Example 1.6: More-Penrose pseudoinverse

Consider a matrix with dependent rows and columns.

$$A = \begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \text{ with } \sigma_1 > 0, \sigma_2 > 0$$

Column space is the x, y plane, so projecting $\boldsymbol{b} = [b_1, b_2, b_3]^T$ gives $\boldsymbol{p} = P\boldsymbol{b} = [b_1, b_2, o]^T$.

$A\bar{x} = p$ becomes

$$\begin{bmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \bar{x}_3 \\ \bar{x}_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ 0 \end{bmatrix}$$

We conclude that $\bar{x}_1 = b_1/\sigma_1$ and $\bar{x}_2 = b_2/\sigma_2$. The other two must be zero by the requirement of minimum length, so:

$$\bar{x} = \begin{bmatrix} b_1/\sigma_1 \\ b_2/\sigma_2 \\ 0 \\ 0 \end{bmatrix} \quad or \quad A^+\boldsymbol{b} = \begin{bmatrix} \sigma_1^{-1} & 0 & 0 \\ 0 & \sigma_2^{-1} & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Exam checklist

Function fitting vs. interpolation:

- What is a difference between function fitting and interpolation?
- How do we chose between the two?

Linear Least Squares:

- What kind of fitting functions can I use in Linear Least Squares? Can I choose any function?
- What norm do we employ in Least Squares method?
- How does Linear Least Squares solution behaves in the presence of outliers?
- What is a geometrical interpretation of Least Squares solution?
- How do we obtain the normal equations?
- For which special matrices A, can the Linear Least Squares solution be simplified?
- What problems can we encounter when we solve Linear Least Squares with a computer?
- When is the problem $Ax = b$ well or ill-posed?
- What are the QR and SVD decompositions? Why do we use them?

Exercises

Question 1

Consider the following two problems:

1. Given 3D data points $\{x_i, y_i, z_i\}$, $i = 1,\ldots,N$ ($N \gg 3$), we wish to find a surface $z = f(x,y) = a + b\,x + c\,y$ such that we approximate $z_i \approx f(x_i, y_i)$

2. Given 2D data points $\{x_i, y_i\}$, $i = 1,\ldots,N$ ($N \gg 3$), we wish to find a Gaussian function $y = g(x) = A\exp(-(x-\mu)^2/\sigma^2)$ such that we approximate $y_i \approx g(x_i)$

Can we solve the two problems with linear least squares? Motivate your answer. (3 options: we can solve only problem 1, only problem 2 or both)

Solution: In the sense of linear least squares we can only solve problem 1.

Question 2

Reconsider problem 2 from question 1. What happens if we rewrite it by taking the (natural) logarithm of $g(x)$: $\ln(y) = \ln[g(x)] = \ln(A) - (x-\mu)^2/\sigma^2$

Can we use linear least squares to approximate $\ln(y_i) \approx \ln[g(x_i)]$? Motivate your answer. (2 options: yes or no)

Solution: Yes we can. We do least squares for $\ln(y) \approx a\,x^2 + b\,x + c$ and given the parameters a, b, c we can evaluate the 3 unknowns of the Gaussian as follows:

$$\begin{aligned}
&\ln[g(x)] = -\frac{1}{\sigma^2}x^2 + \frac{2\mu}{\sigma^2}x + \ln(A) - \frac{\mu^2}{\sigma^2} \\
&\implies a = -\frac{1}{\sigma^2},\ b = \frac{2\mu}{\sigma^2},\ c = \ln(A) - \frac{\mu^2}{\sigma^2} \\
&\implies \sigma^2 = -\frac{1}{a},\ \mu = -\frac{b}{2a},\ A = \exp\left(c - \frac{b^2}{4a}\right)
\end{aligned}$$

Question 3

Reconsider the last question. What error do we minimise if we approximate $y_i \approx g(x_i)$ with the least squares solution of question 2? Do we minimise $\sum_i \left[y_i - g(x_i)\right]^2$? (2 options: yes or no)

Solution: No, we do not. We minimise

$$\sum_i \{\ln(y_i) - \ln[g(x_i)]\}^2 = \sum_i \left\{ \ln\left[\frac{g(x_i)}{y_i} \right] \right\}^2 .$$

Question 4

Compare the least squares solution of problem 1 of question 1 (3D data) with the least squares solution of question 2. In both cases we will end up with a matrix to invert to solve for the unknown parameters. Which of the two has the bigger matrix to invert? (2 options: Gaussian, 3D, both same)

Solution: Both will end up in a 3×3 matrix to solve as we have 3 unknowns.

Question 5: Extensions to higher dimensions

Imagine that you were given N data points in 3D ($\{x_i, y_i, z_i\}$ with $i = 1 \ldots N$), which were roughly located on a surface (i.e. $z_i = a\,x_i + b\,y_i + c + \xi_i$, where a, b, c are unknown real values and ξ_i is a noise term, which can be imagined as being sampled from a normal distribution). How would you try to estimate the values a, b, c using the data?

Solution: For Least Squares see exercise set 1 question 2.

LECTURE 2

Nonlinear systems I: Bisection, Newton, Secant methods

2.1 Introduction

So far we have looked at equations and systems of equations where the unknown coefficients were entering linearly into the equations. Now, we consider general nonlinear equations and how to numerically solve such equations.

Example 2.1: Population growth

Let $N(t)$ be the number of people at time t. Let λ be the birth rate. Then the equation describing the population growth has the form:

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \lambda N(t).$$

Assume that we also have an immigration at a constant rate u. Then the population growth equation will include an additional term:

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \lambda N(t) + u.$$

The solution of the Eq. (2.1) is given by the following formula:

$$N(t) = N_0 e^{\lambda t} + \frac{u}{\lambda}(e^{\lambda t} - 1),$$

where N_0 is a constant indicating the initial population size.

Numerical example. Suppose we initially have 1,000,000 people. Assume that 15,000 people immigrate every year and that 1,080,000 people are present at the end of the year. What is the birth rate of this population? To find the birth rate we need to solve the following equation for λ (remember that $t = 1$ [year]):

$$1,080,000 = 1,000,000e^{\lambda} + \frac{15,000}{\lambda}(e^{\lambda} - 1).$$

We can reformulate this nonlinear equation into

$$1,000,000e^{\lambda} + \frac{15,000}{\lambda}(e^{\lambda} - 1) - 1,080,000 = 0.$$

A non-linear equation $g(x) = h(x)$ can always be written as $g(x) - h(x) = f(x) = 0$. Like this we transform the problem to finding a root (zero) or the function $f(x)$, i.e.

$$\boxed{f(x^*) = 0.} \tag{2.1.1}$$

This is therefore a **root-finding problem** (see Figure 2.1).

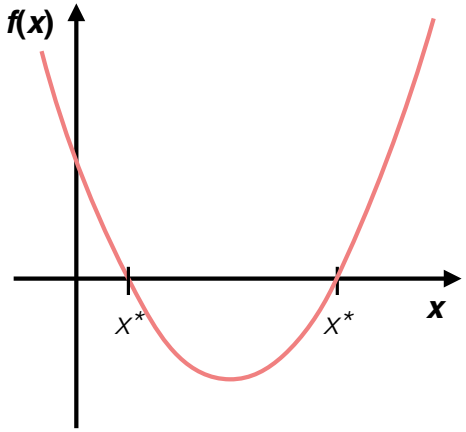


Figure 2.1: Function $f(x)$ is equal to zero at x^*.

In general, there is no formula to find a root of a non-linear function. There is even a proof that for polynomial functions of degree 5 or higher there is no algebraic expression for the roots (Abel–Ruffini theorem). For this reason, the non-linear equations are solved with **iterative schemes** that given initial guess $x^{(0)}$ generate a sequence $x^{(0)}, x^{(1)}, \ldots, x^{(k)}, \ldots$, which is usually written as $\{x^{(k)}\}_{k=0}^{\infty}$.

In practice, we only do finite amount of steps and terminate the algorithm, when some stopping criteria is satisfied. For example, when $|x^{(k)} - x^{(k-1)}| < \text{tol}$, where tol is a user defined tolerance. Therefore, we do not expect to obtain the solution x^* exactly.

2.2 Preliminaries

Before we discuss different algorithms for non-linear equations, we need to consider the following:

- When does a solution to our problem exist theoretically?
- Is there a ways to find this solution numerically?
- How fast will we reach the solution within some tolerance?

Provided there is a root, we want to find a root-finding algorithm to be: efficient (fast converging) and robust (rarely fails to find a solution).

Existence of a root

Recall that for linear problems finding the solution of $A\boldsymbol{x} = \boldsymbol{b}$ required that A is non-singular. To determine this criterion for a non-linear equation, we look at the Intermediate Value Theorem.

Intermediate Value Theorem:
If a function f is continuous on interval $[a,b]$, i.e. $f \in C([a,b])$, then for all K between $f(a)$ and $f(b)$, i.e. $K \in [f(a), f(b)]$, there exists a number $x^* \in (a,b)$ for which $f(x^*) = K$.
Consequence:
If a function $f(x)$ is continuous on interval $[a,b]$ with $f(a)$ and $f(b)$ of opposite sign, i.e., $f(a)f(b) < 0$ then according to the Intermediate Value Theorem there is a $x^* \in (a,b)$ with $f(x^*) = 0$.

Note: There is no simple analog of this theorem for the multi-dimensional case.

Sensitivity and Conditioning

Consider an approximate solution $\tilde{x}$ to $f(x) = 0$, where x^* is the true solution.

$$\left|f(\tilde{x})\right| \approx 0 \quad \overset{?}{\rightarrow} \quad \left|\tilde{x} - x^*\right| \approx 0, \tag{2.2.1}$$

As we have already seen in the linear setting a small residual $\left|f(\tilde{x})\right|$ implies an accurate solution (i.e., $\left|\tilde{x} - x^*\right| \approx 0$) only if the problem is well-conditioned. Note that this is very important, as we generally do not know the true solution x^* apriori and are only able to evaluate $\left|f(\tilde{x})\right|$.

well-conditioned A small change in the "input" causes a small change in the "output".

ill-conditioned A small change in the "input" causes a large change in the "output".

We can measure the behaviour of the function by the condition number:

$$\kappa = \frac{|\delta y|}{|\delta x|} = \frac{|f(x+\delta x) - f(x)|}{|\delta x|}, \tag{2.2.2}$$

where δx is the change in the input and $\delta y = f(x+\delta x) - f(x)$ is the change in the output for $y = f(x)$. Assuming that δx is small we can expand $f(x+\delta x)$ using a Taylor series $f(x+\delta x) \approx f(x) + f'(x)\delta x$, where we have neglected higher order terms. Therefore we can conclude that $\delta y \approx f'(x)\delta x$ and $\kappa = |f'(x)|$. Carefull, this is a condition number for the forward evaluation of the function, i.e. when given x we evaluate $y = f(x)$.

The root-finding problem is a reverse problem to function evaluation. For a root x^* of a function $f(x)$, we need to evaluate $x^* = f^{-1}(0)$, where f^{-1} is an inverse function. Using the previous result and the inverse function theorem we find the condition number for our root finding problem as

$$\kappa = \left| \frac{\partial}{\partial y} \left[f^{-1}(y) \right] \right| = \frac{1}{|f'(x^*)|}. \tag{2.2.3}$$

We see that for small $|f'(x^*)|$ the root-finding problem is ill-conditioned. Small $|f'(x^*)|$ means that the tangent line is nearly horizontal, see Figure 2.2.

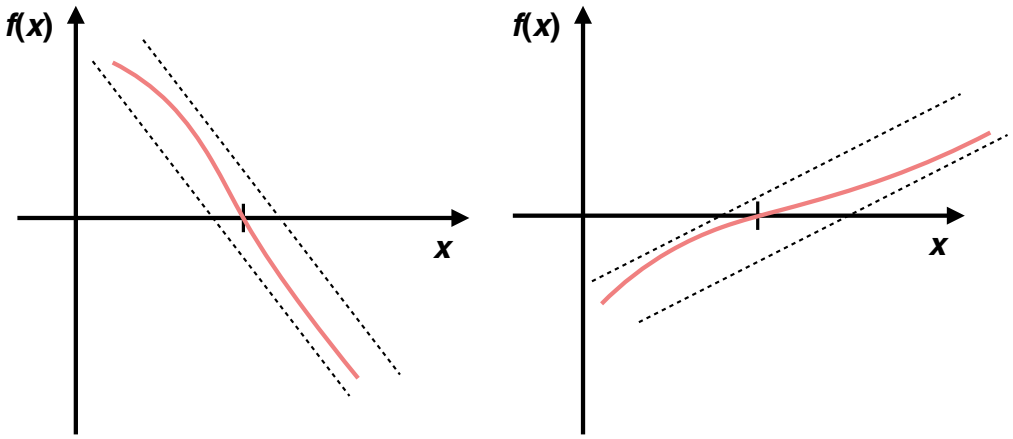


Figure 2.2: The well (left) and ill-conditioned (right) roots.

Note: If $f'(x^*) = 0$, the problem is ill-conditioned. This is the case for roots with multiplicity $m > 1$. Thus, roots with multiplicity $m > 1$ are ill-conditioned[1] (see Figure 2.3 for examples).

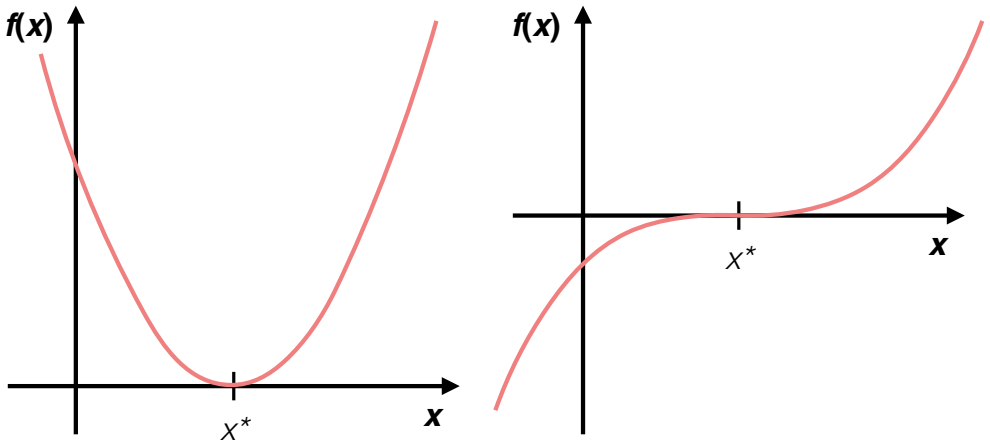


Figure 2.3: Graph of the function $f(x) = x^2 - 2x + 1 = (x-1)^2$ (left), which has a root $x^* = 1$ with $m = 2$ and $f(x) = x^3 - 3x^2 + 3x - 1 = (x-1)^3$ (right), which has a root $x^* = 1$ with $m = 3$.

[1] Recall that for a root of order k we have $f(x) = f'(x) = \cdots = f^{(k-1)}(x) = 0$ and $f^{(k)}(x) \neq 0$

Convergence Rate

We would like that the sequence $\{x^{(k)}\}_{k=0}^{\infty}$ produced by an algorithm converges to a true solution x^* as fast as possible. In order to quantify the rate of convergence we regard the error for $x^{(k)}$, given by

$$E^{(k)} = x^{(k)} - x^* \tag{2.2.4}$$

If the sequence of $x^{(k)} \overset{k\to\infty}{\longrightarrow} x^*$ we know that there is some r, such that the following exists

$$\boxed{\lim_{k\to\infty} \frac{|E^{(k+1)}|}{|E^{(k)}|^r} = C} \tag{2.2.5}$$

The value $r \in \mathbb{R}_{\geq 1}$ is the **order of convergence** and the constant $C \in [0,\infty)$ is the **convergence/asymptotic error constant**. Common cases according to the value of r:

- $r = 1$: linear convergence for $(0 < C < 1)$; if $C = 0$ superlinear; if $C = 1$ sublinear
- $r = 2$: quadratic

Example 2.2: Convergence Rates

1. $x^{(k)} = \frac{1}{2^k}$, $\lim_{k\to\infty} x^{(k)} = x^* = 0$

$$\lim_{k\to\infty} \frac{|x^{(k+1)}-x^*|}{|x^{(k)}-x^*|^r} = \lim_{k\to\infty} \frac{(2^k)^r}{2^{k+1}} = \begin{cases} \frac{1}{2}, & r = 1 \\ \infty, & r > 1 \end{cases} \longrightarrow \text{Linear}$$

2. $x^{(k)} = \frac{1}{k!}$, $\lim_{k\to\infty} x^{(k)} = x^* = 0$

$$\lim_{k\to\infty} \frac{|x^{(k+1)}-x^*|}{|x^{(k)}-x^*|^r} = \lim_{k\to\infty} \frac{(k!)^r}{(k+1)!} = \begin{cases} 0, & r = 1 \\ \infty, & r > 1 \end{cases} \longrightarrow \text{Superlinear}$$

3. $x^{(k)} = \frac{1}{k^2}$, $\lim_{k\to\infty} x^{(k)} = x^* = 0$

$$\lim_{k\to\infty} \frac{|x^{(k+1)}-x^*|}{|x^{(k)}-x^*|^r} = \lim_{k\to\infty} \frac{(k^2)^r}{(k+1)^2} = \begin{cases} 1, & r = 1 \\ \infty, & r > 1 \end{cases} \longrightarrow \text{Sublinear}$$

4. $x^{(k)} = \frac{1}{2^{2^k}}$, $\lim_{k\to\infty} x^{(k)} = x^* = 0$

$$\lim_{k\to\infty} \frac{|x^{(k+1)}-x^*|}{|x^{(k)}-x^*|^r} = \lim_{k\to\infty} \frac{(2^{2^k})^r}{2^{2^{k+1}}} = \lim_{k\to\infty} \frac{(2^{2^k})^r}{(2^{2^k})^2} = \begin{cases} 0, & 1 \leq r < 2 \\ 1, & r = 2 \\ \infty, & r > 2 \end{cases} \longrightarrow \text{Quadratic}$$

2.3 Bisection Method

The bisection method is based on the intermediate value theorem. The method works by beginning with an initial interval and then halving the length until a solution has been isolated as accurately as desired. Thereby the new interval is chosen such that the two points always have opposite sign (see Figure 2.4 and Algorithm 1).

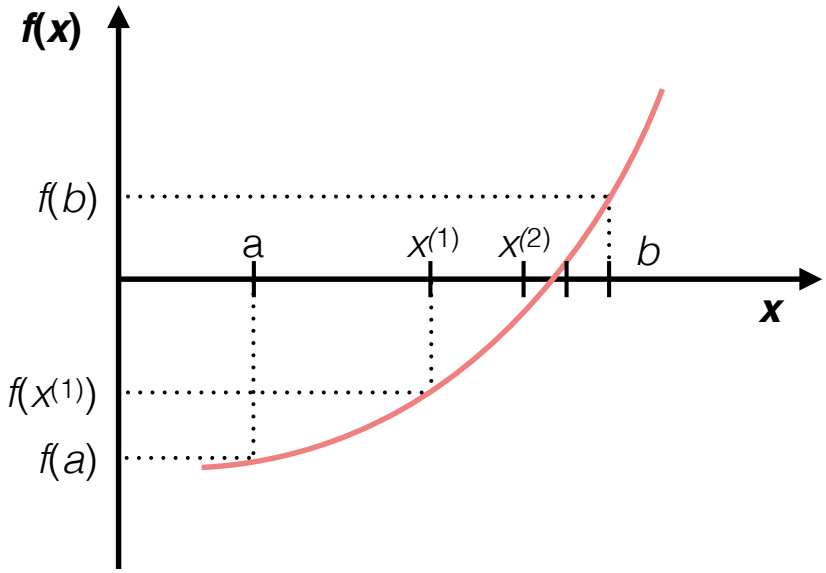


Figure 2.4: Iterative approach used in the bisection method.

Convergence rate

At each iteration the worst case for the error, namely the length of the interval is reduced by 1/2, i.e. $|E^{(k)}| = |x^{(k)} - x^*| \leq (b-a)/2^k$, where $[a,b]$ is the starting interval. Taking this worst case we find

$$\lim_{k\to\infty} \frac{|E^{(k+1)}|}{|E^{(k)}|^r} = \lim_{k\to\infty} \frac{2^{rk}}{2^{k+1}} = \begin{cases} \frac{1}{2}, & r = 1 \\ \infty, & \text{else} \end{cases} \tag{2.3.1}$$

Comparing Eq. (2.3.1) with Eq. (2.2.5) we observe that $r = 1$ and $C = 0.5$, i.e., the bisection method converges linearly. In terms of binary numbers, one bit of accuracy is gained in the approximate solution for each iteration of the bisection method.

Given the starting interval $[a,b]$ and an error tolerance tol, we can compute the required number of iterations k to achieve the required tolerance (regardless of f)

$$|E^{(k)}| = \text{tol} \quad \rightarrow \quad \frac{b-a}{2^k} = \text{tol} \quad \rightarrow \quad k = \log_2\left(\frac{b-a}{\text{tol}}\right). \tag{2.3.2}$$

Algorithm 1 Bisection Method

Input:

a, b, {initial interval}
tol, {tolerance}
$k_{\max}$, {maximal number of iterations}

Output:

$x^{(k)}$, {approximate solution after k iterations}

Steps:

$k \leftarrow 1$
while $\left(b-a\right) > tol$ and $k < k_{\max}$ **do**
  $x^{(k)} \leftarrow (a+b)/2$
  if $\text{sign}(f(a)) = \text{sign}(f(x^{(k)}))$ **then**
    $a \leftarrow x^{(k)}$
  else
    $b \leftarrow x^{(k)}$
  end if
  $k \leftarrow k+1$
end while

Pros

- The bisection method is certain to converge.
- The bisection method makes no use of actual function values, only of their signs.
- The function doesn't need to be differentiable, the only assumption on f is that it is continuous

Cons

- The convergence is slow.
- The initial interval needs to be known beforehand.
- Can not be easily generalized to higher dimensions and many unknowns

2.4 Newton's Method

Assume a function f is differentiable and has a zero at x^*. Furthermore, let $x^{(k)}$ be an approximation to x^* such that $f'(x^{(k)}) \neq 0$ and $|x^* - x^{(k)}|$ is small. We can then expand the function f about x^* using

Taylor series

$$0 = f(x^*) \approx f(x^{(k)}) + f'(x^{(k)})(x^* - x^{(k)}). \tag{2.4.1}$$

Solving the Eq. (2.4.1) for x^* only gives the true solution for linear functions, since we did only keep the terms up to this order

$$x^* \approx x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}. \tag{2.4.2}$$

Nonetheless we will use this results in order to update $x^{(k)}$. Given the initial guess $x^{(0)}$ Newton's method generates a sequence $\{x^{(k)}\}_{k=0}^{\infty}$ where

$$\boxed{x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}}. \tag{2.4.3}$$

Graphically the procedure is visualized in Figure 2.5. Newton's method approximates the non-linear function f by the tangent line to f at $x^{(k)}$, which can be easily seen when realizing that $f'(x^{(0)}) = \tan(\theta) = \frac{f(x^{(0)})}{x^{(0)} - x^{(1)}}$.

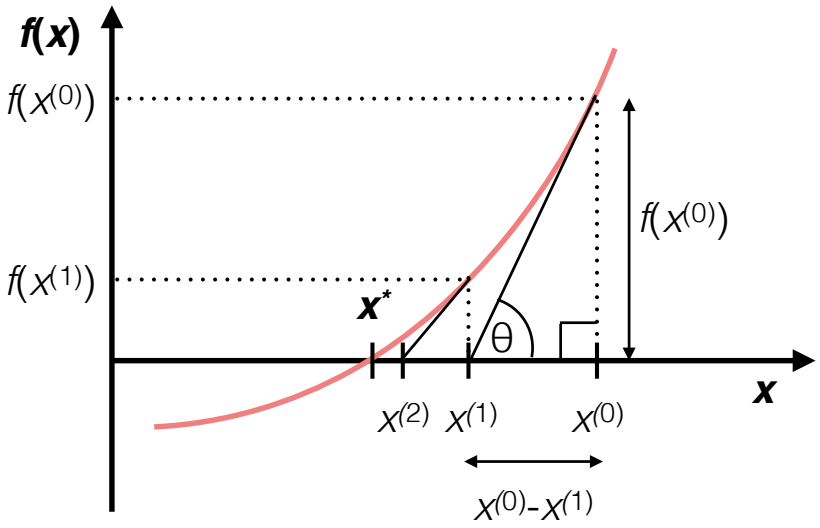


Figure 2.5: Graphical example of Newton iterations as we follow the tangents of $f(x)$.

Convergence rate

In order to compute the error let us consider the case of a differentiable function f on some interval $[a,b]$ that has a simple root $x^* \in [a,b]$ ($f(x^*) = 0$ and $f'(x^*) \neq 0$). Expanding this function in Taylor

series around the root gives

$$\begin{aligned}
\overbrace{f(x^*)}^{0} &\approx f(x^{(k)}) + f'(x^{(k)})(x^* - x^{(k)}) + \frac{1}{2}f''(x^{(k)})(x^* - x^{(k)})^2 \\
0 &\approx \underbrace{\frac{f(x^{(k)})}{f'(x^{(k)})} - x^{(k)}}_{-x^{(k+1)}} + x^* + \frac{f''(x^{(k)})}{2f'(x^{(k)})}(x^* - x^{(k)})^2 \\
0 &\approx \underbrace{x^* - x^{(k+1)}}_{-E^{(k+1)}} + \frac{f''(\xi)}{2f'(x^{(k)})}\underbrace{(x^* - x^{(k)})^2}_{[E^{(k)}]^2} \\
E^{(k+1)} &\approx \frac{f''(\xi)}{2f'(x^{(k)})}[E^{(k)}]^2
\end{aligned} \tag{2.4.4}$$

In the limit $k \to \infty$ this allows us to compute the convergence rate as

$$\lim_{k\to\infty} \frac{|E^{(k+1)}|}{|E^{(k)}|^2} = \lim_{k\to\infty} \frac{|f''(x^{(k)})|}{2|f'(x^{(k)})|} = \frac{|f''(x^*)|}{2|f'(x^*)|} = C < \infty \quad \to \quad r = 2 \tag{2.4.5}$$

Thus in this case Newton's method will converge quadratically. It turns out that for roots of multiplicity $m > 1$, the convergence (if the method is converging) is linear unless we modify the iteration (Eq. (2.4.3)) to $x^{(k+1)} = x^{(k)} - mf(x^{(k)})/f'(x^{(k)})$.

Pros

- Quadratic convergence

Cons

- The method is not guaranteed to converge. It is sensitive to initial conditions.
- If for some k the $f'(x^{(k)}) = 0$ we cannot proceed.
- Newton's method requires the evaluation of both function and derivative at each iteration.

2.5 Secant Method

Evaluating the derivative may be expensive or inconvenient. The key idea of the Secant method is to replace the derivative in Newton's method with a numerical approximation. Let

$$f'(x^{(k)}) \approx \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \tag{2.5.1}$$

then the method reads as

$$\boxed{x^{(k+1)} = x^{(k)} - f(x^{(k)})\frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}} \tag{2.5.2}$$

The secant method approximates the non-linear function f by a secant line through previous two iterations (see Figure 2.6).

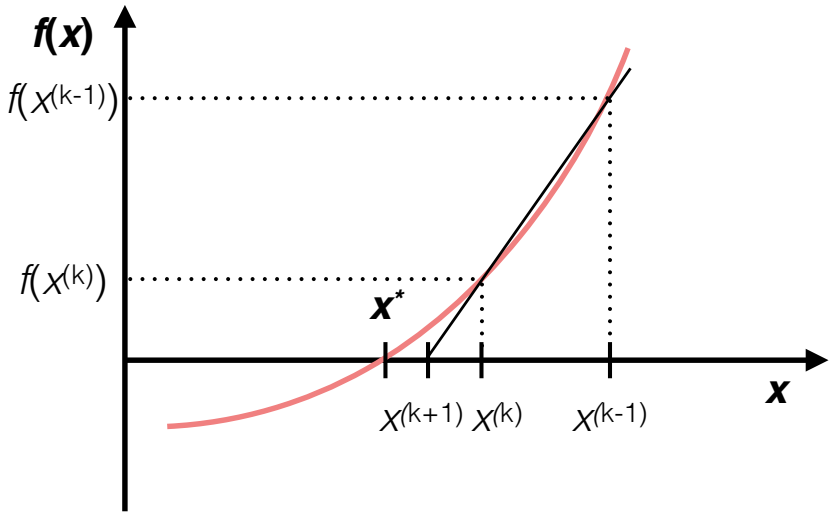


Figure 2.6: Iterative approach used in the secant method.

Convergence rate

It can be shown that the convergence rate of the Secant method for simple roots is $r \approx 1.618$, i.e., between linear and quadratic[2]. As we saw above the secant method uses a linear approximation of the function to construct its derivative. This idea can be extended so that higher order interpolations. Quadratic interpolation (Müller's method) has convergence $r \approx 1.839$.

Pros

- We avoid the evaluation of the derivative.
- At each step, we need to perform 1 function evaluation.

Cons

- The convergence rate is not quadratic.
- We need two initial approximations.

[2]For the interested readers more details about the convergence of the secant method can be found in P. Díez, A note on the convergence of the secant method for simple and multiple roots, Applied Mathematics Letters, Volume 16, Issue 8, 2003, Pages 1211-1215, https://doi.org/10.1016/S0893-9659(03)90119-4

Example 2.3: Comparing Bisection, Newton and Secant methods

Consider the function $f(x)$ given by

$$f(x) = 2\cosh(x/4) - x$$

The Bisection (starting with [0,10]), Newton's (with $x^{(0)} = 8$) and Secant methods (with $x^{(0)} = 10$ and $x^{(1)} = 8$) will give us the following sequence

k	0	1	2	3	4	5	6
Bisection: $f(x^{(k)})$	-1.22	0.96e-1	0.85	0.35	0.12	0.95e-2	-0.43e-1
Newton: $f(x^{(k)})$	-4.76e-1	8.43e-2	1.56e-3	5.65e-7	7.28e-14	1.78e-15	
Secant: $f(x^{(k)})$	2.26	-4.76e-1	-1.64e-1	2.45e-2	-9.93e-4	-5.62e-6	1.30e-9

We see that for the Bisection the error is approximately half of the error in the previous step. For Newton, the accurate digits essentially double at each iteration, while for the Secant method the accurate digits increase more than linearly but less than quadratically.

Exam checklist

- What are the key algorithms for solving non-linear equations?
- How do we define a convergence rate of an algorithm?
- When is a root-finding problem ill-conditioned?
- Which statements are true for investigated algorithms. (i) The algorithm is efficient. (ii) The algorithm always finds a solution. (iii) The algorithm requires a continuous function. (iv) The algorithm requires an evaluation of the derivative of the function.
- Suppose Newton's method does not converge. Does this mean that there is no root in the vicinity of the starting point?
- When is Newton's method converging linearly?
- When to use derivatives or their approximation in Newton's methods?
- What is a graphical interpretation of Newton's and Secant methods?
- How fast will Newton's/Secant method converge for a linear function? Will it converge for any initial value?
- Can you suggest a method that will in part overcome the disadvantage of Bisection in terms of slow convergence and at the same time overcome the disadvantage of Newton's method of local convergence?

Exercises

Use Newton's method to find the roots of the following nonlinear functions.

1. $f(x) = x^3 - 2x^2 - 11x + 12$ using different initial values $x^{(0)} = 2.352875270, 2.352836327$, and 2.352836323.
   Solution:
   Using different initial conditions, the method will converge to different roots: $4, -3$ and 1. This example, demonstrates Newton's method sensitivity to initial conditions.

2. $f(x) = x^3 - 2x^2 + 2$ using initial value $x^{(0)} = 0$.
   Solution:
   We obtain a sequence: $0, 1, 0, 1, \ldots$ Newton's method is stuck in a cycle.

3. $f(x) = x^2$ using initial value $x^{(0)} = 1$.
   Solution:
   The first derivative is $f'(x) = 2x$, which is 0 at x^*. The second derivative is $f''(x) = 2$. The iteration update is in this case:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} = x^{(k)} - \frac{[x^{(k)}]^2}{2x^{(k)}} = \frac{x^{(k)}}{2}$$

   From Eq. (2.4.5) we see that

$$\lim_{x \to 0} \frac{|f''(x)|}{2|f'(x)|} = \lim_{x \to 0} \frac{2}{2|2x|} = \infty$$

   This example, demonstrates the linear convergence for Newton's method in the case of a multiple root ($f'(x^*) = 0$).

LECTURE 3

Nonlinear systems II: set of equations

3.1 Introduction

We now want to generalize the discussion from the previous chapter to find the solution for a general system of N non-linear equations $f_i(\boldsymbol{x})$ $(i = 1, \ldots, N)$, where $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ is a vector of N unknowns. We wish to find $\boldsymbol{x}^*$, such that

$$f_i(\boldsymbol{x}^*) = 0, \qquad i = 1, \ldots, N \tag{3.1.1}$$

We write this system of equations as

$$\boldsymbol{F}(\boldsymbol{x}^*) = \begin{pmatrix} f_1(\boldsymbol{x}^*) \\ f_2(\boldsymbol{x}^*) \\ \vdots \\ f_N(\boldsymbol{x}^*) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \boldsymbol{0} \tag{3.1.2}$$

where $\boldsymbol{F} : \mathbb{R}^N \to \mathbb{R}^N$ and the components $f_i : \mathbb{R}^N \to \mathbb{R}$.

Example 3.1: Required pressure to sink an object

The amount of pressure required to sink a large heavy object in a soft homogeneous soil that lies above a hard base soil can be predicted by the amount of pressure required to sink smaller objects.

Let p denote the amount of pressure required to sink a circular plate of radius r at a distance d in the soft soil, where the hard soil lies at a distance $D > d$ below the surface. p can be approximated by an equation of the form:

$$p = k_1 e^{k_2 r} + k_3 r, \tag{3.1.3}$$

where k_1, k_2, k_3 depend on d and the consistency of the soil but not on r.

Now the task is to find k_1, k_2, k_3. To do that we need to have 3 equations which we can obtain by taking plates of different radii r_1, r_2, r_3 and sinking them. After doing so we will get the following

system of equations:

$$\begin{aligned} p_1 &= k_1 e^{k_2 r_1} + k_3 r_1, \\ p_2 &= k_1 e^{k_2 r_2} + k_3 r_2, \\ p_3 &= k_1 e^{k_2 r_3} + k_3 r_3. \end{aligned} \tag{3.1.4}$$

As in the previous Lecture, we will solve this system with iterative algorithms that given an initial guess $\boldsymbol{x}^{(0)}$ compute a sequence $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(k)}, \ldots$ that hopefully converge to the true solution $\boldsymbol{x}^*$. Note that if $N = 1$, our problem simplifies to the case of a single non-linear equation.

Some of the algorithms that we have encountered in the previous Lecture can be extended to a system of non-linear equations but not all. In particular, the Bisection method cannot be extended, but Newton's and Secant methods can be.

Taylor series for vector functions

To extend the algorithms for a system of non-linear equations we first need an extension of scalar Taylor Series theorem to vector functions.

If $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ and $\boldsymbol{F} = (f_1, \ldots, f_N)^T$, which has bounded derivatives up to order at least two, then for a direction vector $\boldsymbol{y} = (y_1, \ldots, y_N)^T$ the Taylor expansion is given by

$$f_i(\boldsymbol{x} + \boldsymbol{y}) = f_i(\boldsymbol{x}) + \sum_{j=1}^{N} \frac{\partial f_i(\boldsymbol{x})}{\partial x_j} y_j + \mathcal{O}(\|\boldsymbol{y}\|^2) \tag{3.1.5}$$

or

$$\boldsymbol{F}(\boldsymbol{x} + \boldsymbol{y}) = \boldsymbol{F}(\boldsymbol{x}) + J(\boldsymbol{x})\boldsymbol{y} + \mathcal{O}(\|\boldsymbol{y}\|^2) \tag{3.1.6}$$

where J is the **Jacobian matrix**. It is an $N \times N$ matrix with elements $J(\boldsymbol{x})_{i,j} = \frac{\partial f_i(\boldsymbol{x})}{\partial x_j}$:

$$J(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\boldsymbol{x})}{\partial x_N} \\ \frac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\boldsymbol{x})}{\partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_N(\boldsymbol{x})}{\partial x_1} & \frac{\partial f_N(\boldsymbol{x})}{\partial x_2} & \cdots & \frac{\partial f_N(\boldsymbol{x})}{\partial x_N} \end{pmatrix} \tag{3.1.7}$$

Graphically, we can think of $\boldsymbol{x}$ as a point in $\mathbb{R}^N$ and of $\boldsymbol{y}$ as a direction vector. The point $\boldsymbol{x} + \boldsymbol{y}$ is obtained by moving from a point $\boldsymbol{x}$ in the direction $\boldsymbol{y}$. If we compare this Taylor series expansion with the one for scalar functions, i.e., $f(x + y) = f(x) + f'(x)y + \mathcal{O}(y^2)$ we see that the derivative f' is for vector functions replaced by the Jacobian matrix J.

Condition number

For a system of N non-linear equations and N unknowns, the condition number of the root finding problem for root $\boldsymbol{x}^*$ of $\boldsymbol{F}$ is $\|J^{-1}(\boldsymbol{x}^*)\|$ where J is the $N \times N$ Jacobian matrix.

3.2 Newton's method

As before, we derive the Newton's method by expanding $\boldsymbol{F}$ around $\boldsymbol{x}^*$

$$\overbrace{\boldsymbol{F}(\boldsymbol{x}^*)}^{0} \approx \boldsymbol{F}(\boldsymbol{x}^{(k)}) + J(\boldsymbol{x}^{(k)})(\boldsymbol{x}^* - \boldsymbol{x}^{(k)}), \tag{3.2.1}$$

where we have kept only two terms since we assume that $|\boldsymbol{x}^* - \boldsymbol{x}^{(k)}|$ is small. Solving for $\boldsymbol{x}^*$ gives us the expression for the update rule

$$\boxed{\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - J^{-1}(\boldsymbol{x}^{(k)})\boldsymbol{F}(\boldsymbol{x}^{(k)})} \tag{3.2.2}$$

In practice, we never invert $J(\boldsymbol{x}^{(k)})$. Instead we solve

$$J(\boldsymbol{x}^{(k)})\boldsymbol{y}^{(k)} = -\boldsymbol{F}(\boldsymbol{x}^{(k)}) \tag{3.2.3}$$

for $\boldsymbol{y}^{(k)}$ and update as $\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)}$ as in the pseudocode Algorithm 2. To sum up: we start with an initial approximation $\boldsymbol{x}^{(0)}$ and iteratively improve the approximation by computing new approximations $\boldsymbol{x}^{(k)}$. The method succeeds when $\boldsymbol{x}^{(k)}$ is close to $\boldsymbol{x}^*$.
We note that for a linear system, we have $\boldsymbol{F}(\boldsymbol{x}) = A\boldsymbol{x} - \boldsymbol{b}$ and $J(\boldsymbol{x}) = A$. A single iteration of Newton's method then computes

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - J^{-1}(\boldsymbol{x}^{(0)})\boldsymbol{F}(\boldsymbol{x}^{(0)}) = \boldsymbol{x}^{(0)} - A^{-1}A\boldsymbol{x}^{(0)} + A^{-1}b = A^{-1}b, \tag{3.2.4}$$

which gives the same solution as when simply solving the system $A\boldsymbol{x} = \boldsymbol{b}$.

Convergence rate

The convergence of Newton's method for systems of equations is quadratic, provided that the Jacobian matrix is non-singular.

Computational Cost

For large N and dense Jacobians J, the cost of Newton's method can be substantial. It costs $O(N^2)$ to build the matrix J and solving the linear system in Eq. (3.2.3) costs $O(N^3)$ operations (if J is a dense matrix).

Algorithm 2 Newton's method

Input:

$\boldsymbol{x}^{(0)}$, {vector of length N with initial approximation}
tol, {tolerance: stop if $\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)}\| < tol$}
k_{max}, {maximal number of iterations: stop if $k > k_{max}$}

Output:

$\boldsymbol{x}^{(k)}$, {solution of $\boldsymbol{F}(\boldsymbol{x}^{(k)}) = \boldsymbol{0}$ within tolerance tol} (or a message if $k > k_{max}$ reached)

Steps:

$k \leftarrow 0$
while $k \leq k_{max}$ **do**
  Calculate $\boldsymbol{F}(\boldsymbol{x}^{(k)})$ and $N \times N$ matrix $J(\boldsymbol{x}^{(k)})$
  Solve the $N \times N$ linear system $J(\boldsymbol{x}^{(k)})\boldsymbol{y} = -\boldsymbol{F}(\boldsymbol{x}^{(k)})$
  $\boldsymbol{x}^{(k+1)} \leftarrow \boldsymbol{x}^{(k)} + \boldsymbol{y}$
  if $\|\boldsymbol{y}\| < tol$ **then**
    break
  end if
  $k \leftarrow k + 1$
end while

Example 3.2: Geometric interpretation

We have N hyper-surfaces each with a tangent hyperplane (of dimension $N-1$) at $\boldsymbol{x}^{(k)}$. The surfaces and the 0 plane intersect at $\boldsymbol{x}^*$. The tangent planes, on the other hand, intersect with 0 plane at $\boldsymbol{x}^{(k+1)}$.

To understand this, consider the case of 2 equations ($f(x,y) = 0$ and $g(x,y) = 0$) and 2 unknowns (x,y). Then the equation $\boldsymbol{F}(\boldsymbol{x}^{(k)}) + J\left(\boldsymbol{x}^{(k)}\right)(\boldsymbol{x} - \boldsymbol{x}^{(k)}) = 0$ can be written as

$$\begin{pmatrix} f(x^{(k)}, y^{(k)}) \\ g(x^{(k)}, y^{(k)}) \end{pmatrix} + \begin{pmatrix} \frac{\partial f(x^{(k)}, y^{(k)})}{\partial x} & \frac{\partial f(x^{(k)}, y^{(k)})}{\partial y} \\ \frac{\partial g(x^{(k)}, y^{(k)})}{\partial x} & \frac{\partial g(x^{(k)}, y^{(k)})}{\partial y} \end{pmatrix} \begin{pmatrix} x - x^{(k)} \\ y - y^{(k)} \end{pmatrix} = 0$$

or using a short notation $f_x = \partial f(x^{(k)}, y^{(k)})/\partial x$, $f_y = \partial f(x^{(k)}, y^{(k)})/\partial y$

$$f_x(x - x^{(k)}) + f_y(y - y^{(k)}) + f(x^{(k)}, y^{(k)}) = 0$$
$$g_x(x - x^{(k)}) + g_y(y - y^{(k)}) + g(x^{(k)}, y^{(k)}) = 0$$

The two equations each represent a plane. The first equation defines a plane tangent to surface

$z = f(x,y)$ at point $\left(x^{(k)}, y^{(k)}, f\left(x^{(k)}, y^{(k)}\right)\right)$, while the second equation defines the tangent plane to $z = g(x,y)$. The intersection of this planes and $z = 0$ gives us the point $\left(x^{(k+1)}, y^{(k+1)}\right)$, i.e., the new approximation to the solution. If we compare this geometrical interpretation with the one for a single non-linear equation we see that a tangent line is here replaced by a tangent plane.

3.3 Other Methods

The cost of Newton's method can be reduced by:

- using function values at successive iterations to build approximate Jacobians and avoid explicit evaluations of the derivatives (note that this is also necessary if for some reason you cannot evaluate the derivatives analytically)
- update factorization (to solve the linear system) of approximate Jacobians rather than refactoring it each iteration

Modified Newton Method

The simplest approach is to keep J fixed during the iterations of Newton's method. So instead of updating or recomputing Jacobian matrix we compute it once $J^{(0)} = J(x^{(0)})$ and (instead of Eq. (3.2.3)) solve

$$J^{(0)}\boldsymbol{y}^{(k)} = -\boldsymbol{F}(\boldsymbol{x}^{(k)}). \tag{3.3.1}$$

This can be done efficiently by performing a LU decomposition of $J^{(0)}$ once and use it over and over again for different $\boldsymbol{F}(\boldsymbol{x}^{(k)})$. This removes the $O(N^3)$ cost of solving the linear system. We only have to evaluate $\boldsymbol{F}(\boldsymbol{x}^{(k)})$ and can then compute $\boldsymbol{y}^{(k)}$ in $O(N^2)$ operations. This method can only succeed if J is not changing rapidly.

Quasi Newton Method

A method in between Newton and modified Newton is the quasi Newton method which *changes* J at every step without recomputing the derivatives $\partial f_i(\boldsymbol{x})/\partial x_j$ (see Eq. (3.1.7)). It instead updates $J^{(0)} = J(x^{(0)})$ by using information from the step itself.

After the first step we know: $\Delta\boldsymbol{x} = \boldsymbol{x}^{(1)} - \boldsymbol{x}^{(0)}$ and $\Delta\boldsymbol{F} = \boldsymbol{F}(\boldsymbol{x}^{(1)}) - \boldsymbol{F}(\boldsymbol{x}^{(0)})$. So derivatives of the f_i are in the direction of $\Delta\boldsymbol{x}$. Then the next $J^{(1)}$ is adjusted to satisfy:

$$J^{(1)}\Delta\boldsymbol{x} = \Delta\boldsymbol{F} \tag{3.3.2}$$

for example by the rank-1 update

$$J^{(1)} = J^{(0)} + \frac{\left(\Delta\boldsymbol{F} - J^{(0)}\Delta\boldsymbol{x}\right)\left(\Delta\boldsymbol{x}\right)^T}{\left(\Delta\boldsymbol{x}\right)^T\left(\Delta\boldsymbol{x}\right)}. \tag{3.3.3}$$

The advantage of the rank-1 update is that it allows the LU decomposition of $J^{(1)}$ to be computed in $O(N^2)$ given the LU decomposition of $J^{(0)}$. As in the modified Newton's method, this essentially reduces the cost of solving the linear system from $O(N^3)$ to $O(N^2)$.

3.4 Non-Linear Optimization

Can we use the root-finding algorithms to solve an optimization problem? Let's consider a minimization problem

$$\min_{\boldsymbol{x}} E(\boldsymbol{x}), \tag{3.4.1}$$

where we have N unknowns $\boldsymbol{x} = (x_1, \ldots, x_N)^T$ and function $E : \mathbb{R}^N \to \mathbb{R}$ is a scalar function of N variables. Note, that a maximization problem $\max_{\boldsymbol{x}} E(\boldsymbol{x})$ is equivalent to a minimization $\min_{\boldsymbol{x}}[-E(\boldsymbol{x})]$.

Sufficient condition for a critical point of $E(\boldsymbol{x})$ at $\boldsymbol{x}^*$ is

$$\nabla E(\boldsymbol{x}^*) = \begin{pmatrix} \frac{\partial E(\boldsymbol{x}^*)}{\partial x_1} \\ \frac{\partial E(\boldsymbol{x}^*)}{\partial x_2} \\ \vdots \\ \frac{\partial E(\boldsymbol{x}^*)}{\partial x_N} \end{pmatrix} = \mathbf{0}, \tag{3.4.2}$$

where $\nabla E(\boldsymbol{x})$ is a gradient vector of first derivatives of E at $\boldsymbol{x}$. The critical point is a minimum if

$$\nabla^2 E(\boldsymbol{x}^*) = H = \begin{pmatrix} \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1^2} & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_1 \partial x_N} \\ \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_2 \partial x_1} & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_2^2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_2 \partial x_N} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_N \partial x_1} & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_N \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^*)}{\partial x_N^2} \end{pmatrix} \tag{3.4.3}$$

is a positive definite matrix. $\nabla^2 E(\boldsymbol{x})$ is called a **Hessian matrix** H, which contains second derivatives of E at $\boldsymbol{x}$. Note, that for a scalar function $E(x)$ with 1 variable x the condition for a minimum at x^* reduces to $E'(x^*) = 0$ and $E''(x^*) > 0$.

The condition for a critical point yields a system of nonlinear equations

$$\boldsymbol{F}(\boldsymbol{x}) = \nabla E(\boldsymbol{x}) = \mathbf{0}, \tag{3.4.4}$$

that we can solve with root-fining algorithms discussed before, e.g., with Newton's method.

Newton's method

The Jacobian matrix $J(\boldsymbol{x})$ is equal to the Hessian matrix of E

$$J(\boldsymbol{x}) = \nabla^2 E(\boldsymbol{x}). \tag{3.4.5}$$

The Newton's update rule is therefore,

$$\begin{aligned} \nabla^2 E(\boldsymbol{x}^{(k)})\boldsymbol{y}^{(k)} &= -\nabla E(\boldsymbol{x}^{(k)}) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)} \end{aligned} \tag{3.4.6}$$

Steepest descent / gradient descent method

Newton's method has several difficulties: it requires evaluation of the Hessian matrix and a solution of linear system of equations at each iteration, far form the minimum it is unreliable and can diverge from minimum. A simpler method is to move in the local gradient descent direction:

$$\begin{aligned} \boldsymbol{y}^{(k)} &= -\nabla E(\boldsymbol{x}^{(k)}) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \eta \boldsymbol{y}^{(k)} \end{aligned}$$

With this method, we do not need to evaluate the Hessian matrix nor do we need to solve a linear system of equations at each iteration step. However, we must determine the appropriate step size η and the convergence is not as fast as with Newton. Gradient based techniques will always find minima that are close to the starting point (these might be local and not a global minimum).

Levenberg-Marquardt Method

A combination of both methods is proposed by Levenberg-Marquardt method, where far away from minimum a Gradient Descent is performed while close to the minimum a Newton's method is used. With such an interpolates one can overcome the possible divergence of Newton's method. Here we have,

$$\begin{aligned} (\nabla^2 E(\boldsymbol{x}^{(k)}) + \lambda I)\boldsymbol{y}^{(k)} &= -\nabla E(\boldsymbol{x}^{(k)}) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)}, \end{aligned}$$

where λ is a damping parameter that is adjusted at each iteration step. Small λ the algorithm is essentially Newton's method while for large λ the update is in the gradient descent direction. We start with large λ. Then, if the step decreases $E(\boldsymbol{x}^{(k)})$, λ is decreased. If the step increases the error, λ is increased. The Levenberg-Marquardt method is primarily used for the non-linear Least Squares problem.

3.4.1 Non-Linear Least Squares

In the first lecture, we considered a linear Least Squares problem of function fitting where given a set of data $\{t_i, b_i\}_{i=1}^N$ we computed the M unknown parameters in $\boldsymbol{x}$ by solving the system $A\boldsymbol{x} \approx \boldsymbol{b}$. The

fitting function $f(t)$ had parameters $\{x_m\}_{m=1}^M$ that appeared outside of the basis functions. Now, we consider a case where the unknown parameters appear inside the basis functions, i.e.,

$$f(t_i) = \sum_{k=1}^{K} \phi_k(t_i; \boldsymbol{x}_k). \tag{3.4.7}$$

Note that now we may have more parameters $\{x_m\}_{m=1}^M$ than basis functions. The cost function can still be expressed as:

$$E^2 = \sum_{i=1}^{N} [b_i - f(t_i; \boldsymbol{x})]^2 = \left\| \boldsymbol{b} - \boldsymbol{F}(t; \boldsymbol{x}) \right\|^2 \tag{3.4.8}$$

We wish to find the $\bar{x}$ which minimizes $E(\boldsymbol{x}) : \mathbb{R}^M \to \mathbb{R}$.

Example 3.3: Gaussian basis functions

Consider fitting a Gaussian function to data where the unknown parameters are the amplitude α, location μ and variance σ.

$$\phi(t; \alpha, \mu, \sigma) = \alpha \exp\left(\frac{(t-\mu)^2}{2\sigma^2} \right)$$

Here $\boldsymbol{x}$ includes all the coefficients to determine for ϕ, i.e., $\boldsymbol{x} = (\alpha_k, \mu_k, \sigma_k)^T$. Note that fixing μ, σ would result in a linear model.

To employ **Newton's method** we need to evaluate the terms ∇E^2 and $\nabla^2 E^2$ that appear in Eq. (3.4.6). The first term is equal to

$$\begin{aligned} \frac{\partial E^2}{\partial x_k} &= -2 \sum_{i=1}^{N} [b_i - f(t_i; \boldsymbol{x})] \frac{\partial f(t_i; \boldsymbol{x})}{\partial x_k} \\ \nabla E^2 &= -2 J(\boldsymbol{x})^T [\boldsymbol{b} - \boldsymbol{F}(t; \boldsymbol{x})] \end{aligned} \tag{3.4.9}$$

where $J(\boldsymbol{x})$ is a NxM Jacobian matrix of $\boldsymbol{F}$. The second term is

$$\begin{aligned} \frac{\partial^2 E^2}{\partial x_k \partial x_l} &= -2 \sum_{i=1}^{N} \left\{ -\frac{\partial f(t_i; \boldsymbol{x})}{\partial x_k} \frac{\partial f(t_i; \boldsymbol{x})}{\partial x_l} + [b_i - f(t_i; \boldsymbol{x})] \frac{\partial^2 f(t_i; \boldsymbol{x})}{\partial x_l \partial x_k} \right\} \\ \nabla^2 E^2 &= -2[-J(\boldsymbol{x})^T J(\boldsymbol{x}) + L(\boldsymbol{x})] \end{aligned} \tag{3.4.10}$$

where $L(\boldsymbol{x}) = \sum_{i=1}^{N} [b_i - f(t_i; \boldsymbol{x})] \frac{\partial^2 f(t_i; \boldsymbol{x})}{\partial x_l \partial x_k}$. Plugging this expressions into Eq. (3.4.6) gives us

$$\begin{aligned} \left[J(\boldsymbol{x}^{(k)})^T J(\boldsymbol{x}^{(k)}) - L(\boldsymbol{x}^{(k)}) \right] \boldsymbol{y}^{(k)} &= J(\boldsymbol{x}^{(k)})^T [\boldsymbol{b} - \boldsymbol{F}(t; \boldsymbol{x}^{(k)})] \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \boldsymbol{y}^{(k)} \end{aligned} \tag{3.4.11}$$

Since $L(\boldsymbol{x}^{(k)})$ is proportional to the residual between the model and the the data it should eventually be small and can therefore be omitted. This would give us the **Gauss–Newton method**.

For the **Steepest descent method**, we would have

$$\begin{aligned} \boldsymbol{y}^{(k)} &= J(\boldsymbol{x}^{(k)})^T[\boldsymbol{b} - \boldsymbol{F}(t;\boldsymbol{x}^{(k)})] \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{x}^{(k)} + \eta \boldsymbol{y}^{(k)} \end{aligned} \tag{3.4.12}$$

Exam checklist

- What are the key algorithms for solving a system of non-linear equations
- What is the Jacobian for Newton's method and what is its role
- Why do we approximate the Jacobian in Newton's methods
- How many solutions is a nonlinear system of n algebraic equations expected to have?
- Explain how we can use the algorithms for solving a system of non-linear equations in optimization problem
- What are the necessary and the sufficient conditions for a minimum?
- Can we use Newton's method to solve Least Squares problem?

LECTURE 4

Interpolation and extrapolation I: Lagrange interpolation

4.1 Introduction

As we learned in the first lecture function fitting may be understood as a *low order model* that describes the given data. The type of interpolation is equivalent to choosing a particular model and as such it reflects some prior knowledge about the data. The choice of the approximating function as well as the choice (when possible) of the sampling points, represents assumptions that we have made about the structure/architecture of the model.

Recall from the first part of of the lecture that we constructed a fitting function $f(x)$ as follows:

$$f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x) = y \tag{4.1.1}$$

The parameters here are:

- M: the number of terms (can be larger, equal or smaller than the number of data points)
- $\phi_k(x)$: the type of basis function

Here we call x_i are the abscissae, y_i the ordinate of the given data points. Furthermore we denote α_k as our coefficients.

In the first section we took the number of datapoints N much bigger then the number of basis functions M. We learned that in the case $M = N$ and a linearly independent basis function our system is solvable for the coefficients. This allows us to **interpolate** among the data (i.e. interpret the data for unseen values) and in order to **extrapolate** (i.e. predict based on the available data). In this sense extrapolation is equivalent to generalisation.

Example 4.1: Polynomial Function Interpolation Revisited

Consider three data points $(x_i, f(x_i))$ with $i = 1,2,3$ in a two dimensional Cartesian coordinate system given by $(-1,-1)$, $(0,0)$ and $(1,1)$. We want to find a polynomial of degree 2 to interpolate the three data points, that is $f(x) = \alpha_1 + \alpha_2 x + \alpha_3 x^2$. We have a linear system of 3 equations with

3 unknowns to solve

$$\begin{bmatrix} \phi_1(x_1), & \phi_2(x_1) & \phi_3(x_1) \\ \phi_1(x_2), & \phi_2(x_2) & \phi_3(x_2) \\ \phi_1(x_3), & \phi_2(x_3) & \phi_3(x_3) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \end{bmatrix}. \tag{4.1.2}$$

With the given data, we have the follow linear system to solve.

$$\begin{bmatrix} 1, & -1 & 1 \\ 1, & 0 & 0 \\ 1, & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \tag{4.1.3}$$

Although the 3×3 matrix is not sparse, we can still solve it quickly without pen or paper by observing its structure. Let us look at the second row first, we can tell directly $\alpha_1 = 0$. Subsequently, let us look at first and third rows together and remember $\alpha_1 = 0$, we can quickly determine $\alpha_3 = 0$ and $\alpha_2 = 1$. To summarize, the interpolating function we seek is $f(x) = x$.

Now consider the three data points again, plotting them it is obvious that they stay on the same straight line. This observation suggests that the interpolation procedure we just performed indeed exclude the contributions from constant and quadratic functions.

Imagine now that we have N data points and we use, for example, Gaussian elimination method to solve the resultant linear system. This would take $O(N^3)$ operations, which is quite expensive for the purpose of determining an interpolating function. It would be convenient if we can select the basis function so that the matrix of the linear system is diagonal and it takes almost no effort to get the coefficients α for the interpolating function. This motivates the construction of the Lagrangian interpolation method.

4.2 Lagrange Interpolation

One way to do achieve a diagonal matrix is to chose N polynomials of degree $N-1$: $\{l_k(x)\}_{k=1,\ldots,N}$, such that $l_i(x_j) = \delta_{ij}$[1]. This property is satisfied by

$$l_k(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_N)}{(x_k-x_1)(x_k-x_2)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_N)} \tag{4.2.1}$$

Using these functions we construct our interpolation function as

$$f(x) = \sum_{k=1}^{N} \alpha_k l_k(x)$$

[1] Recall that $\delta_{ij} = \begin{cases} 1, & i = j \\ 0, & i \neq j \end{cases}$.

In order to find the coefficients we insert one of our data-points

$$f(x_i) = \sum_{k=1}^{N} \alpha_k l_k(x_i) = \alpha_i = y_i.$$

Where we used the constructive property of the polynomials (i.e. that $l_k(x_k) = 1$ and $l_k(x_i) = 0$ for $k \neq i$). Thus we found our Lagragian interpolation function as

$$\boxed{f(x) = \sum_{k=1}^{N} y_k l_k(x)} \tag{4.2.2}$$

Important Notes:

- The Lagrange polynomials involve single polynomials which pass through all the data points.
- The approximation error of the Lagrange polynomial $f(x)$ of a function $y(x)$ that has been sampled at points (x_k, y_k) and for some $x_1 \leq \xi \leq x_N$ is given by:

  $$\left|y(x) - f(x)\right| = \left|\frac{y^{(n)}(\xi)}{n!} \Pi_{k=1}^{n}(x - x_k)\right|$$

  This term can be minimized if we chose the coordinates x_k of the sampling points to be the roots of the n-degree Chebychev polynomials $T_n(x)$. These points satisfy $T_n(x_k) = 0$ and they can be computed as: $x_k = -\cos(\frac{2k-1}{2n}\pi)$.

Exam checklist

Lagrange interpolation:

- Polynomials with degree $N-1$ for N data points
- Analytical expression from data points
- Sensitivity to noise
- Predictability issues
- Extensions to higher dimensions?

Exercises

Question 1: Lagrange vs Least Squares

We consider N data points in 2D ($\{x_i, y_i\}$ with $i = 1 \ldots N$), where we wish to find a function $f(x)$ such that $y_i \approx f(x_i)$. As an example, imagine that the data is given on a straight line (i.e. $y_i = a\,x_i + b$ for

some real values a and b):

- How do you expect the Lagrange functions ($l_k(x)$ in Eq. (4.2.1)) to look like for $N = 3$? How about larger N?
  Solution: For $N = 3$, the Lagrange functions $l_k(x)$ are polynomials of order $N-1 = 2$ (quadratic). For larger N, the Lagrange functions $l_k(x)$ will always be polynomials of order $N-1$.

- How do you expect the Lagrange interpolator ($f(x)$ in Eq. (4.2.2)) to look like?
  Solution: The Lagrange interpolator is expected to be the equation of a line $f(x) = ax+b$ in every case (no matter how many N points we use). This means, that for an order-1 function (linear) the higher order (quadratic, cubic etc.) terms do not contribute to the interpolator. We provide a simple example in order for this to be clearer: Assume we have to interpolate through the following set of points that are drown from the linear function $f(x) = 2x+1$, $\{x_i, y_i\} = (0,1), (2,5), (4,9)$.
  Using $N = 2$ points:

$$
\begin{aligned}
l_1(x) &= \frac{x-2}{-2} = -\frac{1}{2}(x-2) \\
l_2(x) &= \frac{x-0}{2} = \frac{x}{2} \\
f(x) &= 1(-\frac{1}{2})(x-2) + 5\frac{x}{2} = 2x+1
\end{aligned}
$$

  Using $N = 3$ points:

$$
\begin{aligned}
l_1(x) &= \frac{(x-2)(x-4)}{(0-2)(0-4)} = \frac{(x-2)(x-4)}{8} = \frac{x^2-6x+8}{8} \\
l_2(x) &= \frac{(x-0)(x-4)}{(2-0)(2-4)} = -\frac{x^2-4x}{4} \\
l_3(x) &= \frac{(x-0)(x-2)}{(4-0)(4-2)} = \frac{x^2-2x}{8} \\
f(x) &= 1\frac{x^2-6x+8}{8} - 5\frac{2x^2-8x}{8} + 9\frac{x^2-2x}{8} = 2x+1
\end{aligned}
$$

  The resulting interpolator for $N = 3$ matches the result for $N = 2$. Generalizing, for a linear function, the final interpolator $f(x)$ using $N \geq 3$ points will be the same as computing it with $N = 2$ points.

- How do you expect those functions ($l_k(x)$ and $f(x)$) to change if we add a bit of noise to the data? Adding noise can be imagined as choosing $y_i = ax_i + b + \xi_i$, where ξ_i is sampled from a normal random distribution with mean 0 and standard deviation σ. How do you expect the value of σ to affect the Lagrange interpolator?
  Solution: By adding noise to the data: $y_i = ax_i + b + \xi_i$, $\xi_i \sim \mathcal{N}(0, \sigma^2)$ the Lagrange functions are not varied: $l_k(x)$ is a function of $\{x_i\}$ only. The Lagrange interpolator $f(x)$ is modified:

for data without noise: $y_i = ax_i + b$
for data with noise: $y_i^* = ax_i + b + \xi_i$

$$f(x) = \sum y_k l_k(x) = \sum (ax_k + b) l_k(x)$$
$$f^*(x) = \sum y_k^* l_k(x) = \sum (ax_k + b + \xi_k) l_k(x)$$

The difference in the interpolator is:

$$|f(x) - f^*(x)| = \sum_{k=1}^{N} |\xi_k l_k(x)| \leq \sum |\xi_k| \sum |l_k(x)|$$

We conclude that including even a bit of noise can cause significant oscillations, whereas oscillations become worse with increasing σ. If $\sigma \to 0$, then ξ_k is distributed close to 0, so error is considerably lower.

- Imagine we did a least squares fit to a linear function $y_i \approx a_L x_i + b_L$ and computed a_L, b_L as in Eq. (1.3.5) (note that there we used a slightly different notation). Would you expect to get $a_L = a$ and $b_L = b$, if the data had no noise? What if we added noise to the data?

  Solution: For least squares fit: $y_i = a_{LS} x_i + b_{LS}$. If data has no noise: $a_{LS} = a$, $b_{LS} = b$.
  If we add noise: $y_i^* = a_{LS}^* x_i + b_{LS}^* + \xi_i$. See exercise set 1 question 3:

$$|f(x) - f^*(x)| \leq |a - a_{LS}^*| x + |b - b_{LS}^*|$$
$$|a - a_{LS}^*| \leq \frac{C}{N} \sum_i^N \xi_i$$

Question 2: Computational complexity

Consider again the case of N data points in 2D: $\{x_i, y_i\}$ $(i = 1 \ldots N)$. Estimate the computational complexity in terms of number of basic floating point operations (FLOPs: add, subtract, multiply, divide) to evaluate $f(x)$ in Eq. (4.2.2) for a given value x. Estimate the computational complexity to evaluate the equivalent $f(x) = \sum_{k=1}^{N} \alpha_k \phi_k(x)$ (as in Eq. (4.1.1)) with basis functions $\phi_k(x) = x^{k-1}$ (as in Eq. (**??**)) with given coefficients α_k. Which one requires less FLOPs and hence is faster to evaluate? How many FLOPs do you save if you evaluate a simplified system such as $f(x) = \sum_{k=1}^{M} \alpha_k \phi_k(x)$ with $M < N$?

Solution: To evaluate $f(x) = \sum_{k=1}^{N} y_k l_k(x)$ where $l_k(x) = \prod_{i=1, i\neq k}^{N} \frac{(x - x_i)}{(x_k - x_i)}$ we have

$$f(x) = y_1 \frac{(x - x_2)(x - x_3) \ldots (x - x_N)}{(x_1 - x_2)(x_1 - x_3) \ldots (x_1 - x_N)} + \ldots$$
$$+ y_N \frac{(x - x_1)(x - x_2) \ldots (x - x_{N-1})}{(x_N - x_1)(x_N - x_2) \ldots (x_N - x_{N-1})}$$

For every l_k the $y_k \prod \frac{1}{(x_k - x_i)}$ is a constant and can be precomputed.

$$f(x) = c_1 [(x - x_2)(x - x_3) \ldots (x - x_N)] + \cdots + c_N [(x - x_1)(x - x_2) \ldots (x - x_{N-1})]$$

We need N multiplications for every term and there are N terms (N basis functions). Thus, in total we need N^2 flops.

To evaluate $f(x) = \sum a_k \Phi_k(x)$ where $\Phi_k(x) = x^{k-1}$ we have

$$f(x) = a_N x^{N-1} + a_{N-1} x^{N-2} + \cdots + a_2 x^1 + a_1$$

If we compute x^{k-1} in each step from scratch it takes: $\frac{(N-1)(N-2)}{2}$ multiplications to compute x^{k-1} and $N-1$ multiplications to multiply with all x^{k-1} with a_k. Therefore, overall $\frac{N^2-N}{2}$ multiplications.
If we compute x^{k-1} using x^{k-2} (computed in previous step) we can save flops. We have to do $N-1$ multiplications for all x^{k-1} and $N-1$ multiplications to multiply with all x^{k-1} with a_k. Together, we need $2N-2$ multiplications.

LECTURE 5

Interpolation and extrapolation II: Cubic splines

5.1 Introduction

In the previous chapter we encountered Lagrange polynomials, used to approximate arbitrary functions from a set of data points. These polynomials are appropriate approximations in many occasions. Nonetheless, global polynomials have a hard time following local behaviour, resulting in undesirable artifacts such as overshooting and ringing. Furthermore Lagrange polynomials are high degree polynomials, which are highly oscillatory, a property that may not reflect the functions that are being approximated from the sampled points. Moreover, small fluctuations in the data, even in a small region of the domain, result in very large fluctuations of the approximating function over the entire domain. This fact restricts the use of Lagrange polynomials when approximating data that arise from measurements in many physical and engineering systems.

An alternative approach is to divide the interval into smaller subdomains and then construct *different approximations* in each interval. This is the key idea of **Splines**. Splines are constructed as piecewise polynomials with different parameters in each interval. They belong to a class of locally defined polynomials with appropriate patching conditions. **Cubic splines** use piecewise cubic polynomials and match their values along with first and second derivatives.

Splines are heavily used in computer graphics and design (ship building, aircraft design, CAD software). They are also occasionally used for data fitting and as a tool in numerical methods even though some special care is required at the boundaries.

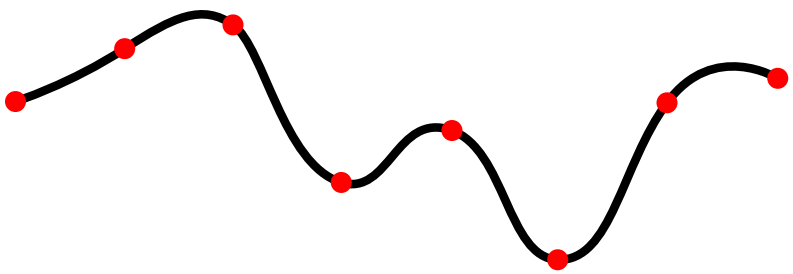


Figure 5.1: Example of a cubic spline fitted through data points (source: wikipedia).

5.2 Cubic Splines

Assume the data is given as $\{x_i, y_i\}_{i=1,\dots,N}$. Now let $f_i(x)$ be a cubic function between the points $x_i \le x \le x_{i+1}$ and $f(x)$ the collection of all $f_i(x)$ for the entire range of $x_1 \le x \le x_N$. We now wish to construct $f(x)$ such that it matches the data as $f(x_i) = y_i$ and such that it has continuous first and second derivatives.

Key Idea The key idea to achieve this is as follows: Since $f(x)$ is piecewise cubic, its second derivative f'' is piecewise linear and will be fully determined if we know $f''(x_i)$ for all i. We therefore introduce the second derivatives as the N unknowns we wish to solve for:

$$f_i'' = f''(x_i) \longrightarrow N \text{ unknowns}$$

Let $\Delta_i = x_{i+1} - x_i$. Since we know that the second derivative is linear we can construct

$$f''(x) = f_i'' \frac{(x_{i+1} - x)}{\Delta_i} + f_{i+1}'' \frac{(x - x_i)}{\Delta_i} \qquad x_i \le x \le x_{i+1}$$

This equation can now be integrated twice in order to obtain $f(x)$. Here the appearing constants of integration can be obtained from

$$y_i = f(x_i) \qquad \text{and} \qquad y_{i+1} = f(x_{i+1})$$

Furthermore assuming continuity for f' we get equations for f_i'' which we can solve.

Derivation Let us make this explicit. Performing the integration gives

$$f'(x) = -f_i'' \frac{(x_{i+1} - x)^2}{2\Delta_i} + f_{i+1}'' \frac{(x - x_i)^2}{2\Delta_i} + C_i, \tag{5.2.1}$$

$$f(x) = f_i'' \frac{(x_{i+1} - x)^3}{6\Delta_i} + f_{i+1}'' \frac{(x - x_i)^3}{6\Delta_i} + C_i(x - x_i) + D_i. \tag{5.2.2}$$

We remark that we kept the form $(x_{i+1} - x)^\alpha$ and $(x - x_i)^\alpha$ for $\alpha = 1, 2$ respectively. This can be done by absorbing the respective terms into the integration constants. We evaluate this at the data points $\{x_i, y_i = f(x_i)\}$ and $\{x_{i+1}, y_{i+1} = f(x_{i+1})\}$ to obtain the constants

$$\begin{aligned} y_i = f(x_i) &= f_i'' \frac{\Delta_i^2}{6} + D_i \\ y_{i+1} = f(x_{i+1}) &= f_{i+1}'' \frac{\Delta_i^2}{6} + C_i\Delta_i + D_i \end{aligned} \tag{5.2.3}$$

This equations can readily be solved for C_i and D_i:

$$C_i = \frac{y_{i+1} - y_i}{\Delta_i} - (f_{i+1}'' - f_i'') \frac{\Delta_i}{6} \tag{5.2.4}$$

$$D_i = y_i - f_i'' \frac{\Delta_i^2}{6} \tag{5.2.5}$$

Now we compute $f'(x)$ (using Eq. (5.2.1) and Eq. (5.2.4)) once with $x_i \le x \le x_{i+1}$ and once $x_{i-1} \le x \le x_i$:

$$\text{for } x_i \le x \le x_{i+1}\text{: } f'(x) = f''_{i+1}\left[\frac{(x-x_i)^2}{2\Delta_i} - \frac{\Delta_i}{6}\right] - f''_i\left[\frac{(x_{i+1}-x)^2}{2\Delta_i} - \frac{\Delta_i}{6}\right] + \frac{y_{i+1}-y_i}{\Delta_i}$$

$$\text{for } x_{i-1} \le x \le x_i\text{: } f'(x) = f''_i\left[\frac{(x-x_{i-1})^2}{2\Delta_{i-1}} - \frac{\Delta_{i-1}}{6}\right] - f''_{i-1}\left[\frac{(x_i-x)^2}{2\Delta_{i-1}} - \frac{\Delta_{i-1}}{6}\right] + \frac{y_i-y_{i-1}}{\Delta_{i-1}}$$

Evaluating the two derivatives at $x = x_i$ and asking that they are equal we get

$$\boxed{\frac{\Delta_{i-1}}{6}f''_{i-1} + \left(\frac{\Delta_{i-1}+\Delta_i}{3}\right)f''_i + \frac{\Delta_i}{6}f''_{i+1} = \frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i-y_{i-1}}{\Delta_{i-1}}} \tag{5.2.6}$$

We can write $N-2$ equations because these equations are not valid at the ends (except for periodic functions). We need 2 additional equations as boundary conditions. Some choices are:

- Natural spline: $f''_1 \equiv f''_N = 0$
- Parabolic runout: Set $f''_1 = f''_2$ and $f''_N = f''_{N-1}$
- Clamping: Set $f'(x_1) = f'(x_N) = 0$

Furthermore we can take arbitrary combinations of the above boundary conditions. All in all we end up with a tri-diagonal system with N equations and N unknowns (the f''_i).

Solution This system can be solved with $\mathcal{O}(N)$ using the Tridiagonalmatrix-Algorithmus (TDMA). To understand this algorithm consider a general tridiagonal matrix

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.2.7}$$

We start by eliminating the lower diagonal elements. This means we devide the first row b_1, multiply by a_2 and subtract it from the second row. This gives

$$\begin{bmatrix} 1 & \frac{c_1}{b_1} & & & 0 \\ & b_2 - a_2\frac{c_1}{b_1} & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} \frac{d_1}{b_1} \\ d_2 - a_2\frac{d_1}{b_1} \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.2.8}$$

We now call the new off-diagonal element in the first row $w_1 = \frac{c_1}{b_1}$ and the element on the right hand side $g_1 = \frac{d_1}{b_1}$. If we continue with the first step of this iterative procedure, namely the division we find

$$\begin{bmatrix} 1 & w_1 & & & 0 \\ & 1 & \frac{c_2}{b_2 - a_2 w_1} & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} g_1 \\ \frac{d_2 - a_1 g_1}{b_2 - a_2 w_1} \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.2.9}$$

Indeed already here we discover a recurrent structure which can be easily checked to hold in general, namely that our updated elements are given by

$$w_i = \begin{cases} \dfrac{c_1}{b_1}, & i = 1 \\ \dfrac{c_i}{b_i - a_i w_{i-1}}, & i = 2,3,\ldots,N-1 \end{cases} \quad \text{and} \quad g_i = \begin{cases} \dfrac{d_1}{b_1}, & i = 1 \\ \dfrac{d_i - a_i g_{i-1}}{b_i - a_i w_{i-1}}, & i = 2,3,\ldots,N \end{cases} \tag{5.2.10}$$

Having this transformation we can directly obtain the solution by starting from x_N, giving

$$x_i = \begin{cases} g_N, & i = N \\ g_i - w_i x_{i+1}, & i = N-1, N-2, \ldots, 1 \end{cases} \tag{5.2.11}$$

As we see the computational complexity is $\mathcal{O}(N)$.

Notes:

- Natural splines are equivalent to mechanical rods or wooden splines used in design. The rod tends to pass through points when fixed on them, trying to minimise the curvature which is equivalent to energy: $E \approx \min \int [f''(x)]^2 dx \Longrightarrow$ cubic spline with natural ends.

Figure 5.2: Wooden spline with hooked weights (so called “ducks”) used to design the hull of a sailing vessel (source: http://www.alatown.com/spline/).

These types of splines have the problem that **moving one point affects all the others** (the system of equations must be solved ane). For an interactive application such as in computer graphics it must be possible to make a local change and not have to recompute the entire curve.

- An alternative to globally defined cubic splines is to patch together piecewise cubic polynomials using four points for each patch. Given four points $\{t_i, y_i\}$ ($i = \{1,2,3,4\}$) we define a cubic function $f(x)$ by using the outer points as end points and the inner points to define the derivatives: $f(t_1) = y_1, f(t_4) = y_4, f'(t_1) = (y_2 - y_1)/(t_2 - t_1), f'(t_4) = (y_4 - y_3)/(t_4 - t_3)$. When drawing such a curve, the user (or the application) can enforce continuity of the first derivatives but enforcing continuity of second derivatives would again require a solution of a system of equations as for the cubic splines. This is the approach used for instance when drawing **Bézier curves**.

5.3 B-splines

B-splines are basis functions of a given degree which can be used to define any other spline (of the same degree). A set of B-splines $B_{i,d,t}(x)$ ($i = 1,\ldots,M$) can be constructed for a given **knot vector** with entries t_j ($j = 1,\ldots,M+d+1$). The knot vector must be ordered and we will first focus on the case where we have no repetitive elements ($t_1 < t_2 < \cdots < t_{M+d+1}$). These values define the intervals of the piecewise polynomial function. The B-spline $B_{i,d,t}(x)$ is recursively constructed as a piecewise polynomials of degree d which is only non-zero for the range $t_i \le x \le t_{i+d+1}$ via:

$$
\begin{aligned}
B_{i,0,t}(x) &= \begin{cases} 1 & \text{if } t_i \le x < t_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \\
B_{i,d,t}(x) &= \frac{x - t_i}{t_{i+d} - t_i} B_{i,d-1,t}(x) + \frac{t_{i+d+1} - x}{t_{i+d+1} - t_{i+1}} B_{i+1,d-1,t}(x).
\end{aligned}
\tag{5.3.1}
$$

The resulting function consists of different piecewise polynomials of degree d for each interval $t_i \le x \le t_{i+1}$. By construction the functions will have **continuous derivatives** up to degree $d-1$.

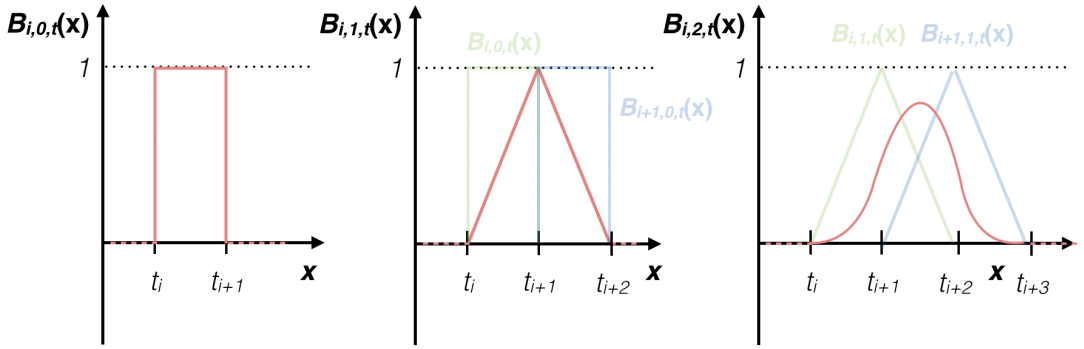


Figure 5.3: Illustration of the construction of the B-splines. The new spline (in red) is obtained from the old (green and blue) by linear interpolation (please see http://geometrie.foretnik.net/files/NURBS-en.swf for accurate plots of the resulting functions).

It is also possible for knots to be repeated, i.e. we the knot vector satisfies $t_1 \leq t_2 \leq \cdots \leq t_{M+d+1}$. A common case is to have a **clamped** effect by setting the first $d+1$ knots to be equal and the last $d+1$ knots to be equal:

$$\{\underbrace{t_1,\ldots,t_{d+1}}_{d+1 \text{ equal knots}},\ \underbrace{t_{d+2},\ldots,t_M}_{M-d-1 \text{ internal knots}},\ \underbrace{t_{M+1},\ldots,t_{M+d+1}}_{d+1 \text{ equal knots}}\} \tag{5.3.2}$$

This choice of knots ensures that the first and last B-splines are 1 on the first and last knot respectively ($B_{1,d,t}(t_1) = B_{M,d,t}(t_{M+d+1}) = 1$), while all the other B-splines are 0 for those knots ($B_{i,d,t}(t_1) = B_{i,d,t}(t_{M+d+1}) = 0$, for $i = 2,\ldots,M-1$). As long as the $M-d-1$ internal knots are different from each other, the resulting splines still have continuous derivatives up to degree $d-1$. In Eq. (5.3.1), multiplied knots can lead to terms of the form $0/0$, which are resolved such that $0/0 = 0$.

Any spline $S_{d,t}(x)$ with piecewise polynomials of degree d can be written as:

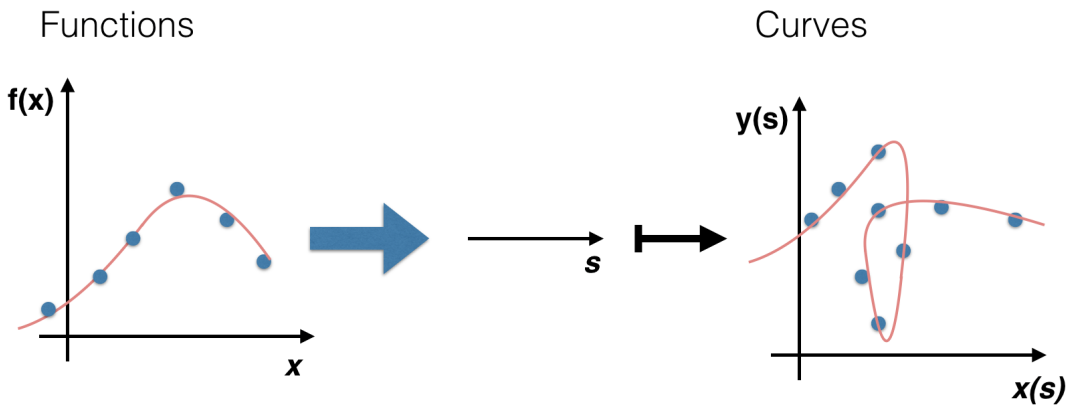
$$S_{d,t}(x) = \sum_{i=1}^{M} \alpha_i B_{i,d,t}(x), \quad \text{for} \quad t_{d+1} \leq x < t_{M+1} \tag{5.3.3}$$

Here the factors α_i are free parameters to be determined. Given N data points as $\{x_i, y_i\}$ ($i = 1,\ldots,N$ with $N \geq M$), we can use the B-splines to find the parameters α_i in Eq. (5.3.3) in the sense of linear least squares. If $N = M$ we can also force spline $S_{d,t}(x)$ to go through all our data points. We note though that the choice of the knots t_i is not obvious when B-splines are used for function fitting. Also the parameters are again coupled as with the cubic splines so that if we change one data point, we must recompute all parameters. A solution to this problem is given by curve fitting using **NURBS**.

5.4 NURBS

When we wish to approximate curves, it is hopeless to try to identify suitable approximating functions such as that $f(x) \approx y(x)$. In this case we parametrize the coordinates via the pathlength $x(s), y(s)$ and

then use for example cubic splines to approximate each one of the curves $x(s)$ and $y(s)$.



A general framework for curve approximation is given by **NURBS** (Non-Uniform Rational B-Splines). NURBS generalize both B-splines and Bézier curves. The key idea is that we do not fit a function to the data but instead compute a linear combination of data points scaled by B-splines and user-specified weights. The resulting curve will generally not pass through the data points but will provide a smooth curve with continuous derivatives and will be efficient to compute.

Given N data points $\boldsymbol{p}_i = \{x_i, y_i\}$ $(i = 1, \ldots, N)$. We now define N B-splines $B_{i,d,t}(s)$ of degree d with knots t_j $(j = 1, \ldots, N + d + 1)$. We furthermore define weights w_i for each data point. The curve is now parametrized as $\boldsymbol{p}(s) = \{x(s), y(s)\}$ for $t_{d+1} \leq s < t_{N+1}$ as follows:

$$\boldsymbol{p}(s) = \sum_{i=1}^{N} R_{i,d,t}(s)\boldsymbol{p}_i, \text{ with } R_{i,d,t}(s) = \frac{B_{i,d,t}(s)\,w_i}{\sum_{j=1}^{N} B_{j,d,t}(s)\,w_j}, \tag{5.4.1}$$

where the B-splines $B_{i,d,t}(s)$ are defined as in Eq. (5.3.1).

Notes:

- If you wish to implement NURBS curves as in Eq. (5.4.1), you would loop for s values in the range $t_{d+1} \leq s \leq t_{N+1}$ and draw the curve as lines connecting subsequent points. Note that you loop over values of s and not over x or anything related to the data points. The loop range depends only on the knot vector.
- By construction, B-splines have the property of partition of unity: $\sum_{i=1}^{N} B_{i,d,t}(s) = 1$ (for $t_{d+1} \leq s \leq t_{N+1}$). For non-weighted NURBS ($w_i = 1$), we therefore have $R_{i,d,t}(s) = B_{i,d,t}(s)$.
- Due to Eq. (5.3.1), any data point i will only affect a bounded region of the parametric coordinate s ($t_i \leq s \leq t_{i+d+1}$). Therefore, changing one data point will only affect a part of the curve.
- Bézier curves (as used in many computer graphics programs) are constructed as unweighted NURBS ($w_i = 1$) by grouping together 4 data points. The resulting curve will be continuous and

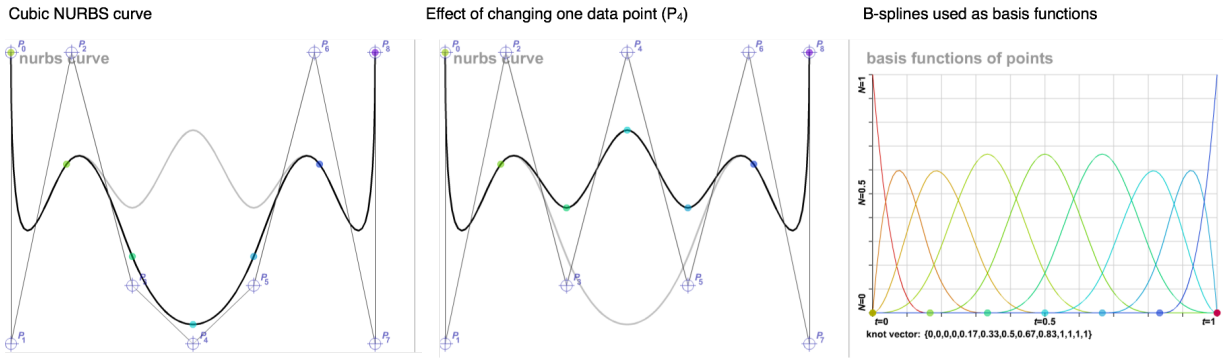


Figure 5.4: A non-weighted NURBS curve with cubic B-splines as basis functions. The left and middle panels show the effect of changing a single data point. The right panel shows the B-splines and knots used. Source: http://geometrie.foretnik.net/files/NURBS-en.swf

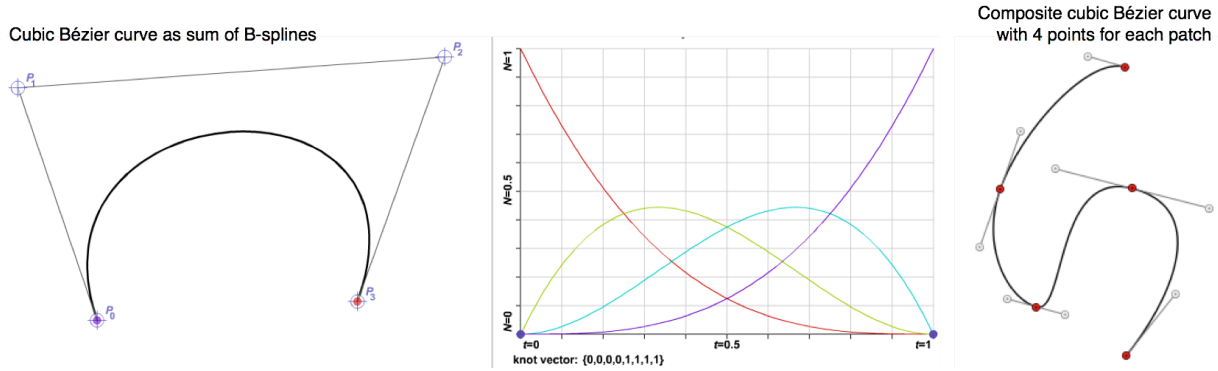


Figure 5.5: Cubic Bézier curve with 4 data points (left) and corresponding B-splines (middle). Longer curves are constructed by patching together such curves (right). Source: left/middle: http://geometrie.foretnik.net/files/NURBS-en.swf, right: Apple Keynote.

will have continuous derivatives if the data points of consecutive groups of data points have the same slope.

- If B-splines with "**clamped**" knots are used (see Eq. (5.3.2)), the curve is guaranteed to start at the first control point and end at the final one.

A demo for NURBS, B-splines and Bézier curves (and the source for the images used here) is given at `http://geometrie.foretnik.net/files/NURBS-en.swf`.

5.5 Multivariate Interpolation

Multivariate interpolation refers to the (most interesting and practical) case of developing interpolating functions for data in higher dimensions.

For simplicity we regard the 2-dimensional case: Given $z_k = Z(x_k, y_k)$ for $k = 1, ..., N$ we must find a reasonable function $f(x, y)$ such that $z_k = f(x_k, y_k)$

5.5.1 Gridded Data

In the case of **gridded data** we can use as functions the tensor products of functions in 1-dimension. So we have that the approximating function reads:

$$f(x, y) = \sum_i \sum_j a_{i,j} \phi_i(x) \phi_j(y) \tag{5.5.1}$$

In general this requires solving a system of equations of dimensions $MN \times MN$ where $M \times N$ are the number of data points in both dimensions. We need to solve for the coefficients $a_{i,j}$ that satisfy:

$$f(x_p, y_q) = Z(x_p, y_q) = \sum_i \sum_j a_{i,j} \phi_i(x_p) \phi_j(y_q) \tag{5.5.2}$$

Using Lagrange polynomials we can show that:

$$f(x_p, y_q) = \sum_i \sum_j Z(x_i, y_j) l_i(x_p) l_j(y_q) \tag{5.5.3}$$

where:

$$l_i(x) = \frac{(x - x_1)(x - x_2) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_M)}{(x_i - x_1)(x_i - x_2) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_M)} \tag{5.5.4}$$

and

$$l_j(y) = \frac{(y - y_1)(y - y_2) \dots (y - y_{j-1})(y - y_{j+1}) \dots (y - y_N)}{(y_j - y_1)(y_j - y_2) \dots (y_j - y_{j-1})(y_j - y_{j+1}) \dots (y_j - y_N)} \tag{5.5.5}$$

Note that the comments we made for Lagrange interpolation in 1D are exacerbated for Lagrange polynomials in 2D.

To approximate a surface given the data points we can use **NURBS surfaces**. We define a "grid" of $N \times M$ control points $\boldsymbol{p}_{i,j} = \{x_{i,j}, y_{i,j}, z_{i,j}\}$ (where the 3D coordinates can be positioned arbitrarily). 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, respectively, and define knot vectors t_U (with $N + d_U + 1$ knots) and t_V (with $M + d_V + 1$ knots), respectively. The surface is finally given as a tensor project of the $R_{i,d,t}$ functions defined in Eq. (5.4.1) as follows:

$$\boldsymbol{p}(u, v) = \sum_{i=1}^{N} \sum_{j=1}^{M} R_{i,d_U,t_U}(u) R_{j,d_V,t_V}(v) \boldsymbol{p}_{i,j} \tag{5.5.6}$$

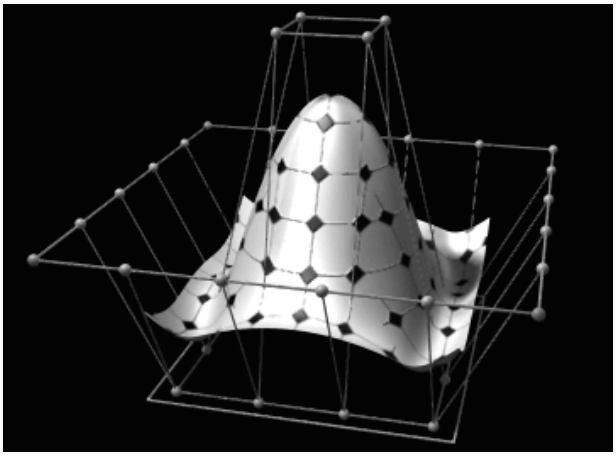


Figure 5.6: A NURBS surface generated with a grid of control points. Source: wikipedia.

5.5.2 Irregular Data

A number of different interpolation methods have emerged over the years that depend strongly on the underlying application and the type of surfaces that the scattered data represent. We outline some representative methods below:

- **Shepard's method:** This has been a popular function to approximate irregular data with applications to computer graphics and in earlier times in simulations using particle based and finite element methods.

  The approximating function for N irregularly spaced points is expressed as

$$f(x,y) = \frac{\sum_{k=1}^{n} z_k\, g(x-x_k, y-y_k)}{\sum_{k=1}^{n} g(x-x_k, y-y_k)} \tag{5.5.7}$$

  where the function $g(x,y)$ is usually selected to be a function with radial symmetry and decaying away from the data points. In Shepard's original approach it was selected that

$$g(x,y) = \frac{1}{(x^2+y^2)^{\mu/2}} \tag{5.5.8}$$

  where μ was a free parameter that was chosen depending on the data. (see article: *Representation and Approximation of Surfaces* by Robert E. Barnhill in: *Rice, John Rischard. 1977. Mathematical Software III: Proceedings of a Symposium Conducted by the Mathematics Research Center, the University of Wisconsin–Madison, March 28-30, 1977. Orlando, FL, USA: Academic Press, Inc.*

- **Coons patch:** This has been the workhorse interpolation method in the 70's and 80's in the automobile industry. We consider surfaces that are composed by four sided patches. These patches can be mapped onto unit squares so the interpolation function can be derived for these squares instead.

Hence we consider a square occupying the area $(0 \le x, y \le 1)$ and that data is given along the edges of the square, that is we know the values at $z(x,0), z(0,y), z(x,1), z(1,y)$.

The method of Coons reads:

$$\begin{aligned} z(x,y) = {} & y \cdot z(x,1) + (1-y) \cdot z(x,0) && \rightarrow \text{linear interpolation in } y \\ & + x \cdot z(1,y) + (1-x) \cdot z(0,y) && \rightarrow \text{linear interpolation in } x \\ & - (1-x) \cdot (1-y) \cdot z(0,0) - (1-x) \cdot y \cdot z(0,1) && \rightarrow \text{bilinear interpolation} \\ & - (1-y) \cdot x \cdot z(1,0) - x \cdot y \cdot z(1,1) && \rightarrow \text{bilinear interpolation} \end{aligned} \tag{5.5.9}$$

Exam checklist

- Method to interpolate data, not to extrapolate data
- Interpolating Splines (e.g. Cubic Splines) are smooth functions which pass through all data points. Cubic splines result in continuous 2nd derivatives (C^2)
- Cubic Splines compared to global Lagrange polynomials: still goes through all data points but with piecewise cubic polynomials (instead of higher order in Lagrange)
- Cubic Splines compared to other piecewise cubic functions: continuous 2nd derivatives
- B-Splines define basis functions to generate splines
- Approximating Splines (e.g. NURBS) are easier to compute and still have continuous 2nd derivatives, but do not pass through all data points
- Multivariate Interpolation is used on functions that depend on multiple coordinates
- NURBS surfaces can be used to generate smooth surfaces for data given on a grid

Exercises

Question 1: Cubic splines with periodicity

How does the matrix equation (Eq. (5.2.6)) look like for periodic functions ($\{x_N, y_N\} = \{x_1, y_N\}$)? Do we have a tri-diagonal matrix to invert?

Solution:
The equations of the interior points will be the same as in Eq. (5.2.6).

$$\frac{\Delta_{i-1}}{6} f''_{i-1} + \frac{\Delta_{i-1} + \Delta_i}{3} f''_i + \frac{\Delta_i}{6} f''_{i+1} = \frac{y_{i+1} - y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}}$$

$$A_i f''_{i-1} + B_i f''_i + C_i f''_{i+1} = D_i$$

For the end points with periodic conditions:

- $i = 1$: $\Delta_{i-1} = \Delta_{N-1}$, $y_{i-1} = y_{N-1}$, $y_1 = y_N$, $f''_{i-1} = f''_{N-1}$

$$\boxed{\frac{\Delta_{N-1}}{6} f''_{N-1}} + \frac{\Delta_{N-1}+\Delta_1}{3} f''_1 + \frac{\Delta_1}{6} f''_2 = \frac{y_2 - y_1}{\Delta_1} - \frac{y_N - y_{N-1}}{\Delta_{N-1}}$$

$$B_1 f''_1 + C_1 f''_2 = D_1$$

- $i = N-1$: $\Delta_{i+1} = \Delta_1$, $y_{i+1} = y_1$, $y_1 = y_N$, $f''_{i+1} = f''_1 = f''_N$

$$\frac{\Delta_{N-2}}{6} f''_{N-2} + \frac{\Delta_{N-2}+\Delta_{N-1}}{3} f''_{N-1} + \boxed{\frac{\Delta_{N-1}}{6} f''_N} = \frac{y_N - y_{N-1}}{\Delta_{N-1}} - \frac{y_{N-1} - y_{N-2}}{\Delta_{N-2}}$$

$$A_{N-1} f''_{N-2} + B_{N-1} f''_{N-1} = D_{N-1}$$

Non-tridiagonal terms are boxed. The above relations are summarized in the following matrix form:

$$\begin{bmatrix} \frac{\Delta_1+\Delta_{N-1}}{3} & \frac{\Delta_1}{6} & 0 & \dots & 0 & \frac{\Delta_{N-1}}{6} \\ \frac{\Delta_1}{6} & \frac{\Delta_1+\Delta_2}{3} & \frac{\Delta_2}{6} & 0 & \dots & 0 \\ 0 & \frac{\Delta_2}{6} & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \ddots & \ddots & \frac{\Delta_{N-2}}{6} \\ \frac{\Delta_{N-1}}{6} & 0 & \dots & 0 & \frac{\Delta_{N-2}}{6} & \frac{\Delta_{N-2}+\Delta_{N-1}}{3} \end{bmatrix} \begin{bmatrix} f''_1 \\ f''_2 \\ \vdots \\ f''_{N-2} \\ f''_{N-1} \end{bmatrix} = \begin{bmatrix} \frac{y_2-y_1}{\Delta_1} - \frac{y_N-y_{N-1}}{\Delta N-1} \\ \vdots \\ \vdots \\ \vdots \\ \frac{y_N-y_{N-1}}{\Delta_{N-1}} - \frac{y_{N-1}-y_{N-2}}{\Delta N-2} \end{bmatrix}$$

The resulting matrix is not tridiagonal.

Question 2: Cubic splines for lines and parabolas

Assume N data points are given as $\{x_i, y_i\}$ $(i = 1, \dots, N)$ with $x_i = i\,h$. Evaluate Eq. (5.2.6) for two special cases:

1. A straight line: $y_i = a + b\,x_i$
2. A parabola: $y_i = a + b\,x_i + c\,x_i^2$

For both cases: simplify Eq. (5.2.6) for the given data (also use $x_{i+1} = x_i + h$ and $x_{i-1} = x_i - h$). What do you expect for the solution f''_i to fit the data perfectly? Is your expected solution an actual solution for your simplified Eq. (5.2.6)?

Solution:
$x_{i+1} = x_i + h$, $x_{i-1} = x_i - h$

- for a straight line:

$$\frac{h}{6} f''_{i-1} + \frac{2h}{3} f''_i + \frac{h}{6} f''_{i+1} = \frac{a + bx_{i+1} - (a + bx_i)}{h} - \frac{a + bx_i - (a + bx_{i-1})}{h}$$

$$= \frac{b(x_i + h - x_i)}{h} - \frac{b(x_i - (x_i - h))}{h} = 0$$

For data following a straight line we expect: $f_i'' = 0$. $y_i' = b$, $y_i'' = 0$. This is a solution of the simplified Eq. (5.2.6).

- for a parabola:

$$\begin{aligned}\frac{h}{6}f_{i-1}'' + \frac{2h}{3}f_i'' + \frac{h}{6}f_{i+1}'' &= \frac{a+bx_{i+1}+cx_{i+1}^2-(a+bx_i+cx_i^2)}{h} - \frac{a+bx_i+cx_i^2-(a+bx_{i-1}+cx_{i-1}^2)}{h} \\ &= \frac{c(x_i+h-x_i)(x_i+h+x_i)}{h} - \frac{c(x_i-x_i+h)(x_i+x_i-h)}{h} \\ &= \frac{ch}{h}[2x_i+h-2x_i+h] = 2ch \\ \frac{1}{6}f_{i-1}'' + \frac{2}{3}f_i'' + \frac{1}{6}f_{i+1}'' &= 2c\end{aligned}$$

For parabola: $y_i' = b + 2cx_i$ and $y_i'' = 2c = const$. Thus, $[\frac{1}{6}+\frac{2}{3}+\frac{1}{6}]2c = 2c$. We find that the expected solution is an actual solution of simplified Eq. (5.2.6).

Question 3: Cubic splines compared to cubic B-splines

Assume N data points are given as $\{x_i, y_i\}$ $(i = 1,\ldots,N)$. We choose N cubic B-splines ($d = 3$) with a knot vector which includes all N x_i values. Since we have N unknowns α_i in Eq. (5.3.3) and N data points we can solve the system of equations with $S_{d,t}(x_i) = y_i$ $(i = 1,\ldots,N)$ to get α_i. What do you expect as a result in comparison with the cubic spline function in Eq. (5.2.2)?

Solution:
The knot vector corresponding to problem data is t_j, $j = 1,\ldots,N+3+1$.

$$S_{d,t}(x_i) = \sum_{i=1}^{N} a_i B_{i,d,t}(x_i) = y_i \quad i = 1,\ldots,N$$

The resulting basis cubic splines (order $d = 3$) are linearly combined in order to construct the final formula using Eq. (5.3.3). This means that a cubic spline can be actually generated using B-splines, by linear combination of basis cubic splines. In fact, one can generate any spline from B-splines.

LECTURE 6

Interpolation and extrapolation III: Radial basis functions

6.1 Orthogonal Functions

In a polynomial fit, the coefficients at each order are intimately connected. Dropping a high-order term gives a completely different function; it is necessary to re-fit to find a lower-order approximation. For many applications, it is convenient to have a model that allows successive terms to be added to improve the agreement without changing the coefficients that have been already found.

Example 6.1

If we store an image in such a way, then the fidelity with which it is displayed can be varied by changing the number of terms used. The fidelity can then be based on the available display, bandwidth and processor.

A classical approach to allow for successive corrections goes via **orthogonal functions**. Orthogonal functions are the functional analog to orthogonal vectors. Using the definition of the inner product on function spaces we can define the notion of orthonormal in exact correspondence

$$\langle \phi_i(x)\phi_j(x)\rangle = \int_{-\infty}^{\infty} \phi_i(x)\phi_j(x)dx = \delta_{ij} \quad . \tag{6.1.1}$$

With the Kronecker delta δ_{ij}. As ϕ_i is assumed to be a basis we can approximate any function using the usual expansion

$$y(x) = \sum_{i=1}^{M} \alpha_i \phi_i(x) \tag{6.1.2}$$

The orthogonality of the basis functions allows us to derive an explicit expression for the coefficients

$$\begin{aligned} \int_{-\infty}^{\infty} y(x)\phi_j(x)dx &= \int_{-\infty}^{\infty} \phi_j(x) \sum_{i=1}^{M} \alpha_i \phi_i(x)dx \\ &= \sum_{i=1}^{M} \alpha_i \int_{-\infty}^{\infty} \phi_i(x)\phi_j(x)dx \\ &= \sum_{i=1}^{M} \alpha_i \delta_{ij} = \alpha_j \end{aligned} \tag{6.1.3}$$

When a set of experimental observations $\{x_i, y_i\}_{i=1,\ldots,N}$ is available instead of a functional form $y(x)$, then it is not possible to directly evaluate the integrals in 6.1.3. We can still use orthonormal functions

but they must be orthonormal with respect of the probability density $p(x)$ of the measurements. This means we now choose the ϕ_i's such that:

$$\langle \phi_i(x)\phi_j(x)\rangle_p = \int_{-\infty}^{\infty} \phi_i(x)\phi_j(x)p(x)dx = \delta_{ij} \tag{6.1.4}$$

In this case we again find a closed form for the coefficients

$$\alpha_i = \langle y(x)\phi_i(x)\rangle = \int_{-\infty}^{\infty} y(x)\phi_i(x)p(x)dx. \tag{6.1.5}$$

Example 6.2

We approximate the probability density $p(x)$ as sums of Dirac delta functions:

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

where Dirac delta functions have the property

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_n)dx = f(x_n).$$

We therefore have

$$\boxed{\alpha_i \approx \frac{1}{N}\sum_{n=1}^{N} y_n\phi_i(x_n)} \tag{6.1.6}$$

Gram-Schmidt orthogonalisation

If we are given a complete set of functions $g_i(x)$ then an orthonormal set of functions $\phi_i(x)$ can be created via the Gram-Schmidt orthogonalization

$$\begin{aligned} \tilde{\phi}_n(x) &= g_n(x) - \sum_{i=1}^{n-1} \phi_i(x) \int_{-\infty}^{\infty} \phi_i(x)g_n(x)\mathrm{d}x \\ \phi_n(x) &= \frac{\tilde{\phi}_n(x)}{\|\tilde{\phi}_n(x)\|} \end{aligned} \tag{6.1.7}$$

Let us make this explicit by performing some steps. We start with an arbitrary vector from our complete set of functions and normalize it

$$\phi_1(x) = \frac{g_1(x)}{[\int g_1(x)g_1(x)dx]^{1/2}}$$

Let us now create a second element which is orthogonal to this first element

$$\tilde{\phi}_2(x) = g_2(x) - \phi_1(x)\int_{-\infty}^{\infty} \phi_1(x)g_2(x)dx$$

We now normalize this element via

$$\phi_2(x) = \frac{\widetilde{\phi}_2(x)}{[\int_{-\infty}^{\infty} \widetilde{\phi_2}(x)\widetilde{\phi_2}(x)dx]^{1/2}}$$

Let us now add another vector to our orthonormal set

$$\widetilde{\phi}_3(x) = g_3 - \phi_1 \int \phi_1 g_3 dx - \phi_2 \int \phi_2 g_3 dx$$

again normalizing

$$\phi_3 = \frac{\widetilde{\phi}_3(x)}{\left[\int \widetilde{\phi}_3 \widetilde{\phi_3} dx\right]^{\frac{1}{2}}}$$

and so on...

We can do the same in the case of experimental data, but replace the scalar product by the one weighted by the distribution underlying our dataset

$$\begin{aligned} \tilde{\phi}_n(x) &= g_n(x) - \sum_{i=1}^{n-1} \phi_i(x) \int_{-\infty}^{\infty} \phi_i(x) g_n(x) p(x) \mathrm{d}x \\ \phi_n(x) &= \frac{\tilde{\phi}_n(x)}{\|\tilde{\phi}_n(x)\|} = \frac{\tilde{\phi}_n(x)}{\sqrt{\int_{-\infty}^{\infty} \tilde{\phi}_i(x)\tilde{\phi}(x) p(x) \mathrm{d}x}} \end{aligned} \tag{6.1.8}$$

In contrast to our previous methods in this case the functions are fit to data by evaluating experimental expectations, rather than doing an explicit search for the fit coefficients.

Example 6.3

Basis functions can be orthogonalised with respect to known distributions. Some well known examples include:

- Hermite polynomials which are obtained by assuming $p(x) \sim e^{-x^2}$
- Laguerre polynomials where we assume $p(x) \sim e^{-x}$
- Chebyshev polynomials which are orthogonal with respect to $p(x) = (1-x^2)^{-1/2})$

6.2 Radial Basis Function

In polynomials the only way to improve the fit is to add more high order terms which eventually diverge even more quickly. High order functions even when passing through the data are useless for interpolation or extrapolation. In particular this is true for functions with discontinuities or sharp peaks. Radial Basis functions (RBF) offer a sensible alternative.

Here we choose a set of identical basis functions ϕ, which depend on the distance from a set of centers c_i and on parameters a_i, which do not necessarily have to enter linearly:

$$y(x) = \sum_{i=1}^{M} \phi(|x - c_i|; a_i) \tag{6.2.1}$$

Advantages:

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

If the centers are fixed and the coefficients enter linearly then we recover the form we are used from the first chapter $y(x) = \sum_{i=1}^{M} a_i \phi(|x - c_i|)$

Issues:

1. **Ambiguity in choice of ϕ**

$$\begin{aligned} \text{linear functions}: &\quad \phi(r) = r \\ \text{power functions}: &\quad \phi(r) = r^n \\ \text{Gaussian functions}: &\quad \phi(r) = e^{-r^2} \qquad \text{and many more...} \end{aligned}$$

   It can be shown that diverging functions have better convergence properties than local ones.

2. **Choice of c_i**: The centers can be chosen
   (a) Random or at preset positions
   (b) Data based, where we further have to destinguish between fixed or moving centers

3. **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.

Exam checklist

Functional representations with basis functions:

- Why/when to use orthogonal functions for interpolations
- How to construct them through the Gram-Schmidt orthogonalisation
- What are radial basis functions

LECTURE 7

Learning from Data: Neural Networks

7.1 Introduction

As we have seen in the previous sections computers are really good in solving task in the case where we can tell them specific rules to execute. Unfortunately many aspects of human thinking and behavior involves very complex or even unknown rules. In this chapter we want to discover neural networks, which are one way to circumvent the need to define such rules. Indeed the so produced machine learning algorithms are very good at recognizing patterns (objects, people, spoken words), recognizing anomalies (unusual behavior in credit card transactions), prediction (Netflix movie suggestions, stock market) that where previously impossible to solve with a computer.

In order to understand the fundamental idea we seek for inspiration in humans. Kids are usually given books with pictures of animals, cars, plants, etc.. By looking at these pictures, the kid learns to classify the objects. In more formal language this means that they gather a lot of examples (data) for which we specify the correct output to a given input. The processing and remembering of this information goes via the human brain. The human brain is made of billions of neurons, where each of these neurons is connected to ten thousands of other neurons. From the point of view of the neuron these connections are devided into dendrites, which are incoming signals and axons, which are outgoing signals (see figure 7.1). The basic functionality is, that the axons start to fire, if the weighted sum of the cumulative signals exceeds some threshold.

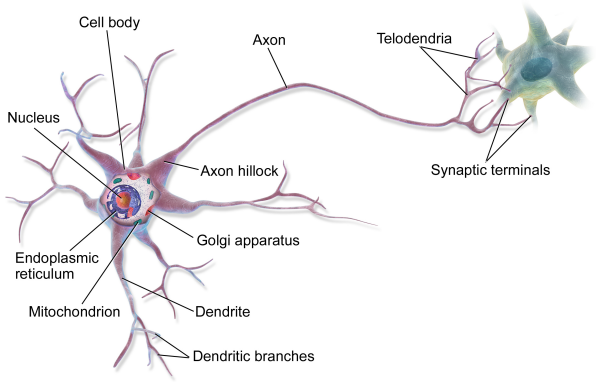


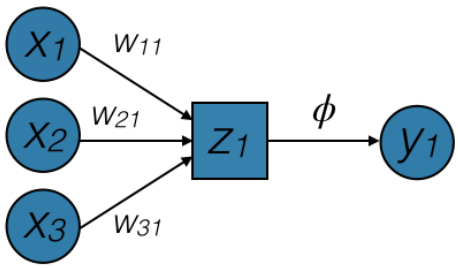
Figure 7.1: Shematic figure of a neuron (source: https://en.wikipedia.org/wiki/Neuron).

In the following we will formalize this and build a virtual brain, namely our neural network. Interest-

ingly it turns out that by doing so we will create universal function approximators, meaning functions that can in principle approximate every other function to arbitrary accuracy.

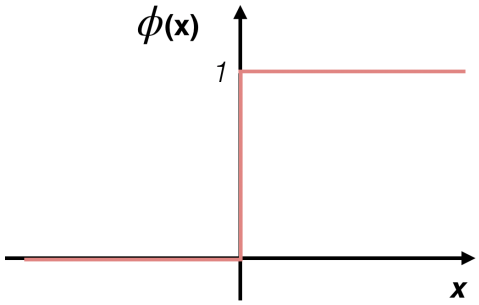
7.2 Neural Networks

Let us redo the above drawing allowing in a bit more ordered fashion. Each neuron consists of incoming signals which we denote by x_i each of these signals has an associated weight W_{ij}. In the neuron a weighted sum z_j is computed and a possibly nonlinear function ϕ computes the output y_j of the neuron. Let us consider the case where $i \in \{1,2,3\}$ and only one output, i.e. $j = 1$ is created

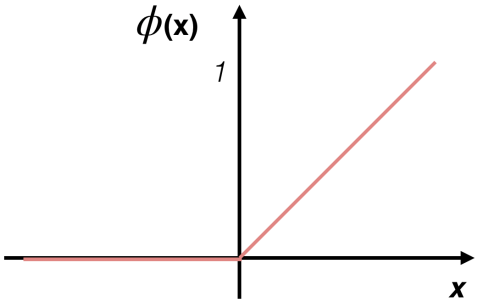


$$y_j = \phi\Big(\underbrace{\sum_{i=1}^{n} W_{ij}x_i + b_j}_{z_j}\Big) \tag{7.2.1}$$

In the original version the Heaviside function was proposed as an activation functions. Later many others joined, where the most prominent ones are listed below:

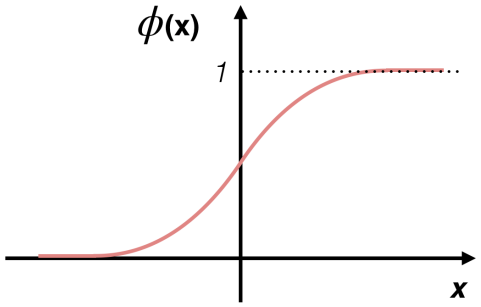


$$\phi(x) = H(x) = \begin{cases} 1, & x > 0 \\ 0, & \text{else} \end{cases} \tag{7.2.2}$$

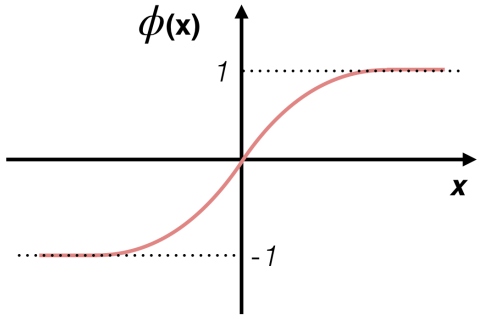


$$\phi(x) = \mathrm{ReLU}(x) = \begin{cases} x, & x > 0 \\ 0, & \text{else} \end{cases} \tag{7.2.3}$$

In modern architectures the prefered choice are smooth functions such as



$$\phi(x) = \sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x} \tag{7.2.4}$$



$$\phi(x) = \tanh(x) \tag{7.2.5}$$

Having this fundamental building blocks we are ready to assemble a **Feedforward Neural Network**, also called **Multilayer Perceptron**. The number of incoming elements is given by n_0. The dimension of the subsequent hidden layers is then denoted by $n_1, \ldots, n_L$. Where L denotes the number of hidden layers. The output computed has a dimension of n_{L+1}. Following the above construction the elements in the first hidden layer is given by

$$h_j^1 = \phi(z_j^1), \quad \text{where} \quad z_j^1 = \sum_{i=1}^{n_0} W_{ij}^1 x_i + b_j^1 \quad j \in \{1, \ldots, n_1\} \tag{7.2.6}$$

This is the input for the second hidden layer, which can be computed as

$$h_j^2 = \phi(z_j^2), \quad \text{where} \quad z_j^2 = \sum_{i=1}^{n_1} W_{ij}^2 h_i^1 + b_j^2 \quad j \in \{1, \ldots, n_2\} \tag{7.2.7}$$

This can now be continued and at the k-th hidden layer we have

$$h_j^k = \phi(z_j^k), \quad \text{where} \quad z_j^k = \sum_{i=1}^{n_{k-1}} W_{ij}^k h_i^{k-1} + b_j^k \quad j \in \{1, \ldots, n_k\} \tag{7.2.8}$$

The output is then computed from the last hidden layer via

$$y_j = \phi(z_j^{L+1}), \quad \text{where} \quad z_j^{L+1} = \sum_{i=1}^{n_L} W_{ij}^{L+1} h_i^L + b_j^{L+1} \quad j \in \{1, \ldots, n_{L+1}\} \tag{7.2.9}$$

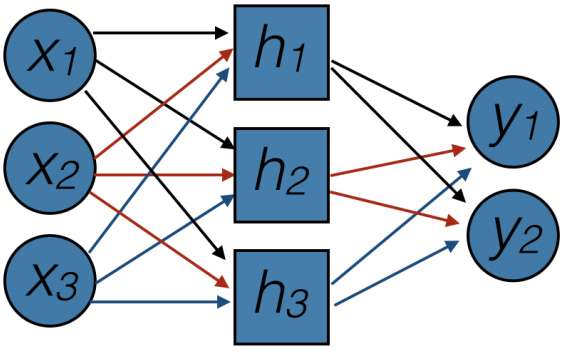


Figure 7.2: Schematic figure of a feedforward neural network with input dimension $n_0 = 3$, $L = 1$ hidden layers with dimension $n_1 = 3$ and output of dimension $n_2 = 2$

In the more compact matrix notation we can write the complete computation from in- to output as

$$\boldsymbol{y} = \phi\left((W^{L+1})^\top \boldsymbol{h}^L + \boldsymbol{b}^{L+1}\right) = \phi\left((W^{L+1})^\top \left(\phi\left((W^L)^\top \boldsymbol{h}^{L-1} + \boldsymbol{b}^L\right)\right) + \boldsymbol{b}^{L+1}\right) = \ldots \tag{7.2.10}$$

Following the above discussion we consider the action of the activation function is to be seen component-wise, i.e. $\sigma(\boldsymbol{x}) = \sum_i \sigma(x_i)\hat{e}_i$ for the standard basis $\hat{e}_i$ and components x_i.

7.3 Architecture

In the above construction we have chosen to take the activation functions to be the same. Generally they can also vary. Already this introduces many possible combinations and therefore a large amount of different architectures. Besides this minor modification there are many other, more elaborate ideas such as convolutional NN or recurrent NN to mention the two used for object recognition and natural language processing (see http://www.asimovinstitute.org/neural-network-zoo/ for an overview). Each of these network architectures has its specific application area and there is no a priori law telling us which one to take.

7.4 Learning

We would like to train the NN on data $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$, where $i = 1, 2, ..., N$ and N is the size of the training data. The cost function E for all the weights $\mathbf{W} = \{W^1, \ldots, W^L\}$ in the network is given by

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} (\boldsymbol{y}_i - \mathrm{NN}(\mathbf{W}, \boldsymbol{x}_i))^2 \tag{7.4.1}$$

Where NN($\mathbf{W}, \boldsymbol{x}$) is the output of our neural network for the given weights $\mathbf{W}$. The cost function can be minimized with the **back-propagation method**, i.e., by computing the derivative of E with respect to the each of the weights W_{ij}^k using the chain rule. Based on the derivatives, we can update our generally unknown weights via

$$\Delta W_{ij}^k = -\eta \frac{\partial E}{\partial W_{ij}^k} \tag{7.4.2}$$

There are many modifications and implementation of this basic idea, e.g., stochastic gradient descent, Adam, RMSProp. All of these methods have in common that it is not a priori clear how one should choose the learning rate. If the learning rate is to slow, the algorithm will need many many iterations to reach the minimum, If in contrast the learning rate is to high, we might fail to reach the minimum and oscillate between suboptimal values. To find the right learning rate is one of the crucial hyper-parameters in deep learning.

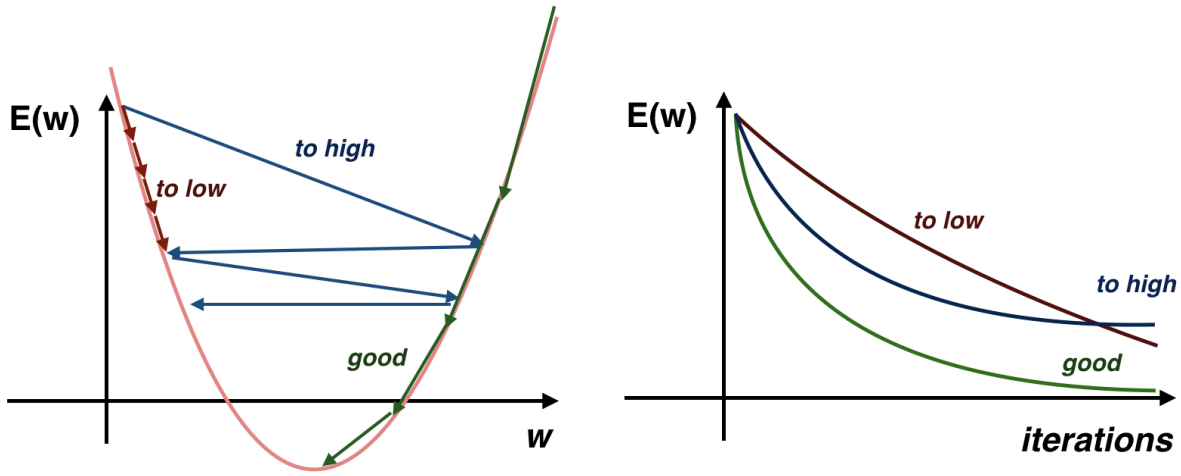


Figure 7.3: The relationship between the error and the number of iterations for different choices of the learning rate.

7.5 Overfitting

Every model with N free parameters should be able to fit exactly N data points. When passing trough all the data points the model is likely to fit behavior of noise and as such it does not generalize efficiently. On the other hand, if too few parameters are used, the model may be forced to ignore meaningful data. Successful fitting requires to find the right balance between underfitting (where model mismatch errors occur) and overfitting (where model estimation errors occur). This is a **bias-variance tradeoff**: A reliable estimate of a biased model or a poor estimate of a model that is capable of a better fit.

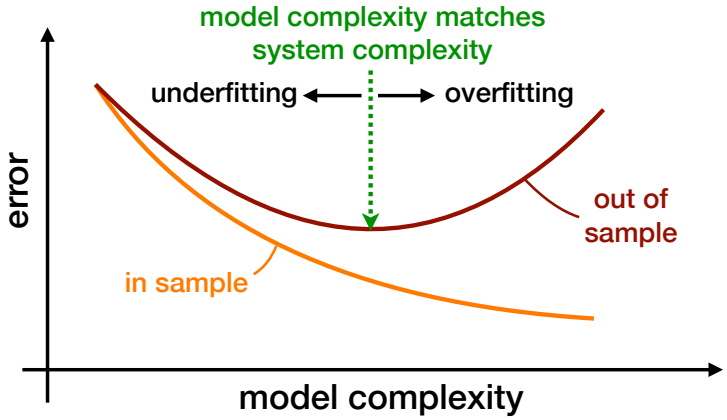


Figure 7.4: Different perspective of overfitting in terms of complexity of the model and the associated errors on the training an the testing datasets. We talk of under-fitting if the model does not allow to capture all the crucial properties of the data. On the otherhand we are overfitting to a specific dataset, if we fit all the data-point perfectly and therefore also fit to artifacts as measurement errors and noise.

Cross-Validation

One way to avoid overfitting is cross-validation. Here we fit the data on one part of the data set but evaluate the model on another part of the data set that you have withheld. When performing gradient descent on the training data with a good learning rate, the error is monotonously decreasing. However, for data that was not used for the training (testing data) the error is a convex function.

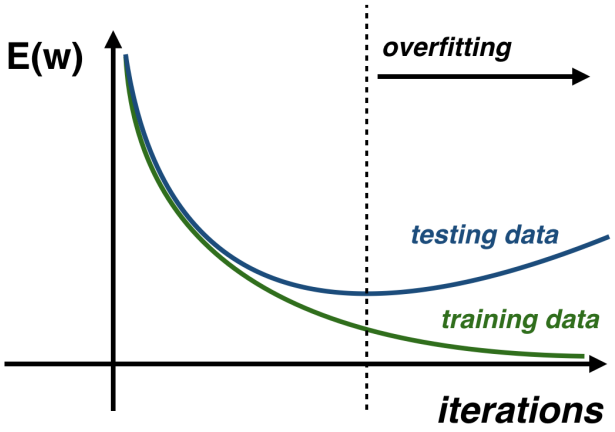


Figure 7.5: The relationship between error on the training set and error on the testing dataset.

The minimum of the testing error defined the regions of under and over-fitting the data. This can be generalized to K-Fold Cross Validation by dividing the training data $Z = \{x_i, y_i; i = 1, \ldots, N\}$ to k disjoint samples of roughly equal size (N/K), $Z_1, Z_2, \ldots, Z_K$. For each validation sample Z_i we use the

remaining 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:

$$r_i = \frac{K}{N} \sum_{Z_i} (f_i(x) - y)^2$$

We can now compute the estimate for the prediction error by averaging the r_i for $Z_1, Z_2, \ldots, Z_k$

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

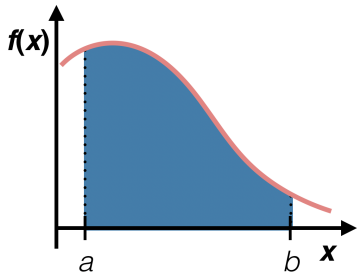
Exam checklist

- Why is the introduced concept called neural network?
- What are typical activation functions?
- How can the values for the weights w_{ij}^k be calculated?
- What is overfitting?
- How can we avoid overfitting and check the generalization properties of our model/network?

LECTURE 8

Numerical integration I: Rectangle, Trapezoidal and Simpson's Rule

We wish to evaluate a definite integral of the function $f(x)$ in the interval $x \in [a,b]$:



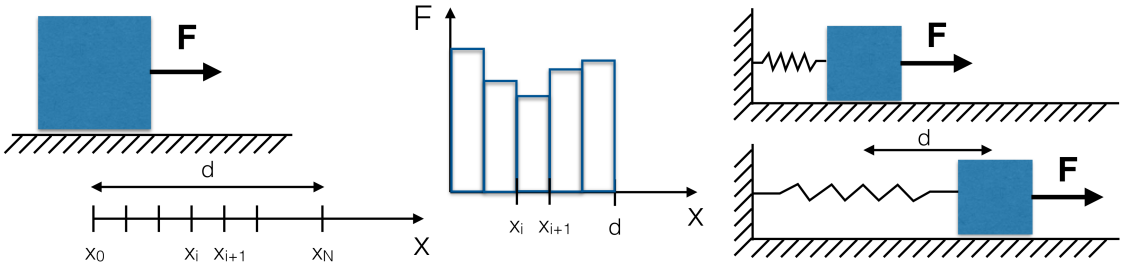
$$I = \int_a^b f(x)\,dx \tag{8.0.1}$$

Numerical integration is crucial in various situations:

- The integral cannot be solved analytically. Example: $I = \int_0^1 \sin(\cos(x))\,\mathrm{d}x$
- Function is only known for a set of *discrete points* as is the case for most experimental data. One can either find a functional form $f(x)$ (e.g. with the methods discussed earlier in this class) and integrate $f(x)$ analytically or one needs to use numerical integration.

Example 8.1: Work

We are pulling a brick on a surface with a force F.



We assume that there is no friction between the brick and the surface. If F is constant and we move the brick by a distance d, then the work we perform is equal to $A = Fd$. However, if F is not constant, i.e., $F = F(x)$ than we can make an approximation and assume that $F(x) = const.$ over a segment Δx (on interval $[x_i, x_{i+1}]$). The work we perform on this interval is $\Delta A_i = F_i \Delta x_i$

and the total work is

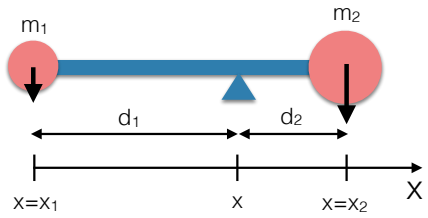
$$A = \sum_{i=0}^{N} \Delta A_i = \sum_{i=0}^{N} F(x_i)\Delta x_i,$$

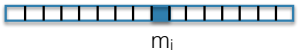
where we have split the distance d into N intervals. If $N \to \infty$ and F is a continuous function we can write the total work as $A = \int_0^d F(x)\,dx$. If the brick is attached to the wall with a linear spring that has a spring constant k, we know that $F = kx$. Thus, the work we perform when we move the brick for a distance d is

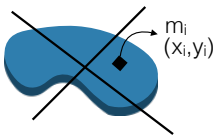
$$A = \int_0^d kx\,dx = \frac{kd^2}{2}.$$

Example 8.2: Moments

Consider a beam that has at left and right ends attached a mass of mass m_1 and m_2, respectively. The question is how to find the position x of the support such that the beam is horizontal.







If we suppose that the beam has no weight, i.e., $m = 0$, then the balance of moments requires that $m_1(\bar{x} - x_1) = m_2(\bar{x} - x_2)$ which gives $\bar{x} = \frac{m_1 x_1 + m_2 x_2}{m_1 + m_2}$. If a beam is not massless, then we can split the beam into small segments and the center-of-mass position of the beam is given by

$$\bar{x} = \frac{\sum_{i=0}^{N} m_i x_i}{\sum_{i=0}^{N} m_i}.$$

Equivalently, we can write for the 2D plate $\bar{x}, \bar{y} = (\sum_{i=0}^{N} m_i(x_i, y_i))/(\sum_{i=0}^{N} m_i)$. Suppose that the density of the beam is not homogeneous, i.e., $\rho_i = m_i/\Delta x_i$ or $m_i = \rho_i \Delta x_i$. Then we can write the above equation as

$$\bar{x} = \frac{\sum_{i=0}^{N} \rho_i(x_i)\Delta x_i x_i}{\sum_{i=0}^{N} \rho_i(x_i)\Delta x_i} \xrightarrow{\text{N}\to\infty} \frac{\int \rho(x)x\,dx}{\int \rho(x)\,dx}.$$

8.1 Key idea

In numerical integration schemes, a common approach is to split the domain $[a,b]$ into N intervals $[x_i, x_{i+1}]$ of length $\Delta_i = x_{i+1} - x_i$. Here, we exploit the property of integrals

$$\int_a^b f(x)\,dx = \int_a^c f(x)\,dx + \int_c^b f(x)\,dx, \tag{8.1.1}$$

i.e., an integral can then be evaluated as a sum of integrals over subdomains. Depending on the application, these points may be given a-priori or they are chosen for a given integration scheme. We may thus rewrite our integral as

$$I = \int_a^b f(x)\,dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)\,dx \tag{8.1.2}$$

In a next step we approximate $f(x)$ within each interval with a simple function $p(x)$ that is easily integrable. The key idea of the numerical integration schemes is thus based on a "divide" and "conquer" approach. Where in step 1 we do a piecewise approximation of $f(x)$ and in step 2 we compute many such integrals exactly.

$$\boxed{I \approx \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} p_i(x)\,dx} \tag{8.1.3}$$

8.2 Numerical Quadrature

Different numerical integration schemes use different approximations to $f(x)$ in each interval $[x_i, x_{i+1}]$.

Rectangle Rule: We approximate $f(x)$ as constant using a single (left) point:

$$\boxed{I_{R_i} = f(x_i)\,\Delta_i} \tag{8.2.1}$$

Midpoint Rule: We approximate $f(x)$ as constant using the middle point:

$$\boxed{I_{M_i} = f(x_{i+1/2})\,\Delta_i = f\left(\frac{x_i + x_{i+1}}{2}\right)\Delta_i} \tag{8.2.2}$$

Trapezoidal Rule: We approximate $f(x)$ by a line (using 2 points):

$$\boxed{I_{T_i} = \frac{f(x_i) + f(x_{i+1})}{2}\Delta_i} \tag{8.2.3}$$

Simpson's Rule: We approximate $f(x)$ by a parabola (using 3 points):

$$I_{S_i} = \frac{f(x_i) + 4f((x_i + x_{i+1})/2) + f(x_{i+1})}{6}\Delta_i \tag{8.2.4}$$

Figure 8.1 shows an example of such approximations.

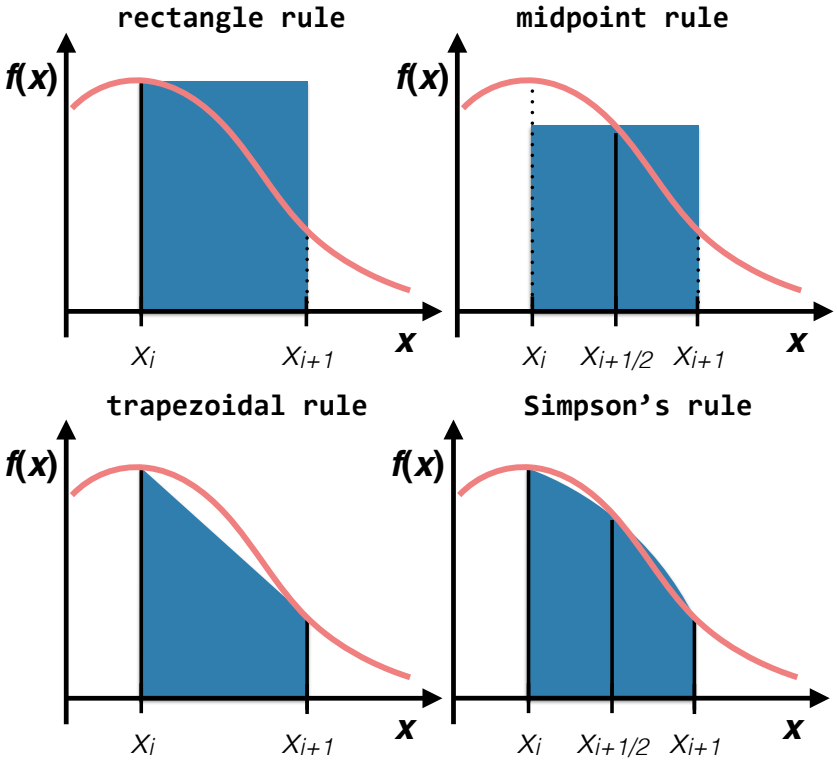


Figure 8.1: Example comparing the rectangle, midpoint, trapezoidal and the Simpson's rule for numerical integration.

Under the assumption of constant spacing $\Delta_i \equiv \Delta_x$ $(\forall i,\ i = 1,\ldots,N)$ we can now compute the total integral

$$\begin{aligned}
&\textbf{Rectangle Rule:} && I \approx \Delta_x \sum_{i=0}^{N-1} f(x_i), \\
&\textbf{Midpoint Rule:} && I \approx \Delta_x \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2}\right), \\
&\textbf{Trapezoidal Rule:} && I \approx \frac{\Delta_x}{2}\left(f(x_0) + 2\sum_{i=1}^{N-1} f(x_i) + f(x_N)\right),
\end{aligned} \tag{8.2.5}$$

For Simpson's approximation, we can rewrite Eq. (8.1.2) to sum over larger intervals $[x_i, x_{i+2}]$. In this way, we are evaluating $f(x)$ only at points x_i. If we assume that the total number of points $N+1$ is

odd we obtain:

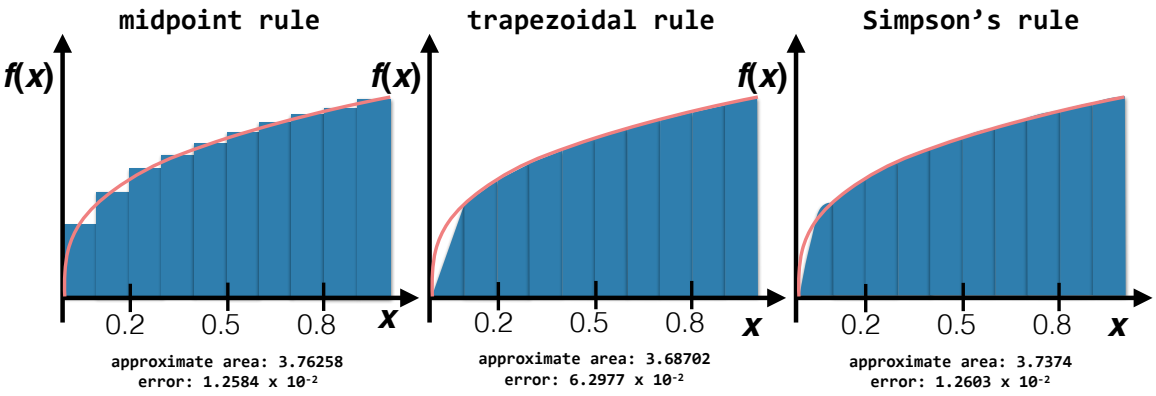
$$I = \int_a^b f(x)\,dx = \sum_{\substack{i=0 \\ i=\text{even}}}^{N-2} \int_{x_i}^{x_{i+2}} f(x)\,dx$$

$$\int_{x_i}^{x_{i+2}} f(x)\,dx \approx \frac{f(x_i) + 4f(x_{i+1}) + f(x_{i+2})}{3}\Delta_x$$

Simpson's Rule:
$$I \approx \frac{\Delta_x}{3}\left(f(x_0) + 4 \sum_{\substack{i=1 \\ i=\text{odd}}}^{N-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=\text{even}}}^{N-2} f(x_i) + f(x_N) \right). \tag{8.2.6}$$

Example 8.3: Composite Integration

We wish to evaluate $I = \int_0^1 5\sqrt[3]{x}\,dx$, i.e., the shaded area in Figure below.



To evaluate the integral we discretize the interval $[0,1]$ using $x_i = 0.1\,i$ $(i = 0,\ldots,10)$, i.e., we have $N = 10$ and $\Delta_x = 0.1$.

We thus see that all of the most fundamental methods write an integral as a weighted sum of f_i:

$$I = \int_a^b f(x)\,dx \approx \sum_{i=0}^{N} w_i f(x_i), \tag{8.2.7}$$

where w_i are the weights. For the trapezoidal rule for instance, we have $w_0 = w_N = \Delta_x/2$ and $w_i = \Delta_x$ for $i = 1,\ldots,N-1$.

8.2.1 Newton–Cotes formulas

A general way to derive quadrature rules is given by the Newton–Cotes formulas. To construct the approximate function $p_i(x)$ in Eq. (8.1.2) we use $M+1$ equidistant points in $[x_i, x_{i+1}]$ $(x_k = x_i + k\,h,$

$k = 0, \ldots, M$) and Lagrange interpolation. The Lagrange interpolant through the points $(x_k, f(x_k))$ is given by

$$\begin{aligned} p_i(x) &= \sum_{k=0}^{M} f(x_k) l_k^M(x), \\ l_k^M(x) &= \frac{(x-x_0)(x-x_1)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_M)}{(x_k-x_0)(x_k-x_1)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_M)}. \end{aligned} \tag{8.2.8}$$

The integral I is then approximated as

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\,dx \approx \int_{x_i}^{x_{i+1}} p_i(x)\,dx = \sum_{k=0}^{M} f(x_k) \int_{x_i}^{x_{i+1}} l_k^M(x)\,dx. \tag{8.2.9}$$

We rewrite this as

$$I_i \approx \Delta_i \sum_{k=0}^{M} C_k^M f(x_k), \text{ where } C_k^M = \frac{1}{\Delta_i} \int_{x_i}^{x_{i+1}} l_k^M(x)\,dx \tag{8.2.10}$$

Properties of C_k^M:

- Lagrange polynomials fit exactly a constant function: e.g. $f(x) = 1$ must be integrated exactly. In Eq. (8.2.10), we hence want to have

  $$I_i = \int_{x_i}^{x_{i+1}} 1\,dx = (x_{i+1} - x_i) \sum_{k=0}^{M} C_k^M 1 \tag{8.2.11}$$

  and it follows that

  $$\sum_{k=0}^{M} C_k^M = 1 \tag{8.2.12}$$

- For $x \in [x_i, x_{i+1}]$ we have that $l_{M-k}^M(x) = l_k^M(x_i + x_{i+1} - x)$. To see this, inserting $x_{M-j} = x_i + (M-j)h$ on the right hand side ($h = \frac{x_{i+1}-x_i}{M}$ and j some arbitrary number). After some algebra we find

  $$l_k^M(x_i + x_{i+1} - x_{M-j}) = \cdots = l_k^M(x_j) = \delta_{jk} = l_{M-k}^M(x_{M-j})$$

  by definition of the Lagrange Polynomial. Since we did the calculation for an arbitrary j the above equality holds. If we now use this on our definition of the coefficient C_k^M and use a substitution of variables we find that

  $$C_k^M = C_{M-k}^M \tag{8.2.13}$$

Example 8.4

Case: M=1

We choose $M = 1$ (i.e. 2 points $x_0 = x_i$, $x_1 = x_{i+1}$). The Lagrange polynomials are then given by

$$l_0^1(x) = \frac{x - x_1}{x_0 - x_1} = \frac{x_{i+1} - x}{x_{i+1} - x_i},\ l_1^1(x) = \frac{x - x_0}{x_1 - x_0} = \frac{x - x_i}{x_{i+1} - x_i}.$$

We integrate this and obtain:

$$C_0^1 = \frac{1}{\Delta_i}\int_{x_i}^{x_{i+1}} l_0^1(x)\,dx = \frac{1}{\Delta_i^2}\int_{x_i}^{x_{i+1}} (x_{i+1}-x)\,dx = \frac{1}{\Delta_i^2}\left(-\frac{1}{2}(x_{i+1}-x)^2\Big|_{x_i}^{x_{i+1}}\right) = \frac{1}{2},$$
$$C_1^1 = \frac{1}{\Delta_i}\int_{x_i}^{x_{i+1}} l_1^1(x)\,dx = \frac{1}{\Delta_i^2}\int_{x_i}^{x_{i+1}} (x-x_i)\,dx = \frac{1}{\Delta_i^2}\left(\frac{1}{2}(x-x_i)^2\Big|_{x_i}^{x_{i+1}}\right) = \frac{1}{2}.$$

If we insert $C_0^1 = C_1^1 = 1/2$ in Eq. (8.2.10), we see that we recover the trapezoidal rule:

$$I_i \approx \Delta_i \sum_{k=0}^{M} C_k^M f(x_k) = \Delta_i \frac{f(x_i)+f(x_{i+1})}{2}.$$

Case: M=2
For $M=2$ (i.e. 3 points $x_0 = x_i$, $x_1 = (x_i + x_{i+1})/2$, $x_2 = x_{i+1}$), we obtain Simpson's rule: $C_0^2 = 1/6$, $C_1^2 = 2/3$, $C_2^2 = 1/6$.

8.2.2 Error analysis

We would like to evaluate the error that with are making when we use different numerical integrations, that is we would like to find an upper bound or an approximation of

$$E_{\text{rule},i} = I_i - I_{\text{rule},i} \tag{8.2.14}$$

We will do so by using Taylor expansions to evaluate the error of the numerical integration schemes.

Midpoint rule: Consider Taylor's series around $x_{i+1/2} = (x_i + x_{i+1})/2$:

$$\begin{aligned} f(x) =& f(x_{i+1/2}) + \left(x - x_{i+1/2}\right) f'(x_{i+1/2}) + \frac{1}{2}\left(x - x_{i+1/2}\right)^2 f''(x_{i+1/2}) \\ &+ \frac{1}{6}\left(x - x_{i+1/2}\right)^3 f'''(x_{i+1/2}) + \cdots, \end{aligned} \tag{8.2.15}$$

We can use this expansion to evaluate I_i:

$$\begin{aligned} I_i &= \int_{x_i}^{x_{i+1}} f(x)\,dx \\ &= f(x_{i+1/2})\int_{x_i}^{x_{i+1}} dx + f'(x_{i+1/2})\int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right) dx \\ &\quad + \frac{1}{2}f''(x_{i+1/2})\int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right)^2 dx + \frac{1}{6}f'''(x_{i+1/2})\int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right)^3 dx + \cdots \\ &= \underbrace{f(x_{i+1/2})\,\Delta_i}_{I_{M_i}} + \underbrace{0 + \frac{1}{24}f''(x_{i+1/2})\Delta_i^3 + 0 + O(\Delta_i^5)}_{E_{M_i}} \end{aligned} \tag{8.2.16}$$

where $O(\Delta_i^5)$ includes all the higher order terms that we dropped from the Taylor series. So for

the midpoint rule we have that:

$$\boxed{E_{M_i} = \frac{1}{24} f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5) + \cdots} \tag{8.2.17}$$

For one interval the midpoint rule is third order accurate.

Trapezoidal rule: One can show that:

$$I_{T_i} = \frac{f(x_i) + f(x_{i+1})}{2}\Delta_i = \Delta_i \left(f(x_{i+1/2}) + \frac{1}{8} f''(x_{i+1/2})\Delta_i^2 + \cdots \right) = I_{M_i} + \frac{1}{8} f''(x_{i+1/2})\Delta_i^3 + \cdots$$

If we compare this with Eq. (8.2.17), we see that

$$\boxed{E_{T_i} = -\frac{1}{12} f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5) + \cdots} \tag{8.2.18}$$

Simpson's rule: We can combine the midpoint and the trapezoidal rule. By comparing Eq. (8.2.4) with Eq. (8.2.2) and Eq. (8.2.3), we see that:

$$I_{S_i} = \frac{2}{3} I_{M_i} + \frac{1}{3} I_{T_i} \tag{8.2.19}$$

therefore

$$\boxed{E_{S_i} = O(\Delta_i^5) + \cdots}. \tag{8.2.20}$$

We remark that the above error estimates are only valid for the local approximation. When considering the whole domain (with a constant spacing $\Delta_x = \Delta_i$ $(\forall i)$) the error is reduced to second order for the midpoint and trapezoidal rule and fourth order for Simpson's rule. For the midpoint rule for instance we get (using Eq. (8.1.2)):

$$\begin{aligned} \sum_{i=0}^{N-1} I_{M_i} &= \sum_{i=0}^{N-1} \left(I_i - \frac{1}{24} f''(x_{i+1/2})\Delta_x^3 + O(\Delta_x^5) + \cdots \right), \\ \left| \sum_{i=0}^{N-1} I_{M_i} - I \right| &< N \frac{1}{24} \max_i(|f''(x_{i+1/2})|)\Delta_x^3 + NO(\Delta_x^5) + \cdots, \\ &= \frac{b-a}{24} \max_i(|f''(x_{i+1/2})|)\Delta_x^2 + O(\Delta_x^4) + \cdots, \end{aligned} \tag{8.2.21}$$

where we used that $N = (b-a)/\Delta_x$.

Exam checklist

After this class, you should understand the following points regarding numerical integration:

- When to use numerical integration.
- How to do numerical integration with the rectangle, midpoint, trapezoidal and Simpson's rule.

- How to estimate errors of a numerical integration scheme.

LECTURE 9

Numerical integration II: Richardson and Romberg

What are requirements for a good numerical integrator?

1. Ease of use: User specifies the function to be integrated, integration bounds and desired accuracy. The code should produce the required result.

2. Cost: minimal time to solution (or alternatively: minimal energy, etc.)

In principle, any of the Newton–Cotes formulas can produce the desired result by halving the subintervals repeatedly until convergence. Such a procedure may be problematic for complex functions with large peaks and valleys.

A better approach is the **Romberg integration** that is based on the concept of **Richardson extrapolation** (also called "deferred approach to the limit"). The concept of Richardson extrapolation is a simple and elegant way to extend the accuracy by which we compute numerically *any quantity* (not necessarily an integral).

9.1 Richardson Extrapolation

If we work on a computer we have to discretize our quantity of interest G. This approximations depend on the chosen discretization, which is normally characterized by some grid-spacing h so that we write

$$G \approx G(h) \tag{9.1.1}$$

In other word, if we calculate any quantity on the computer, the result depends on our choice of the grid via the spacing h. As this grid spacing is usually small $h \ll 1$ and we want to have a consistent discretization (i.e. one for which $G(h) \xrightarrow{h\to 0} G$) we can expand our approximation in terms of a Taylor series for h:

$$G(h) = G(0) + c_1 h + c_2 h^2 + \cdots, \tag{9.1.2}$$

where $c_1, c_2, \ldots$ are the typical constants we obtain from this expansion $G'(0), G''(0)/2, \ldots$ (some constants may be 0). As already remarked above, the term $G(0)$ is the exact value ($G = G(0)$), while the rest of the terms are the errors we wish to eliminate. As said at the beginning one way to go about that is to splitt h into $h/2$ so that

$$G(h/2) = G + \frac{1}{2}c_1 h + \frac{1}{4}c_2 h^2 + \cdots. \tag{9.1.3}$$

We have now doubled the operations (assuming that we compute $G(h)$ in $O(1/h)$) and halved the error. Richardson's ingenious idea is to combine Eq. (9.1.2) and Eq. (9.1.3) to reduce the error:

$$G_1(h) = 2\,G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \cdots, \tag{9.1.4}$$

where c_2', c_3' are fractions of c_2, c_3. Now for $G_1(h)$ the leading error term is h^2. So as $h \to 0$, $G_1(h)$ is much more accurate than $G(h)$ and $G(h/2)$ at very little extra cost (one subtraction). We can repeat this process to obtain

$$G_2(h) = \frac{1}{3}\left(4\,G_1(h/2) - G_1(h)\right) = G + O(h^3), \tag{9.1.5}$$

which is even more accurate. Finally, we define $G_0(h) = G(h)$ and obtain

$$\boxed{G_n(h) = \frac{1}{2^n - 1}\left(2^n\,G_{n-1}(h/2) - G_{n-1}(h)\right) = G + O(h^{n+1})} \tag{9.1.6}$$

Example 9.1

Compute 2π by measuring the perimeter of a regular polygon inscribed in the unit circle. A polygon with E edges in the unit circle will have an edge length of $2\sin(\pi/E)$. We can use a Taylor expansion of $\sin(\pi/E)$ around $\sin(0)$ and obtain:

$$P_0(E) = 2E\sin(\pi/n) = 2E\left((\pi/E) - (\pi/E)^3/6 + O(1/E^5)\right) = 2\pi + O(1/E^2).$$

Since the error contains only even terms $(O(1/E^2), O(1/E^4), \ldots)$, we can adapt Eq. (9.1.6) and write

$$P_n(E) = \frac{1}{4^n - 1}\left(4^n\,P_{n-1}(2E) - P_{n-1}(E)\right) = 2\pi + O(1/E^{2n+2}).$$

9.2 Error Estimation

The same idea can be used to estimate the error of an approximation. We have

$$G(h/2) - G(h) = -\frac{1}{2}c_1 h - \frac{3}{4}c_2 h^2 + O(h^3). \tag{9.2.1}$$

On the other hand, we can rearrange Eq. (9.1.3) to get

$$\epsilon(h/2) = G - G(h/2) = -\frac{1}{2}c_1 h - \frac{1}{4}c_2 h^2 + O(h^3). \tag{9.2.2}$$

So to first order (i.e. dropping $O(h^2)$ terms), the error of the approximation with $h/2$ can be estimated as

$$\boxed{\epsilon(h/2) \approx G(h/2) - G(h)} \tag{9.2.3}$$

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.

9.3 Romberg integration

The key idea is to take inaccurate integration methods and improve them by using Richardson's extrapolation. We start with the trapezoidal rule with a single interval. Then recalculate with two intervals, four, eight, and so on with the results:

$$I_0^1, I_0^2, I_0^4, \cdots \tag{9.3.1}$$

In the calculation of I_0^n for n intervals, half of the needed functions have been computed earlier.

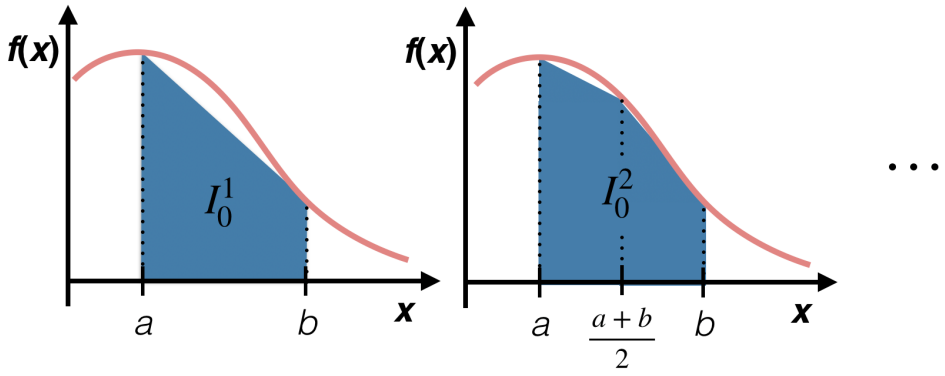


Figure 9.1: Graphical representation of the refinement step on the example of the trapezoidal rule.

From our numerical analysis of the error of the trapezoidal rule we have

$$I = \int_a^b f(x)dx = \underbrace{\frac{h}{2}\left[f(a)+f(b)+2\sum_{j=1}^{n-1} f_j\right]}_{I_0^n} + c_1 h^2 + c_2 h^4 + c_3 h^6 + \cdots, \tag{9.3.2}$$

where we have equidistant intervals of length $h = (b-a)/n$ with $f_j = f(a+j\,h)$. Then the exact I and I_0^n are related as

$$I_0^n = I - c_1 h^2 - c_2 h^4 - c_3 h^6 \tag{9.3.3}$$

Now we evaluate with half the grid size $h_1 = h/2$:

$$I_0^{2n} = I - c_1 \frac{h^2}{4} - c_2 \frac{h^4}{16} - c_3 \frac{h^6}{64} \cdots \tag{9.3.4}$$

We eliminate the $O(h^2)$ term by using Richardson extrapolation:

$$(\star)\ I_1^n = \frac{4I_0^{2n} - I_0^n}{3} = I + \frac{1}{4}c_2 h^4 + \frac{5}{16}c_3 h^6 \tag{9.3.5}$$

This is a fourth order approximation for I. In fact we rediscovered Simpson's rule. For $h_2 = h_1/2 = h/4$ we get

$$I_0^{4n} = I - c_1 \frac{h^2}{16} - c_2 \frac{h^4}{256} - c_3 \frac{h^6}{4096} - \cdots \tag{9.3.6}$$

To get another fourth order estimate we combine I_0^{2n} and I_0^{4n}

$$(\star\star)\ I_1^{2n} = \frac{4I_0^{4n} - I_0^{2n}}{3} = I + \frac{1}{64}c_2h^4 + \frac{5}{1024}c_3h^6 \tag{9.3.7}$$

So now we can combine I_1^n and I_1^{2n} and eliminate the $O(h^4)$ terms and get a 6th order accurate

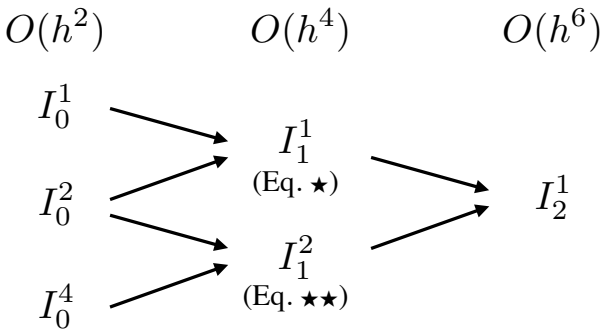


Figure 9.2: Graphical representation of Romberg integration.

formula. The process may be continued to get

$$I_k^n = \frac{4^k I_{k-1}^{2n} - I_{k-1}^n}{4^k - 1} \tag{9.3.8}$$

In principle, arbitrary accuracy can be achieved.

Exam checklist

After this class, you should understand the following points regarding numerical integration:

- How to use Richardson extrapolation to improve the accuracy of a numerically estimated quantity and to estimate errors
- How to use Romberg integration to improve the accuracy of trapezoidal rule

LECTURE 10 Numerical integration III: Adaptive Quadrature

While Romberg integration can achieve arbitrary accuracy, it is not the most efficient method in terms of function evaluations. One of the reasons Romberg integration requires many evaluations in some cases (e.g. functions with sharp peaks) is that the intervals are equispaced. Many evaluations take place in areas that are not "important" (i.e. in areas where the function value is small or slowly varying). Intuitively we must use fewer points where the function varies slowly and more at places with strong variations.

10.1 Adaptive Integration

We want to increase the accuracy where needed and stop refining in the other regions. Thus we have to identify the regions where we need refinement. Thinking of the methods learned in the last lecture we realize, that we already derived the formal tool to estimate the error for a given quadrature rule when we talked about Richardson extrapolation. We then found

$$\epsilon(h/2) \approx G(h/2) - G(h) \tag{10.1.1}$$

Using this we can thus define an adaptive integration procedure as follows:

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
```

10.2 Gauss Quadrature

In the above spirit, namely that certain points and regions are more important when evaluating the integral we expect to obtain higher accuracy by adapting the weights and abscissas of our quadrature rules. We want to choose them such that they maximize the accuracy of the formulas. As we will see in the following this can be cast into the problem of solving a non-linear set of equations.

10.2.1 Method of undetermined coefficients

As a warm up, let us re-derive the trapezoidal rule using the method of undetermined coefficients. We start by approximating

$$\int_a^b f(x)dx \approx c_1 f(a) + c_2 f(b). \tag{10.2.1}$$

Let the right hand side of (10.2.1) be exact for integrals of a straight line, e.g.

$$\int_a^b f(x)dx = \int_a^b (a_0 + a_1 x)dx = \left[a_0 x + a_1 \frac{x^2}{2} \right]_a^b = a_0(b-a) + a_1 \left(\frac{b^2 - a^2}{2} \right). \tag{10.2.2}$$

In particular, for $f(x) = a_0 + a_1 x$, we want right hand sides of (10.2.1) and (10.2.2) to be equal,

$$a_0(b-a) + a_1 \left(\frac{b^2 - a^2}{2} \right) = c_1(a_0 + a_1 a) + c_2(a_0 + a_1 b), \tag{10.2.3}$$

which can be rewritten as

$$a_0(b-a) + a_1 \left(\frac{b^2 - a^2}{2} \right) = a_0(c_1 + c_2) + a_1(c_1 a + c_2 b). \tag{10.2.4}$$

Since the above equation needs to be satisfied for arbitrary values of constants a_0 and a_1 for a general straight line, we require

$$c_1 + c_2 = b - a, \qquad c_1 a + c_2 b = \frac{b^2 - a^2}{2}. \tag{10.2.5}$$

The above quadratic system of equations can be easily solved, obtaining

$$c_1 = c_2 = \frac{b-a}{2}, \tag{10.2.6}$$

which recovers the trapezoidal rule.

Derivation of two-point Gauss rule

The two-point Gauss quadrature is an extension of the trapezoidal rule approximation, where the arguments of the function are not predetermined as a and b, but considered as unknowns x_1 and x_2. Henceforth, in the two-point Gauss quadrature rule, the integral is approximated as

$$I = \int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2). \tag{10.2.7}$$

There are four unknowns x_1, x_2, c_1, and c_2. These are found by requiring that the rule (10.2.7) gives exact results for integration of a general third order polynomial, $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$[1]. Hence

$$\begin{aligned}\int_a^b f(x)dx &= \int_a^b (a_0 + a_1 x + a_2 x^2 + a_3 x^3)dx = \left[a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + a_3 \frac{x^4}{4} \right]_a^b \\ &= a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) + a_2\left(\frac{b^3-a^3}{3}\right) + a_3\left(\frac{b^4-a^4}{4}\right).\end{aligned} \tag{10.2.8}$$

Alternatively, applying quadrature rule (10.2.7), we obtain

$$\int_a^b f(x)dx \approx c_1 f(x_1) + c_2 f(x_2) = c_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + c_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3). \tag{10.2.9}$$

Equating right hand sides of (10.2.8) and (10.2.9), we have

$$\begin{aligned}&a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) + a_2\left(\frac{b^3-a^3}{3}\right) + a_3\left(\frac{b^4-a^4}{4}\right) \\ &= c_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + c_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3) \\ &= a_0(c_1 + c_2) + a_1(c_1 x_1 + c_2 x_2) + a_2(c_1 x_1^2 + c_2 x_2^2) + a_3(c_1 x_1^3 + c_2 x_2^3).\end{aligned} \tag{10.2.10}$$

Since in (10.2.10), the constants a_0, a_1, a_2, and a_3 are arbitrary, the respective coefficients must be equal as well, resulting in the following cubic system equations for c_1, c_2, x_1, and x_2:

$$\begin{aligned}b - a &= c_1 + c_2, \\ \frac{b^2-a^2}{2} &= c_1 x_1 + c_2 x_2, \\ \frac{b^3-a^3}{3} &= c_1 x_1^2 + c_2 x_2^2, \\ \frac{b^4-a^4}{4} &= c_1 x_1^3 + c_2 x_2^3.\end{aligned} \tag{10.2.11}$$

Without proof, we can find the above four simultaneous nonlinear equations have only one acceptable solution

$$\begin{aligned}c_1 &= \frac{b-a}{2}, \\ c_2 &= \frac{b-a}{2}, \\ x_1 &= \left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}, \\ x_2 &= \left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}.\end{aligned} \tag{10.2.12}$$

[1] As we now consider the abscissa to be unknowns we obtain two additional degrees of freedom and thus we can consider a cubic polynomial instead of the linear one from before

Solution values for c_1, c_2, x_1, and x_2 as in (10.2.12), inserted into (10.2.7), lead to the **two-point Gauss quadrature rule**:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} f\left[\left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right)+\frac{b+a}{2}\right]+\frac{b-a}{2} f\left[\left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right)+\frac{b+a}{2}\right]. \tag{10.2.13}$$

Higher order Gauss quadrature rules and their properties are better understood in terms of Hermite interpolation and Legendre polynomials and their properties.

10.2.2 Hermite Interpolation

In order to derive n-point Gauss quadrature rules we have to introduce a new case of interpolation: Hermite interpolation. This is an interpolation that allows for extra degrees of smoothness than Lagrange interpolation. Recall for Lagrange interpolation we have that:

- Lagrange interpolants tend to oscillate about the exact function.
- Smooth functions are interpolated more accurately than the ones that oscillate or have concentrated curvature.
- Decrease in error as the number of points and degree of polynomials increase depends strangely on the function nature.
- Extrapolation yields much larger errors than interpolation.

Returning to the requirement of enhanced smoothness, one way to interpolate a function is not only to interpolate its function values but also to interpolate its derivatives.

Given y_i, y_i' at $i = 1,\ldots,n$ data points $x_1,\ldots,x_n$ find a polynomial $f(x)$ of degree $2n-1$ that fits all the data (i.e. $y_i = f(x_i)$ and $y_i' = f'(x_i)$):

$$\boxed{f(x) = \sum_{k=1}^{n} U_k(x) y_k + \sum_{k=1}^{n} V_k(x) y_k',} \tag{10.2.14}$$

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

$$U_k(x_j) = \delta_{jk}, \qquad U_k'(x_j) = 0, \tag{10.2.15}$$

$$V_k(x_j) = 0, \qquad V_k'(x_j) = \delta_{jk}. \tag{10.2.16}$$

The required polynomials can be constructed using properties of the Lagrange polynomials $L_k(x)$:

$$L_k(x) = \frac{(x-x_1)(x-x_2)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_n)}{(x_k-x_1)(x_k-x_2)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_n)} \tag{10.2.17}$$

The resulting U_k and V_k can then be expressed in terms of $L_k(x)$ as follows:

$$U_k(x) = \left[1 - 2L_k'(x_k)(x - x_k)\right] L_k^2(x), \tag{10.2.18}$$

$$V_k(x) = (x - x_k) L_k^2(x). \tag{10.2.19}$$

By the properties of the Lagrange polynomials we can check that the so constructed functions satisfy the wished properties. The polynomials are smoother and more accurate but suffer from some of the pathologies of Lagrange polynomials.

10.2.3 n-point Gauss rule

As is conventional, we will move from our general integration interval $I = (a, b)$ to $I' = (-1, 1)$. In order make use of the found weights and abscissas we have to perform a transformation of variables

$$\boxed{\xi = \frac{2x - (a + b)}{b - a}} \tag{10.2.20}$$

So any x in (a, b) is transformed to an integral in ξ on $(-1, 1)$.

Let us now use the found interpolation to obtain $\int_{-1}^{1} f(x)\mathrm{d}x$. As done previously we approximate f by Hermite polynomials (Eq. (10.2.14))

$$\int_{-1}^{1} f(x)dx = \sum_{k=1}^{n} y_k \int_{-1}^{1} U_k(x)dx + \sum_{k=1}^{n} y_k' \int_{-1}^{1} V_k(x)dx, \tag{10.2.21}$$

Which can be rewritten as follows:

$$\int_{-1}^{1} f(x)dx = \sum_{k=1}^{n} u_k f(x_k) + \sum_{k=1}^{n} v_k f'(x_k), \tag{10.2.22}$$

where:

$$u_k = \int_{-1}^{1} U_k(x)dx, \qquad v_k = \int_{-1}^{1} V_k(x)dx. \tag{10.2.23}$$

In order for equation 10.2.22 to be of the form $I = \int_{-1}^{1} f(x)\mathrm{d}x = \sum_{i=1}^{n} w_i f(x_i)$ we must make $v_k = 0$ for all k. This can be assured by choosing accordingly the abscissas. Inserting the expression for $V_k(x)$ we find

$$v_k = \int_{-1}^{1} (x - x_k) L_k^2(x)dx. \tag{10.2.24}$$

Now recall that we can decompose the Lagrange polynomials as $L_k(x) = C_k F(x)/(x - x_k)$ with

$$C_k = \frac{1}{(x_k - x_1)(x_k - x_2)\cdot\ldots\cdot(x_k - x_{k-1})(x_k - x_{k+1})\cdot\ldots\cdot(x_k - x_n)}, \tag{10.2.25}$$

$$F(x) = (x - x_1)(x - x_2)\cdot\ldots\cdot(x - x_{n-1})(x - x_n). \tag{10.2.26}$$

Inserting this factorisation for one of the L_k's we find

$$v_k = C_k \int_{-1}^{1} F(x) L_k(x)\, dx. \tag{10.2.27}$$

As C_k is non-zero we have to make sure that the following integral if it is zero for all k

$$\boxed{0 = \int_{-1}^{1} F(x) L_k(x)\, dx} \tag{10.2.28}$$

We know that $F(x)$ is a polynomial of degree n, where $L_k(x)$ is a polynomial of degree $n-1$. If the abscissas x_k are all different, the Lagrange polynomials form a linearly independent set. Hence Eq. (10.2.28) says that $F(x)$ is a polynomial of degree n that is orthogonal to any polynomial of degree $n-1$. This polynomial is unique and is well known to be the Legendre polynomial of degree n, $P_n(x)$. The Legendre polynomials have this orthogonality property:

$$\int_{-1}^{1} P_n(x) P_m(x)\, dx = N_n \delta_{nm}, \tag{10.2.29}$$

where $N_n = 2/(2n+1)$ is a normalization constant. So this analysis shows that the abscissas we try to find are the zeros of the Legendre polynomial of degree n; it is possible to show that the zeros lie between -1 and $+1$.

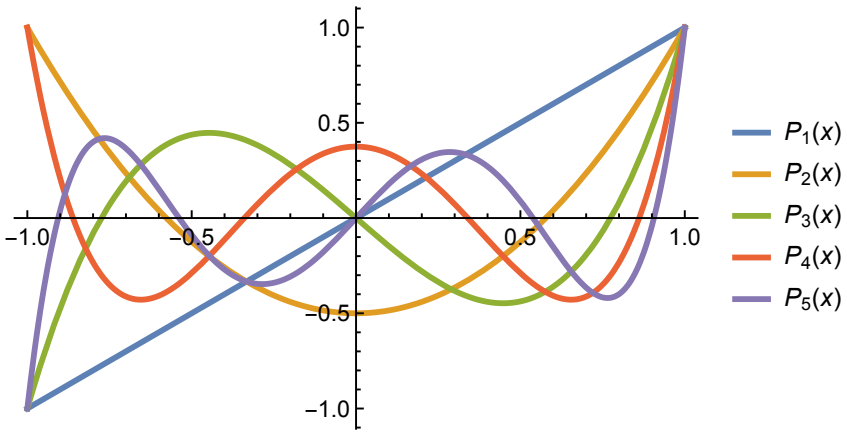


Figure 10.1: Legendre polynomials $P_n(x)$ for $n = 1, \ldots, 5$.

The weights can be computed from Eq. (10.2.23) for the given abscissa so we have that

$$\boxed{\int_{-1}^{1} f(x)\, dx = \sum_{k=1}^{n} u_k f(x_k)} \tag{10.2.30}$$

We compute x_k as the k-th root of $P_n(x)$ and one can show that the weights in Eq. (10.2.30) are given by (according to *Handbook of Mathematical Functions* by Abramowitz and Stegun):

$$u_k = \frac{2}{\left(1 - x_k^2\right) \left(P_n'(x_k)\right)^2}. \tag{10.2.31}$$

The values of x_k and the respective weights for $n = 8$ can be found in the following table

Points x_k	-0.96029	-0.796666	-0.525532	-0.183435	0.183435	0.525532	0.796666	0.96029
Weights u_k	0.101229	0.222381	0.313707	0.362684	0.362684	0.313707	0.222381	0.101229

Error The error with n abscissas is:

$$\varepsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\xi). \tag{10.2.32}$$

Gauss Quadrature gives the best accuracy (in the sense of correctly integrating polynomials of highest possible order for a given number of function evaluations). Abscissas for various orders are all different. If one wishes to improve the accuracy, new calculations must be done from scratch. Hence for some cases Romberg or adaptive integration are still "better".

Exam checklist

After this class, you should understand the following points regarding numerical integration:

- How to improve accuracy of integration locally with adaptive quadrature
- How to choose the locations of function evaluation optimally with Gauss quadrature

LECTURE 11 Numerical integration IV: Monte Carlo

In this chapter, we discuss another method where the locations of the data points (abscissas) are not uniform. The motivation to introduce another method, which is different from the already "optimal" Gauss quadrature will become clear when going to high dimensional integrals. An example of a high-dimensional integration is, for instance, a statistical mechanics model with N particles, where $6N$-dimensional integrals ($3N$ positions and $3N$ momenta) need to be estimated. As an example considering that a liter of water contains $N \sim 10^{26}$ water molecules, these integrals are very high dimensional. As we will see in the following section for such integrals we run into problem. Luckily we can resolve these problems by means of Monte Carlo integration.

11.1 Curse of dimensionality

For a given multi-variate function depending on $d \in \mathbb{N}$ variables,

$$f : \mathbb{R}^d \to \mathbb{R}, \qquad \boldsymbol{x} = (x^{(1)}, \dots, x^{(d)}) \mapsto f(x^{(1)}, \dots, x^{(d)}), \tag{11.1.1}$$

we are interested in estimating its integral over a multi-dimensional domain $\Omega = \Omega_1 \times \cdots \times \Omega_d$, where $\Omega_r = [a_r, b_r]$ for $r = 1, \dots, d$

$$I = \int_\Omega f(x^{(1)}, \dots, x^{(d)}) \mathrm{d}\boldsymbol{x} \tag{11.1.2}$$

By Fubinis theorem one way to estimate I would be to split it into d nested integrals

$$I = \int_{\Omega_1} \cdots \int_{\Omega_d} f(x^{(1)}, \dots, x^{(d)}) \mathrm{d}x^{(d)} \cdots \mathrm{d}x^{(1)}, \tag{11.1.3}$$

and to apply any of the 1-dimensional quadrature rules with n points from the previous sections along each dimension $r = 1, \dots, d$. In particular, given a one dimensional quadrature rule with locations $x_1, \dots, x_n$ and weights $w_1, \dots, w_n$, the d-dimensional quadrature rule obtained using this Cartesian product is given by

$$I \approx \sum_{\substack{i_1=1 \\ \cdots \\ i_d=1}}^{n} \tilde{w}_{i_1,\dots,i_d} f(x^{(1)}_{i_1}, \dots, x^{(d)}_{i_d}), \quad \text{with} \quad \tilde{w}_{i_1,\dots,i_d} = \prod_{r=1}^{d} w_{i_r}. \tag{11.1.4}$$

Note that in such case, the total number of function evaluations is $M = n^d$

The **curse of the dimensionality** becomes evident once we look into the accuracy of such multi-dimensional quadrature rule. For instance, we know that one-dimensional Simpson's rule is forth order accurate, i.e.

$$I - I_S = \mathcal{O}(h^4). \tag{11.1.5}$$

However, unlike in one-dimensional setting, where interval size h is inversely proportional to the total number of quadrature points M (since for $d = 1$ we have $M = n$). i.e.

$$h = \frac{b_1 - a_1}{n} = \mathcal{O}(n^{-1}) = \mathcal{O}(M^{-1}), \tag{11.1.6}$$

in the multi-dimensional ($d > 1$) case we have an exponential dependence with the exponent d,

$$h = \frac{(b_1 - a_1)}{n} = \cdots = \frac{(b_d - a_d)}{n} = \mathcal{O}(n^{-1}) = \mathcal{O}(M^{-1/d}). \tag{11.1.7}$$

Taking this into account, the order of accuracy with respect to the number of function evaluations is no longer 4, but $4/d$,

$$I - I_S = \mathcal{O}(M^{-4/d}). \tag{11.1.8}$$

In general, an order s scheme is order s/d in d dimensions. For large dimensions d, the order of accuracy is hence significantly reduced and the required computational cost $M = n^d$ could be unfeasible.

11.2 Probability background

Monte Carlo methods are stochastic and therefore can only be analyzed from a statistical viewpoint. Hence, we first review some basic principles from probability theory before describing Monte Carlo integration. One of the most fundamental object in probability theory are **random variables** X. For simplicity let us first regard discrete random variables. They can be seen as objects which can take values in some subspace of the natural numbers, i.e. $x \in \Omega \subseteq \mathbb{N}$. For each of the values we assign a probability $P(x) \in [0, 1]$. These probabilities are defined via the property that they sum up to one

$$\sum_i P(x_i) = 1. \tag{11.2.1}$$

A classical example of a discrete random variable is a coin toss. Here the realizations are head H and tail T. To map this to the language introduced above we realize that we can identify head and tail with 0 and 1, thus our space is given by $\{\text{Head}, \text{Tail}\} \equiv \{0, 1\} \equiv \Omega$. Our random variable is the outcome of a toss X, where for a fair coin we assume the probability for each of the two states to be the same $P(\text{Head}) = P(\text{Tail}) = 0.5$. From this canonical example we can now form one of the most important discrete probability distributions $\mathbb{P}(X)$, the **Binomial distribution**. It gives the probability of trowing k heads in n tosses, given the probability of tossing a single head $\mathbb{P}(X = \text{Head}) = p$

$$P(k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{11.2.2}$$

In general we do not only regard discrete spaces, but continuous cases. In order to treat them we have to introduce further objects. From a mathematical point of view this requires us to take a step back from the gained view that we can assign a probability to each element of a set, but rather to intervals on the space.

11.2.1 Cumulative distribution and density functions

The **cumulative distribution function** $F_X(x)$, or CDF, of a continuous random variable X (i.e. an object which takes value in a subset of the real numbers $x \in \Omega \subseteq \mathbb{R}$) is the probability P that a value chosen from the variable's distribution is less than or equal to some threshold x

$$F_X(x) = P(X \leq x). \tag{11.2.3}$$

The corresponding **probability density function** p, or PDF, is the derivative of the CDF:

$$p(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x). \tag{11.2.4}$$

CDFs are always monotonically increasing, which means that the PDF is always non-negative $\mathrm{pdf}(x) \geq 0$. Furthermore they also integrate up to one

$$\int p(x)\mathrm{d}x = 1 \tag{11.2.5}$$

It is important to realize although they share some properties, the PDF can not be identified directly with the probability of an event as it was the case in the discrete setting. The introduces objects give rise to an important relationship and other property of the PDF, namely that the probability that a random variable lies within an interval is given by

$$P(a \leq X \leq b) = \int_a^b p(x)\mathrm{d}x. \tag{11.2.6}$$

A typical example here is the **uniform distribution** $\mathcal{U}([a,b])$, which is defined on an interval $[a,b]$ and whose PDF reads

$$p_{\mathcal{U}}(x) = \frac{1}{b-a} \tag{11.2.7}$$

This can be readily be generalized to a domain $\Omega \subseteq \mathbb{R}^n$, where we then find $p_{\mathcal{U}}(\boldsymbol{x}) = \frac{1}{|\Omega|}$ with the volume of the domain denoted by $|\Omega|$. The second very important continuous distribution is the **normal distribution** $\mathcal{N}(\mu, \sigma^2)$ which is defined via it's mean μ and standard deviation σ. It's pdf reads

$$p_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{11.2.8}$$

11.2.2 Expected value and variance

The **expected value** or **expectation** $\mathbb{E}$ of a random variable X over a domain Ω is defined as

$$\mathbb{E}[X] = \langle X \rangle = \int_\Omega x p(x) \mathrm{d}x. \tag{11.2.9}$$

We remark that given X is a random variable also $f(X)$ is a random variable, thus computing the expectation value of a random variable is closely related to integration. In the discrete setting this reduces to the **mean**

$$\mathbb{E}(x) = \bar{x} = \sum_i x_i P(x_i) \tag{11.2.10}$$

It is easy to see from this definition, that this is a linear operation, i.e. for any constant a, b and random variables X, Y we have

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{11.2.11}$$

The **variance** Var of a random variable X over a domain Ω is defined as

$$\mathrm{Var}[X] = \sigma^2[X] = \mathbb{E}\Big[(X - \mathbb{E}[X])^2\Big], \tag{11.2.12}$$

where σ, the **standard deviation**, is the square root of the variance. Using linearity of the expectation value it is possible to derive a simpler expression for the variance:

$$\sigma^2[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2. \tag{11.2.13}$$

If two random variables X, Y are **uncorrelated**, i.e. for the expectation value of a product of two variables we find

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y], \tag{11.2.14}$$

then the linearity also holds for the variance

$$\sigma^2\Big[\sum_i a_i Y_i\Big] = \sum_i a_i^2 \sigma^2[Y_i]. \tag{11.2.15}$$

Another property is **independent**, in which case the expectation of products of random variables can be factored, nonetheless this property is stronger and not all uncorrelated variables are independent.

11.3 Monte Carlo Integration

Before going into the formal details, let us regard the problem of estimating the value of π. To achieve this using integration techniques, we recall that the area of a circle with radius r is πr^2. Via the computation of the area of a quarter of a circle ("the pond") with radius $r = 1$, we would obtain the value π, see Figure 11.1. This means, that we need to integrate function $f : [0,1]^2 \to \mathbb{R}$, defined by

$$f(x,y) = \begin{cases} 1 & \text{if } \sqrt{x^2 + y^2} \le 1, \\ 0 & \text{else,} \end{cases} \tag{11.3.1}$$

over the domain $[0,1]^2$.

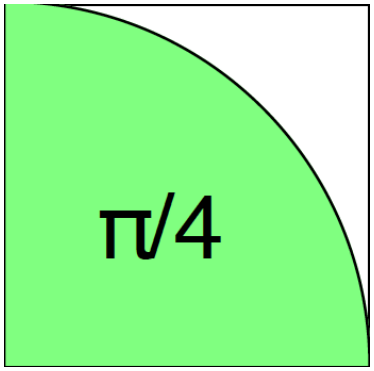


Figure 11.1: The area of the quarter circle with radius $r = 1$ is equal to $\pi/4$.

Instead of using the ideas from the previous lectures we want to find a way to approximate the value of the integral by sampling random points from the domain count how many stones hit the pond (i.e. how many coordinates (x, y) lie inside the circle). Let us make this idea formal:

For a given multi-dimensional integrand $f : \mathbb{R}^d \to \mathbb{R}$, we denote the integral by

$$I = |\Omega|\langle f \rangle, \tag{11.3.2}$$

where we can identify $\langle f \rangle$ as the previously introduced expectation value of the integrand f over Ω, assuming an uniform distribution

$$\langle f \rangle = \frac{1}{|\Omega|} \int_\Omega f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x}, \qquad |\Omega| = \int_\Omega \mathrm{d}\boldsymbol{x}. \tag{11.3.3}$$

To approximate $\langle f \rangle$, instead of evaluating f at n^d locations obtained from a d-fold Cartesian product of one-dimensional quadrature rules, we evaluate it at M points $\boldsymbol{x}_i$ (called "samples") chosen from an uniform random distribution in Ω (i.e. regarded as samples from a random variable) and compute the average of all values (perform "sampling")

$$\langle f \rangle \approx \langle f \rangle_M = \frac{1}{M} \sum_{i=1}^{M} f(\boldsymbol{x}_i). \tag{11.3.4}$$

Two interpretations of MC estimation are given in Figure 11.2, where MC sampling is performed for a one-dimensional function f on domain $|\Omega| = b - a$ and the number of samples is set to $M = 4$.

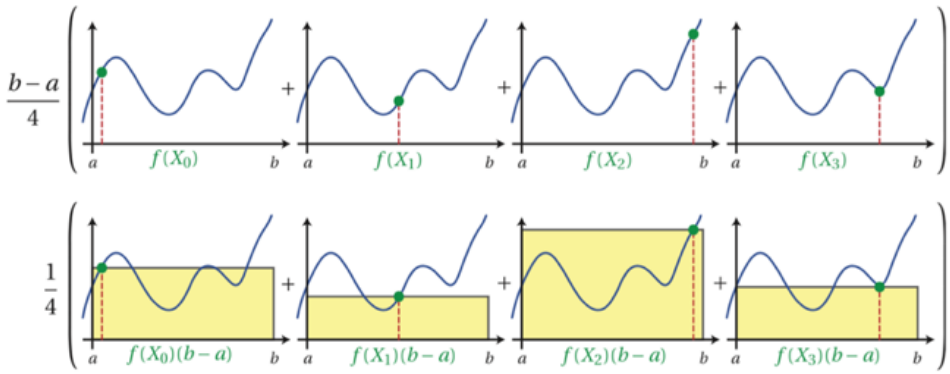


Figure 11.2: Illustration of two interpretations of the Monte Carlo estimator (11.3.4): computing the mean value (or height) of the function and multiplying by the interval length (top), or computing the average of several rectangle rules with random evaluation locations.

Due to the random nature of Monte Carlo sampling (changing random number sequence would result in a slightly different result), the estimate $\langle f \rangle_M$ itself is a random variable. However, assuming that all samples $\boldsymbol{x}_i$ are independent, we obtain that the expected value of this estimate $\langle f \rangle_M$ is the exact integral:

$$\mathbb{E}[\langle f \rangle_M] = \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} f(\boldsymbol{x}_i)\right] = \frac{1}{M}\sum_{i=1}^{M}\mathbb{E}[f(\boldsymbol{x}_i)] = \frac{1}{M}\sum_{i=1}^{M}\langle f \rangle = \frac{1}{M}M\langle f \rangle = \langle f \rangle. \tag{11.3.5}$$

Such estimators are called unbiased ("true") estimators. Furthermore, as we increase the number of samples M, the estimator $\langle f \rangle_M$ becomes a close and closer approximation of $\langle f \rangle$. Due to the Strong Law of Large Numbers, in the limit we can guarantee that we have the exact solution:

$$\mathbb{P}\left\{\lim_{M\to\infty}\langle f \rangle_M = \langle f \rangle\right\} = 1. \tag{11.3.6}$$

11.3.1 Estimating the error

To compare the accuracy and efficiency of Monte Carlo sampling to other quadrature methods, we need to estimate its error. We define the error to be the standard deviation (i.e., the root mean square (RMS) error) of the difference between random estimate $\langle f \rangle_M$ and exact deterministic integral $\langle f \rangle$,

$$\varepsilon_M = \sqrt{\mathrm{Var}[\langle f \rangle_M - \langle f \rangle]} \tag{11.3.7}$$

Using the expression from equation 11.2.13, the linearity of the expectation value and the fact that $\mathbb{E}[\langle f \rangle_M] = \langle f \rangle$ and is a deterministic quantity, we can further simplify ε_M to

$$\varepsilon_M = \sqrt{\mathrm{Var}[\langle f \rangle_M]}. \tag{11.3.8}$$

Next, we will derive a closed form expression for the error ε_M in terms of the variance of the integrand f and the number of samples M. Let us therefore plug in the definition of $\langle f \rangle_M$ into our simplified expression for the variance (equation 11.2.13), use the that $\langle \langle f \rangle_M \rangle = \langle f \rangle$ and that $f(\boldsymbol{x}_i)$ and $f(\boldsymbol{x}_j)$ are independent if $i \neq j$:

$$\begin{aligned}
\varepsilon_M^2 = \mathrm{Var}[\langle f \rangle_M] &= \langle \langle f \rangle_M^2 \rangle - \langle \langle f \rangle_M \rangle^2 \\
&= \frac{1}{M^2} \sum_{i,j=1}^{M} \Big(\mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)] - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} \sum_{i=1}^{M} \Big(\mathbb{E}[f(\mathbf{x}_i)^2] - \langle f \rangle^2 \Big) + \frac{1}{M^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{M} \Big(\underbrace{\mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)]}_{= \mathbb{E}[f(\mathbf{x}_i)] \mathbb{E}[f(\mathbf{x}_j)] = \langle f \rangle^2} - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} \sum_{i=1}^{M} \Big(\langle f^2 \rangle - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} M \Big(\langle f^2 \rangle - \langle f \rangle^2 \Big) = \frac{\langle f^2 \rangle - \langle f \rangle^2}{M} = \frac{\mathrm{Var}[f]}{M}.
\end{aligned} \tag{11.3.9}$$

We have obtained that the variance of the Monte Carlo estimator $\langle f \rangle_M$ is M times smaller than the initial variance of the integrand f. Hence, Monte Carlo method is $1/2$-order accurate, i.e.

$$\varepsilon_M = \sqrt{\frac{\mathrm{Var}[f]}{M}} = \mathcal{O}(M^{-1/2}). \tag{11.3.10}$$

Notice, that order $1/2$ is lower than any of the quadrature rules discussed so far. Order $1/2$ implies that in order to reduce the error by a factor of 2, we would need to evaluate 4 times as many samples. However, there is a trade-off here: the order does *not* depend on the dimension d of the integrand f. Having such dimension-independent accuracy, Monte Carlo sampling has a significant advantage for very high-dimensional integrals. As an example we can compare Monte Carlo sampling to Simpson's rule with order of accuracy $4/d$: we see that Monte Carlo quadrature is more efficient (meaning $1/2 > 4/d$) for any dimension d higher than 8.

Remark The error ε_M of the Monte Carlo estimator is *not* a strict bound of the error, meaning that the following inequality does is *not always* satisfied:

$$|\langle f \rangle - \langle f \rangle_M| \leq \varepsilon_M. \tag{11.3.11}$$

Instead, ε_M describes the most probable values of the error, i.e. for large number of samples M, we expect

$$|\langle f \rangle - \langle f \rangle_M| < \begin{cases} \varepsilon_M, & \text{with probability of 68\%,} \\ 2\varepsilon_M, & \text{with probability of 95\%,} \\ 3\varepsilon_M, & \text{with probability of 99\%.} \end{cases} \tag{11.3.12}$$

11.3.2 Summary

Recipe for Monte Carlo Integration of uncorrelated data is as follows:

1. Sample a points $\mathbf{x}_i$ from an uniform distribution and evaluate the integrand f to get random variables $f(\mathbf{x}_i)$.

2. Store the number of samples, the sum of values, and the sum of squares

$$M, \quad \sum_{i=1}^{M} f(\mathbf{x}_i) \quad \sum_{i=1}^{M} f(\mathbf{x}_i)^2. \tag{11.3.13}$$

3. Compute the mean as the estimate of the expectation (normalized integral)

$$\frac{I}{|\Omega|} = \langle f \rangle \approx \langle f \rangle_M = \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}_i). \tag{11.3.14}$$

4. Estimate the variance using the unbiased sample variance, which is given as follows

$$\mathrm{Var}[f] \approx \frac{M}{M-1} \left(\frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}_i)^2 - \langle f \rangle_M^2 \right) = \frac{M}{M-1} \Big(\langle f^2 \rangle_M - \langle f \rangle_M^2 \Big), \tag{11.3.15}$$

   and use it to estimate the error

$$\varepsilon_M = \sqrt{\frac{\mathrm{Var}[f]}{M}} \approx \sqrt{\frac{1}{M-1} \Big(\langle f^2 \rangle_M - \langle f \rangle_M^2 \Big)}. \tag{11.3.16}$$

11.4 Non-Uniform Distributions

In practical applications - as to compute macroscopic quantities of a liter of water in the statistical physics context - we usually apply Monte Carlo Integration to estimate the expected value of a function $f(x)$ over some probability density function $p(x)$, i.e.,

$$\mathbb{E}_p[f] = \int f(x) p(x) \, \mathrm{d}x. \tag{11.4.1}$$

The Monte Carlo estimation of this expected value is

$$\mathbb{E}_p[f] \approx \frac{1}{M} \sum_{i=1}^{M} f(x_i) \tag{11.4.2}$$

where $\{x_i | i = 1, \ldots, M\}$ is a set of M samples drawn from probability distribution with probability density function $p(x)$. Recall that for the integrals of the form $I = \int_\Omega f(x) \, \mathrm{d}x$ that we considered so far, we have

$$I = |\Omega| \langle f \rangle = |\Omega| \mathbb{E}_p[f], \tag{11.4.3}$$

where

$$p(x) = \begin{cases} \frac{1}{|\Omega|}, & \text{if } x \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{11.4.4}$$

Hence, we would need to draw samples from uniform distribution. Uniformly distributed random numbers can be obtained using the so-called pseudo-random number generators, available in most of the commercial and free software. Hence, we will assume that such random number are available. For other problems where $p(x)$ is non-uniform, we would like to generate non-uniform random numbers for estimating the expected value $\mathbb{E}_p[f]$.

11.4.1 Inverse Transform Sampling

Let X be a random variable (RV) with a probability density function (PDF) $p_X(x)$ and cumulative density function (CDF) $F_X(x)$. Samples from the PDF $p_X(x)$ can be generated using the **Inverse Transform Sampling** method, which makes use of the samples $u^{(i)}$, $i = 1, \ldots, N$ obtained from a uniformly distributed random variable $U \sim \mathcal{U}([0,1])$. In order to do so we want to construct a transformation between the values of the random variables X and U

$$x = g(u) \tag{11.4.5}$$

For our uniform random variable U, one has that $p_U(u) = 1$ for $u \in [0,1]$ and the cdf correspondingly reads

$$F_U(u) = \int_0^u p_U(s)\mathrm{d}s = u \tag{11.4.6}$$

Using the fact that for any cdf we find $F_X(x) \in [0,1]$ where the value grows monotonically we can define the value x via the requirement that

$$F_X(x) = u \tag{11.4.7}$$

Thus the following transformation is true

$$x = F_X^{-1}(u) \tag{11.4.8}$$

which means that the values of x are given by the inverse of the CDF $F_X(x)$ which also this inverse represents exactly the function $g(u)$. This transformation allows to draw samples $x^{(i)}$, $i = 1, \cdots, N$ from the PDF $p_X(x)$ as

$$x^{(i)} = F_X^{-1}(u^{(i)}) \tag{11.4.9}$$

where $u^{(i)}$ are samples from the uniform distribution over interval $[0,1]$.

Another (more formal way) way to see this and easily check whether a given transformation fulfills the wished relation goes vie the substitution rule. Consider the following integral and the transformation $\boldsymbol{x} = g(\boldsymbol{u})$

$$\int_{g(U)} p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \int_U p(\boldsymbol{u})|J(\boldsymbol{x})|^{-1}\mathrm{d}\boldsymbol{u} \tag{11.4.10}$$

Where J denotes the Jacobian and $g(U)$ the transformed domain.

Sampling from the Exponential Distribution Use the inverse transform sampling method to sample from the exponential distribution

$$p(x) = \lambda e^{-\lambda x}, x > 0 \tag{11.4.11}$$

The CDF of the exponential distribution is

$$F(x) = \int_0^x p(x)dx = \int_0^x \lambda e^{-\lambda x} dx = 1 - e^{-\lambda x} \tag{11.4.12}$$

Thus the transformation $x = g(u)$ which is obtained from $x = F^{-1}(u)$ or equivalently $F(x) = u$ is derived by solving

$$1 - e^{-\lambda x} = u \tag{11.4.13}$$

with respect to x to yield

$$x = -\frac{1}{\lambda}\ln(1-u) \tag{11.4.14}$$

The transformation (11.4.14) between the random variable X and the standard uniform variable U defines an exponentially distributed random variable X and it allows to draw samples $x^{(i)}$, $i = 1,\ldots,N$ from the PDF $p(x)$ as

$$x^{(i)} = -\frac{1}{\lambda}\ln(1-u^{(i)}) \tag{11.4.15}$$

where $u^{(i)}$, $i = 1,\ldots,M$ are samples drawn from the standard uniform distribution.

Sampling from Normal (Gaussian) distribution The inverse transform sampling method requires the inversion of the CDF $F_X(x)$. This may be time consuming for cases where $F_X^{-1}(u)$ is not known in closed form as, for example, the Normal distribution.

For Normal distribution, an alternative, called **Box-Muller Transformation**, yields an exact method that uses the inverse transform method to convert two independent uniform random variables $u_1, u_2 \sim \mathcal{U}([0,1])$ into two independent normally distributed random variables $x_1, x_2 \sim \mathcal{N}(0,1)$ via the following transformation

$$\begin{aligned} x_1 &= \sqrt{-2\ln(u_1)}\cos(2\pi u_2) \\ x_2 &= \sqrt{-2\ln(u_1)}\sin(2\pi u_2) \end{aligned} \tag{11.4.16}$$

To see that this indeed holds we invert the above equation and find

$$\begin{aligned} u_1 &= \exp\left(-\frac{x_1^2 + x_2^2}{2}\right) \\ u_2 &= \frac{1}{2\pi}\tan^{-1}\left(\frac{x_2}{x_1}\right) \end{aligned} \tag{11.4.17}$$

Calculating the inverse Jacobean of the above transformation and taking the determinant implies the result. Intuitively we can motivate this approach by regarding an integral over the product of two Gaussian probability densities with $\sigma = 1$ and $\mu = 0$ over the unit sphere takes and transforming it to polar coordinates

$$\frac{1}{2\pi} \iint\limits_{x_1^2+x_2^2 \leq r^2} \exp\left(-\frac{x_1^2+x_2^2}{2}\right) \mathrm{d}x_1 \mathrm{d}x_2 = \frac{1}{2\pi} \int_0^{2\pi} \int_0^r \exp\left(-\frac{r^2}{2}\right) r \,\mathrm{d}r \,\mathrm{d}\varphi = 1 - \exp\left(-\frac{r^2}{2}\right) \tag{11.4.18}$$

Setting the right hand side to be u_1 and solve for r we find the first factor of equation 11.4.16. The second factor corresponds to the transformation of the uniformly distributed angle on the unit sphere.

In the literature we also find a second form, the so called **Marsaglia polar method**, where the transformation reads

$$\begin{aligned} x_1 &= u_1 \left(\frac{-2\ln(r^2)}{r^2}\right)^{1/2} \\ x_2 &= u_2 \left(\frac{-2\ln(r^2)}{r^2}\right)^{1/2} \end{aligned} \tag{11.4.19}$$

Here we use the complex representation of points on a circle.

11.5 Rejection Sampling

Another method for generating random numbers according to a distribution, suggested by von Neumann is Rejection Sampling. In some sense it is a generalization of the ideas we used for the integration in the uniform case. One should find a simple distribution with probability density function $h(x)$ from which we already know how to generate samples, with the property that $h(x)$ bounds $p(x)$, i.e. such that

$$p(x) < \lambda h(x) \tag{11.5.1}$$

for some $\lambda \in \mathbb{R}$. For graphical explanation, refer to Figure 11.3. Then the algorithm continues as follows:

1. draw a random sample x from distribution $h(x)$,
2. draw a uniform random number u in the interval $[0,1]$
3. accept x if $u < \dfrac{p(x)}{\lambda h(x)}$, otherwise reject (forget) x,
4. continue the same procedure until sufficiently many (accepted) samples generated.

One of the easiest (not always the best or even appropriate) choice for $h(x)$ is a uniform distribution, as depicted in Figure 11.3. In this case the method is equivalent to integration approach. However,

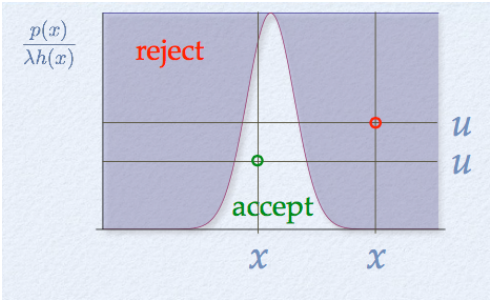


Figure 11.3: Rejection Sampling method

the probability of rejection increases exponentially with d, i.e., for high dimensional distributions (d is large), we are facing the same curse of dimensionality that we encountered for other deterministic Quadrature rules.

11.6 Importance Sampling

For distribution functions that are highly irregular, multi-modal and peaked, Rejection sampling will waste a lot of effort in regions of low importance (i.e., rejection occurs significantly more often than acceptance). An example of this kind of function can be seen in Figure 11.4 (left). Ideally, we would like to focus our sampling effort in the area where the distribution has more volume in order to explore it better. For example, imagine that you want to sample a rare event. You know, for instance, that you have a distribution function that shows a significant basin around a tiny support of 10^{-5} around 0. We would like to bias the random draws so that the majority of the "candidate" points x are being selected around that area; however, such bias needs to be accounted for later. This idea leads to a method called Importance Sampling.

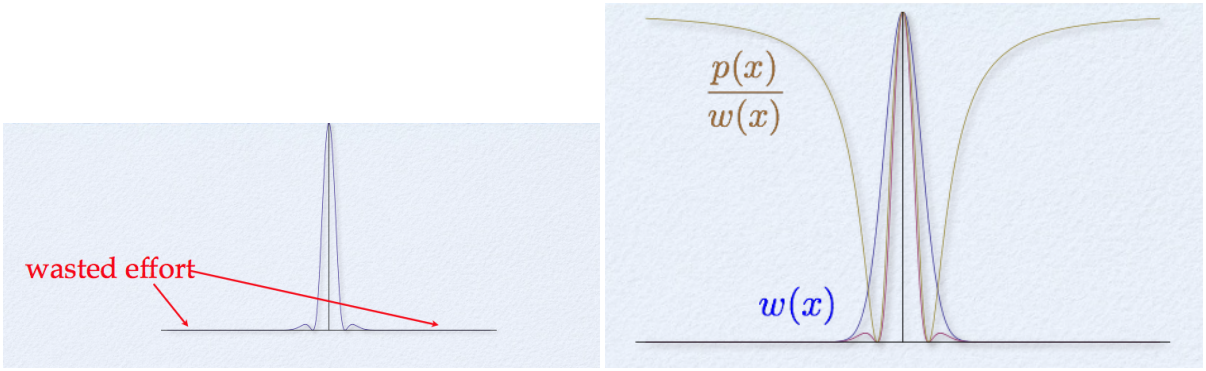


Figure 11.4: Left: Wasted effort while choosing sampling points in areas of low importance. Right: Illustration of usage of importance sampling to overcome difficulties arising from uniform sampling strategies.

Importance sampling allows us to draw samples x that are distributed as probability $w(x)$, instead of a

uniform distribution. We compensate for the bias by normalizing $p(x)$ by the very same "importance" function $w(x)$ and sample $p(x)/w(x)$ instead, resulting in the following integral:

$$\langle f \rangle_p = \int_a^b f(x) \frac{p(x)}{w(x)} w(x) dx \approx \frac{1}{M} \sum_{i=1}^{M} f(x_i) \frac{p(x_i)}{w(x_i)}, \tag{11.6.1}$$

with each x_i being sampled from distribution $w(x)$. For an example distribution $w(x)$, refer to Figure 11.4 (right).

Exam checklist

After this class, you should understand the following points regarding numerical integration:

- How can quadrature rules be extended to arbitrary dimensions using the Cartesian product approach.
- What is the curse of dimensionality?
- How does Monte-Carlo integration work? What are its properties?
- How can be generate random numbers from an arbitrary distribution, given uniformly distributed numbers?
- How do Inverse-, Rejection- and Importance-Sampling work?

LECTURE 12

Modeling Data: Bayesian inference

Recall our cocktail example from the first lecture. Our aim was to determine the optimal amount of shaking. In order to determine this, we measured the temperature of the cocktail after different times. We then fitted a model to our data, allowing us to determine the point where we did not observe any further decrease of temperature (see figure 12.1)

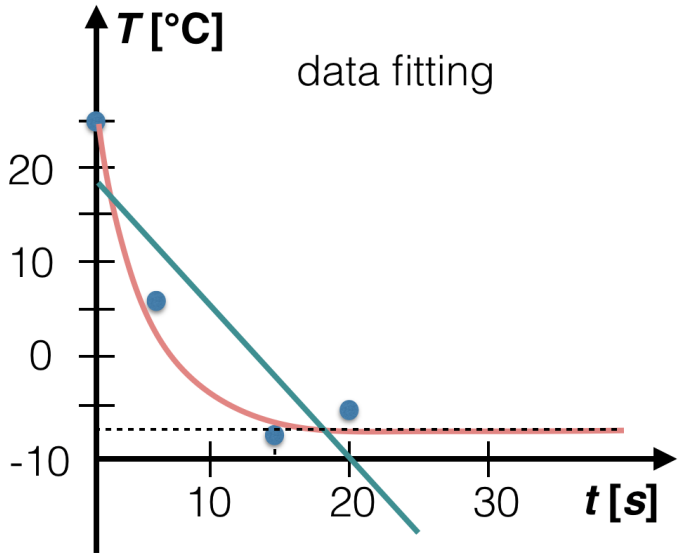


Figure 12.1: Example on shaking (see http://www.cookingissues.com/index.html%3Fp=1527.html): we would like to know how long one should shake a cocktail. We insert a thermostat into the shaker and measure the temperature at some time instances during the shaking of the cocktail.

We saw that our choice of model was by no means unique, but we where free to chose different properties such as interpolation vs. fitting, the function type, whether to take linear or non-linear coefficients and many more. In this lecture we want to address this problem by setting up a probabilistic measure allowing us to compare our models in a probabilistic way. More formally we want to regard the probability of a certain model, given the experimental data at hand. In mathematical language we write this as a conditional probability

$$P(\text{model}|\text{data}) \tag{12.0.1}$$

We will spend the first part of the lecture to make this expression more clear, i.e. introduce **Bayesian inference**. This is a particular type of inferential statistics which has the goal to estimate properties of a population based on samples from this population. This thus has very wide ranging applications.

Let us summarize:

We are given:

1. Data measured in an experiment
2. Models based on certain postulates about our system

Our goal is:

1. Estimate the Parameters of our Model
2. Compare the wealth of different Models

In order to make this formal we will extends our preliminary knowledge of probability theory:

12.1 Probability Theory

We already saw some of the basics of probability theory in the last lecture. Here we want to make another step back and discuss more of the basics and also re-frame the ideas to our current topic. Let us do this by first comparing the core objects for our current treatment with the ones from the last lecture

experiment = process with unknown outcome, which were our random variables

range of values in experiment = state/sample space $\mathscr{S}$

data point = event, which is a subset of our sample space Ω

In order to ensure mathematical consistency we also have to add an unintuitive event, namely the null event, which formally is represented by the empty set, i.e. null event= ø = {}.

Example 12.1: Roll a Dice

Adding to the coin toss in the last lecture, another basic example is rolling a dice. Regarding a standard dice with six faces, we realize that our state or sample space is given by

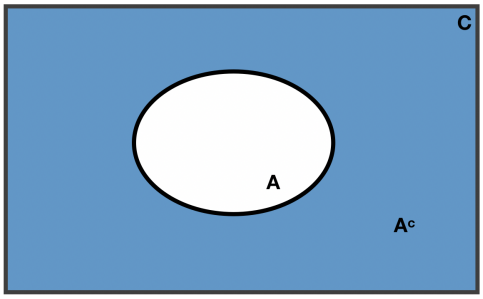
$$\mathscr{S} = \{1,2,3,4,5,6\} \tag{12.1.1}$$

Examples for events are for example drawing an odd number A, or an even number B

$$A = \{1,3,5\}, \qquad B = \{2,4,6\} \tag{12.1.2}$$

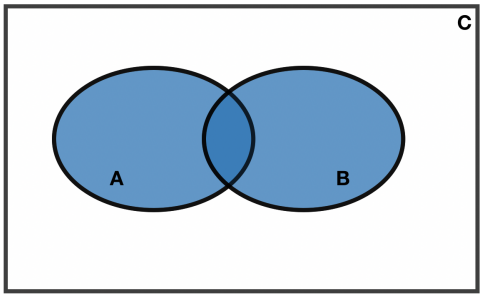
From the basic example given above we realize that our main object we operate with are sets, thus we need some of the fundamentals of set theory to make this formally correct. Let us regard the basic operations from set theory:

complement of a set $A^c = \mathscr{S} \setminus A$, as a Venn-diagram:



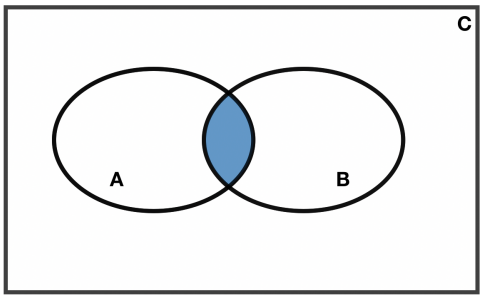
In the dice example the complement of A is B.

union of two sets $A \cup B$, as a Venn-diagram:



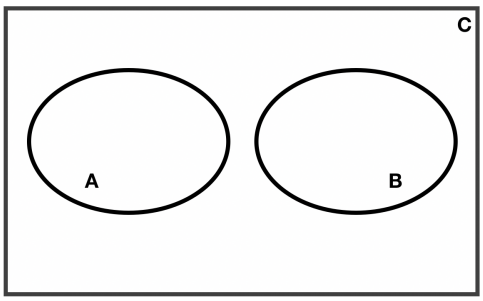
In the dice example the union is the whole space $A \cup B = \mathscr{S}$.

intersection of two set $A \cap B$, as a Venn-diagram:



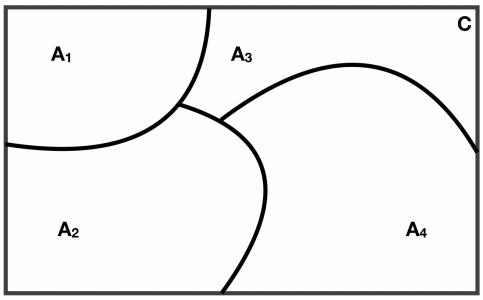
In the dice example the intersection is the empty set $A \cap B = \emptyset$.

disjoint sets Disjoint sets do not have an intersection $A \cap B = \emptyset$. As a Venn diagram:



The example events for the dice are disjoint.

exhaustive sets We call sets $A_1, \ldots, A_n$ exhaustive, if they span the whole space. As a Venn diagram:



The example events are exhaustive.

Having these set theoretical notations at hand we can continue constructing. our probability space by adding a measure, which we will see as the probability of an event $P(A)$ (i.e. the chance that this event will eventually happen). This function is defined by the following set of axioms:

non-negativity For all event A the probability is non-negative $P(A) \geq 0$

unitarity The probability that any event happens is one $P(\mathscr{S}) = 1$

additivity For any countable set of of disjoint events $A_1, \ldots, A_n, \ldots$ we find

$$P(A_1 \cup A_2 \ldots) = \sum_{i=1}^{\infty} P(A_i)$$

From these axioms we can find some equivalent statements and settle further interesting notations

- From unitarity and additivity it follows that $P(\emptyset) = 0$
- From the above and additivity is follows that for any event $A \neq \{\emptyset, \{\mathscr{S}\}\}$ we have $P(A) \leq 1$
- For two sets A, B we find that the probability of the union is $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

→ A special case of the above rule are **independent events** where we have $P(A \cap B) = P(A) \cdot P(B)$.

This completes our fundamental description of probability theory. We can now move on and define the central object we discussed in the introduction, the **conditional probability** of an event A, given an event B

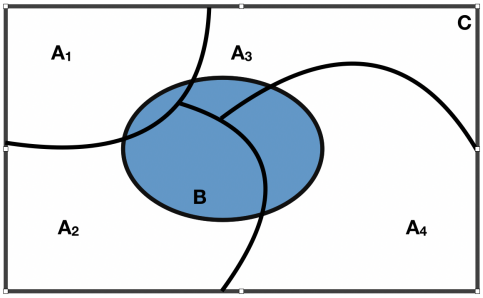
$$P(A|B) \equiv \frac{P(A \cap B)}{P(B)} \tag{12.1.3}$$

We can set the link to the above definition of independence by realizing that the $P(A|B) = P(A)$ in case of independent events.

12.2 Bayes' Rule

We are now almost ready to formulate the most fundamental theorem which will prove very useful for any calculation in Bayesian statistics, **Bayes' rule**. In order to understand it we need one further ingredient, the **Law of Total Probability**. Therefore consider a set of mutually disjoint and exhaustive events $A_1, \ldots, A_n$ (i.e. $A_i \cap A_j = \emptyset$ for all $i \neq j$ and $\bigcup_i A_i = \mathscr{S}$). We then find

$$P(B) = \sum_{i=1}^{n} P(B \cap A_i) \tag{12.2.1}$$



We may now formulate **Bayes' theorem** as

$$P(A_j|B) \overset{12.1.3}{=} \frac{P(A_j \cap B)}{P(B)} \overset{\substack{12.2.1 \\ A_j \cap B = B \cap A_j \\ 12.1.3}}{=} \frac{P(B|A_j)P(A_j)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)} \tag{12.2.2}$$

Example 12.2: Rare Events

One of the most astonishing insights we can gain with the derived rule is considered with rare events. Let us for example consider cancer. We know that approximately one in thousand has cancer. Furthermore we know that the available cancer tests have the following properties:

- In 93% of the cases the test correctly shows a positive results (i.e. it shows positive given the patient has cancer)
- In 99% of the cases the test correctly shows a negative results (i.e. it shows negative given the patient has no cancer)

Given this information we now want to find the probability that a patient has cancer, given the test is positive. To do that let us transform the above into our introduced language of probability

Event A A patient has cancer

Event B The test is positive

The above number now translate as follows

Probability of having cancer $P(A) = 0.001$

Probability of positive test, given cancer $P(B|A) = 0.93$

Probability of negative test, given no cancer $P(B^c|A^c) = 0.99$

Probability of cancer, given positive test $P(A|B) =$?

Let us now invoke Bayes theorem to answer the posed question

$$P(A|B) = \frac{P(B|A)P(A)}{P(B|A)P(A) + P(B|A^c)P(A^c)} \tag{12.2.3}$$

We realize that we have all this information, given the summation rule, which allows us to calculate $P(B|A^c) = 1 - P(B^c|A^c)$. Filling in the numbers results in $P(A|B) = 0.09$, thus despite the fact that the tests are quite accurate the probability to have cancer when obtaining a positive test is really low. This steams from the fact that the overall probability to have cancer in the first place is very low.

12.3 Parameter Estimation

Having the formalism and Bayes theorem at hand we can now move on with the initially promised task of examining our models using methods from probability theory.

Suppose we have observed some set of data D and we construct a model M that explains the data. We assume that we have a computational model that depends on some parameters. These parameters are considered to be random variables and will be denoted by θ. A prior distribution $P(\theta \mid I)$ can be imposed on them, e.g., if we know that θ takes only positive values, $P(\theta \mid I)$ can be the gamma

distribution. We can use the Bayes' Rule to infer the parameters θ of the model M

$$P(\theta \mid D, I) = \frac{P(D \mid \theta, I)\, P(\theta \mid I)}{P(D \mid I)}, \tag{12.3.1}$$

where I stand for any other information we might have. The $P(\theta \mid D, I)$ is called the *posterior* probability, the $P(D \mid \theta, I)$ is the *likelihood*. The denominator $P(D|I)$ is the evidence of the model, which (in the case of continuous variables) can be written as

$$P(D \mid I) = \int P(D \mid \theta, I)\, P(\theta \mid I)\, \mathrm{d}\theta. \tag{12.3.2}$$

It is the normalizing constant that makes the right hand side of 12.3.1 a probability density function. Using 12.3.1 we are able to find the distribution of the parameters conditioned on the data. Stated differently, we can answer the question "what values for the parameters will make the computational model fit the data better?".

We would like to know

- the model parameter for which the posterior density function is maximized,
- measure of reliability of the best estimate.

As we will see below, the posterior distribution around the best estimate can be locally approximated with a Gaussian distribution, by employing the Laplace approximation method.

12.4 Laplace Approximation

The best estimate θ_0 is the global maximum of the posterior distribution. Thus, the following two conditions must hold

$$\left.\frac{\partial P}{\partial \theta}\right|_{\theta_0} = 0, \tag{12.4.1}$$

$$\left.\frac{\partial^2 P}{\partial \theta^2}\right|_{\theta_0} < 0. \tag{12.4.2}$$

In order to avoid numerical issues for low probability events and simplify some of the appearing calculation we consider the natural logarithm of our posterior distribution

$$L(\theta) = \log(P(\theta)). \tag{12.4.3}$$

From $\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial P}\frac{\partial P}{\partial \theta} = \frac{P'}{P}$ we observe that this transformation of our density preserves the location of the maximum. By performing Taylor expansion of L around the maximum θ_0 we have

$$L(\theta) = L(\theta_0) + \frac{1}{2}\left.\frac{\partial^2 L}{\partial \theta^2}\right|_{\theta_0} (\theta - \theta_0)^2 + \mathcal{O}\left((\theta - \theta_0)^3\right), \tag{12.4.4}$$

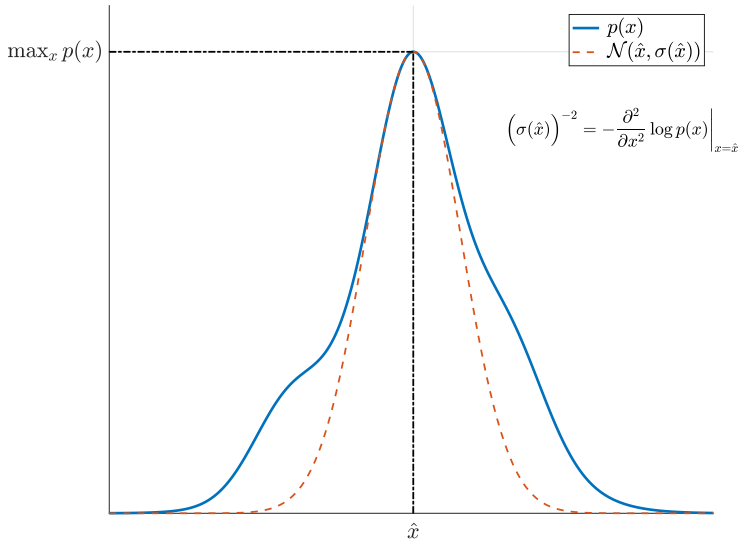


Figure 12.2: Laplace approximation. In the limit of many observation data the probability distribution function $P(\theta \mid D, I)$ is locally approximated as a Gaussian around the best estimate, i.e. the value that maximizes the density function.

where we used 12.4.1. Keeping only terms up to second order, we can write the probability distribution as,

$$P(\theta \mid D, I) \approx A \exp\left(\frac{1}{2} \frac{\partial^2 L}{\partial \theta^2}\bigg|_{\theta_0} (\theta - \theta_0)^2\right), \tag{12.4.5}$$

where A is a constant $A = \exp\big(L(\theta_0)\big)$. We obtained a Gaussian approximation of the probability density function with variance

$$\sigma^2 = -\left(\frac{\partial^2 L}{\partial \theta^2}\bigg|_{\theta_0}\right)^{-1}. \tag{12.4.6}$$

This is positive as the second derivative is negative according to the condition 12.4.2. We can finally write the Gaussian approximation, omitting the normalization constant, as

$$P(\theta \mid D, I) \propto \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \big(\theta - \theta_0\big)^2\right). \tag{12.4.7}$$

We see that around θ_0 the gaussian approximation is a good approximation for the probability distribution function $P(\theta \mid D, I)$ (see Figure 12.2).

12.4.1 Example I: The coin flipping problem

A coin comes up heads 4 times in 16 flips.
Is this a fair coin?

Define H the bias-weighting of the coin. For example

- if $H = 0$: a tail comes at every flip,
- if $H = 1$: a head comes at every flip,
- if $H = \frac{1}{2}$: a fair coin.

Here, H plays the role of the model parameter θ. Suppose we observe "R heads in N tosses". We want to estimate the posterior distribution of H given the observed data $D = (R, N)$,

$$p(H \mid D). \tag{12.4.8}$$

Using Bayes' theorem, we write

$$p(H \mid D) \propto p(D \mid H)p(H). \tag{12.4.9}$$

Here, we omit the normalization factor for simplicity. We choose a uniform prior,

$$p(H) = \begin{cases} 1, & \text{if } 0 \le H \le 1, \\ 0, & \text{otherwise.} \end{cases} \tag{12.4.10}$$

Such prior is used when we do not have any prior knowledge about the fairness of the coin: it is equally probable to have a fair coin or to have a coin completely biased towards head. We need to define the likelihood function in 12.4.9. In words, the likelihood function measures the chance to observe certain data if the value of the bias-weighting is given. Assuming independent events, it is easy to observe that the likelihood of obtaining "R heads in N tosses" follows a binomial distribution,

$$p(D \mid H) \propto H^R (1 - H)^{N-R}. \tag{12.4.11}$$

This is intuitively derived by considering

- H^R is the probability of having R "heads",
- $(1 - H)^{N-R}$ is the probability of having $N - R$ "tails".

Note that we again omitted the constant factor in 12.4.11 as it does not depend on H. Posterior distributions of the bias-weighting of the coin H are shown on Figure 12.3, starting from three different priors. The different lines show posterior densities for different priors. The first figure for 0 data points represents the three priors. It can be seen that after many observations, the three posterior densities

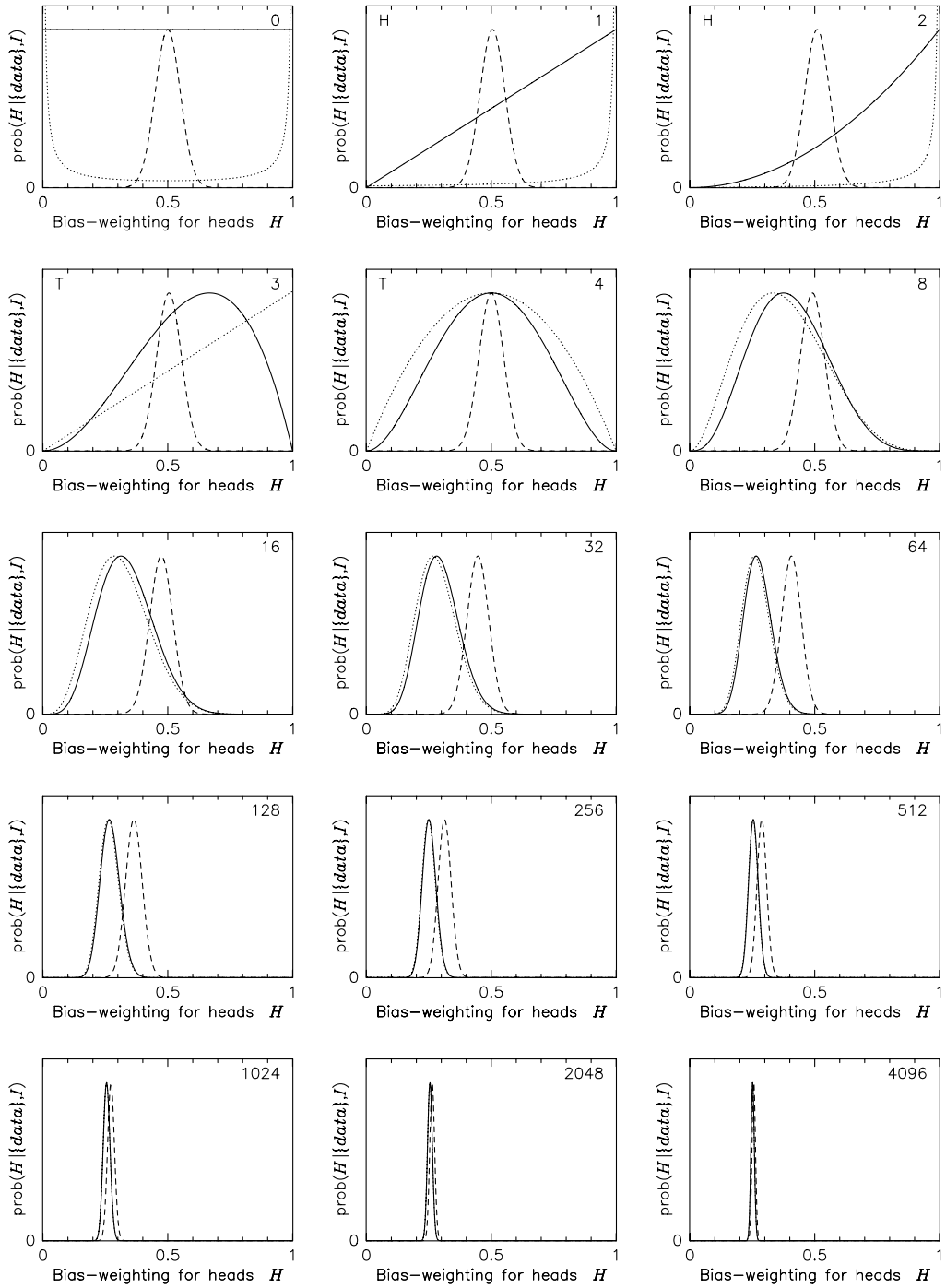


Figure 12.3: Evolution of the posterior density of the bias-weighting of a coin as the number of data increases. Taken from D.Sivia and J. Skilling. Data analysis: a Bayesian tutorial. OUP Oxford, 2006.

converge to the same distribution. However, the effect of the prior is evident for smaller observation data sets, as the biased Gaussian prior converges later to the actual posterior density. Comparing 12.3 with our original problem, we can see that the probability of the coin to be fair still lies in the confidence region. However it is more likely that the coin is not fair, given the data. We need more data to increase our confidence about H.

In the Bayesian framework, we now approximate the posterior distribution of the parameter with a Gaussian distribution around the best estimate, i.e. the value that maximizes the posterior. Taking the logarithm of the posterior (eq. 12.4.9) using the found likelihood (eq. 12.4.11) and our assumption on the prior (eq. 12.4.10), we get

$$L(H) = \text{const} + R\log(H) + (N-R)\log(1-H), \tag{12.4.12}$$

where the constant does not play any role as we want to find the best estimate. The first two derivatives read,

$$\frac{\partial L}{\partial H} = \frac{R}{H} - \frac{N-R}{1-H}, \tag{12.4.13}$$

$$\frac{\partial^2 L}{\partial H^2} = -\frac{R}{H^2} - \frac{N-R}{(1-H)^2}. \tag{12.4.14}$$

The condition 12.4.1 gives the best estimate $\hat{H} = \frac{R}{N}$. The standard deviation is therefore given by

$$\sigma = \left(-\left.\frac{\partial^2 L}{\partial H^2}\right|_{\hat{H}} \right)^{-\frac{1}{2}} = \sqrt{\frac{\hat{H}(1-\hat{H})}{N}}. \tag{12.4.15}$$

Note that the certainty increases as we add more data. We can also notice that it is easier to detect a biased coin than a fair coin. Indeed, the uncertainty is maximized for $\hat{H} = \frac{1}{2}$.

12.4.2 Example II: Gaussian mean estimator

Consider N independent and identically distributed (i.i.d.) observations $D = (d_1, d_2, \ldots d_N)$. We assume that the data are randomly generated from a Gaussian distribution with known variance σ^2 and unknown mean $\theta = \mu$,

$$p(d_k \,|\, \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \left(d_k - \mu\right)^2 \right). \tag{12.4.16}$$

What is the best estimate for μ and what is our confidence for this estimate? From Bayes' theorem, the posterior probability of the mean μ is given by

$$p(\mu \,|\, D) \propto p(D \,|\, \mu)\, p(\mu). \tag{12.4.17}$$

Since the data is i.i.d., the likelihood function takes the form

$$p(D \,|\, \mu) = \prod_{k=1}^{N} p(d_k \,|\, \mu). \tag{12.4.18}$$

Here, we assume an uninformative uniform prior for the mean of the Gaussian,

$$p(\mu) = \begin{cases} c = \frac{1}{\mu_{max} - \mu_{min}}, & \mu_{\min} \le \mu \le \mu_{\max} \\ 0, & \text{otherwise.} \end{cases} \tag{12.4.19}$$

The posterior distribution is then given by

$$\begin{aligned} p(\mu \,|\, D) &\propto c \prod_{k=1}^{N} p(d_k \,|\, \mu) \\ &= c \prod_{k=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \left(d_k - \mu\right)^2 \right) \\ &= c \frac{1}{(2\pi\sigma^2)^{N/2}} \exp\left(-\frac{1}{2\sigma^2} \sum_{k=1}^{N} (d_k - \mu)^2 \right). \end{aligned}$$

We compute the logarithm of our posterior,

$$\begin{aligned} L(\mu) &= \log(p(\mu \,|\, D)), \\ &= \text{const} - \sum_{k=1}^{N} \frac{(d_k - \mu)^2}{2\sigma^2}. \end{aligned}$$

The best estimate $\hat{\mu}$ must satisfy

$$\begin{aligned} \left. \frac{\mathrm{d}L(\mu)}{\mathrm{d}\mu} \right|_{\hat{\mu}} &= \sum_{k=1}^{N} \frac{d_k - \hat{\mu}}{\sigma^2} = 0, \\ \Rightarrow \sum_{k=1}^{N} d_k &= \sum_{k=1}^{N} \hat{\mu}, \\ \Rightarrow \hat{\mu} &= \frac{1}{N} \sum_{k=1}^{N} d_k \end{aligned}$$

We compute the second derivative of the log-likelihood

$$\frac{\mathrm{d}^2 L}{\mathrm{d}\mu^2} = -\sum_{k=1}^{N} \frac{1}{\sigma^2} = -\frac{N}{\sigma^2}, \tag{12.4.20}$$

which is negative, meaning that $L(\hat{\mu})$ is indeed a maximum. Finally, the standard deviation of the posterior is equal to $\sigma/\sqrt{N}$ and the confidence interval on the best estimate is given by

$$\mu = \hat{\mu} \pm \frac{\sigma}{\sqrt{N}}. \tag{12.4.21}$$

12.5 Posterior robust prediction

We can use the posterior distribution of the parameters to predict another quantity of interest q by using the law of total probability.

$$P(q \,|\, D) = \int P(q, \theta | D) = \int P(q \,|\, \theta, D)\, p(\theta \,|\, D)\, \mathrm{d}\theta. \tag{12.5.1}$$

Thus, we can propagate the uncertainty in the parameters to the output of the model in order to quantify the uncertainty it the predictions.

12.6 Model selection

The Bayesian framework can also be used to compare different models. Suppose, we have 2 different models, i.e, M_1 and M_2. We can compute the $P(M_1 \mid D)$ and $P(M_2 \mid D)$. The ration between the two is the posterior ratio

$$\frac{P(M_1 \mid D)}{P(M_2 \mid D)} = \frac{P(D \mid M_1)}{P(D \mid M_2)} \frac{P(M_1 \mid I)}{P(M_2 \mid I)}. \tag{12.6.1}$$

If we assume that both models have only one parameter and a uniform prior, i.e., $P(M_1 \mid I) = \frac{1}{m_1^{max} - m_1^{min}}$ and $P(M_2 \mid I) = \frac{1}{m_2^{max} - m_2^{min}}$ which gives us

$$\frac{P(M_1 \mid D)}{P(M_2 \mid D)} = \frac{P(D \mid M_1)}{P(D \mid M_2)} \frac{m_2^{max} - m_2^{min}}{m_1^{max} - m_1^{min}}. \tag{12.6.2}$$

Exam checklist

After this class, you should understand the following points regarding Bayesian inference:

- What is Bayes' Rule and how can it be used?
- How can we perform a maximum-a-posteriori (MAP) estimate of parameters of our model?
- How does the Gaussian- / Laplace-Approximation work?
- How can we compare models using Bayesian Inference?