CLASS NOTES

Models, Algorithms and Data: Introduction to computing

2020

Petros Koumoutsakos, Jens Honore Walther, Julija Zavadlav, Georgios Arampatzis

(Last update: June 18, 2020)

IMPORTANT DISCLAIMERS

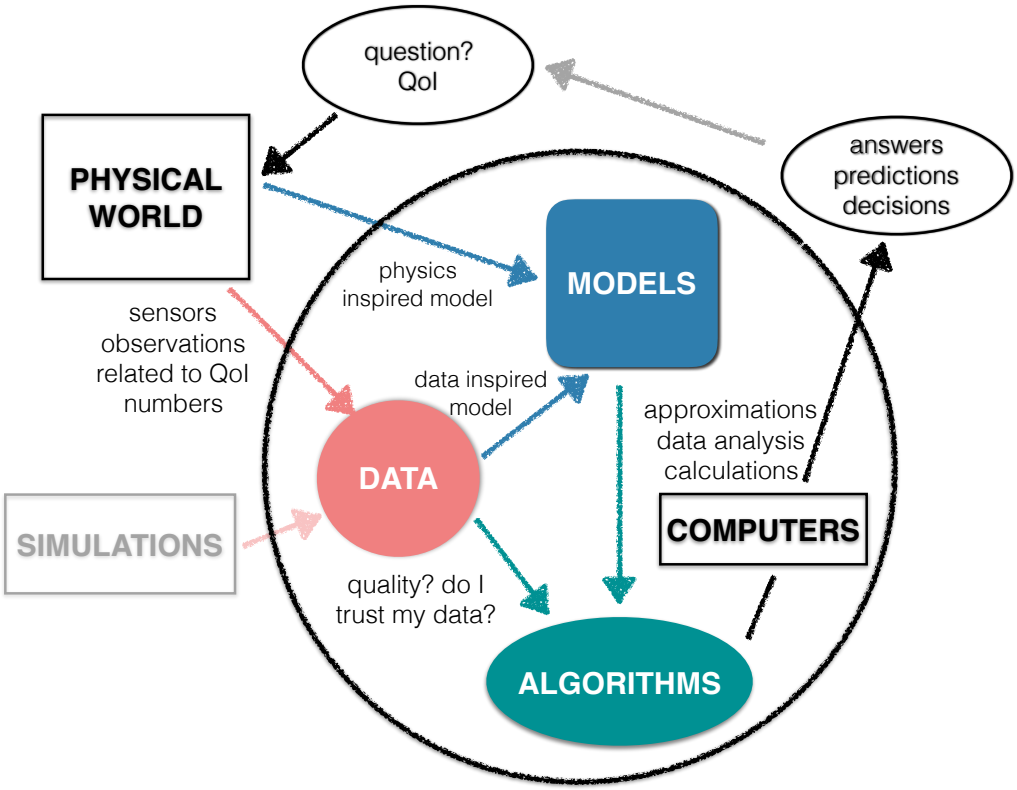
Much of the material (ideas, definitions, concepts, examples, etc) in these notes is taken for teaching purposes, from several references:

- 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 2020. 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.

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.*

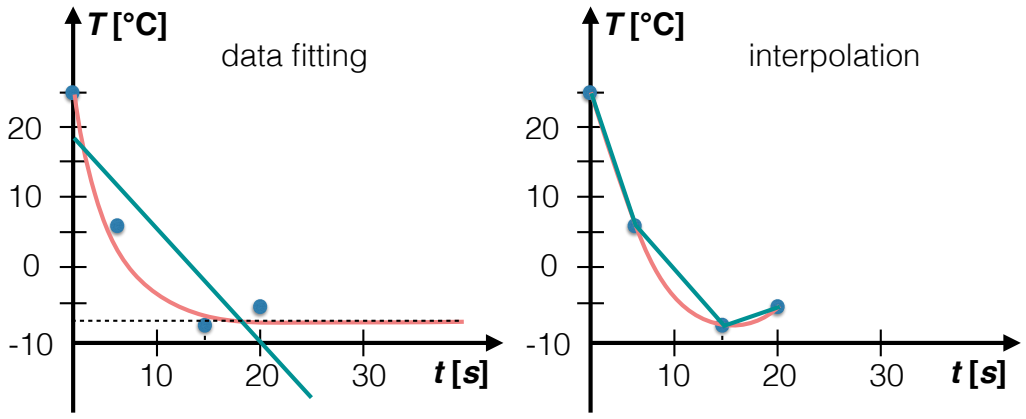


Figure 1: Example on shaking: we would like to know how long one should shake a cocktail. We insert a thermostat into the shaker and measure the temperature at some time instances during the shaking of the cocktail.

- **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)

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

We define the vectors $\boldsymbol{x} = (x_1, \ldots, x_N)$, $\boldsymbol{y} = (y_1, \ldots, y_N)$ and $\boldsymbol{e} = \big(y_1 - f(x_1), \ldots, y_N - f(x_N)\big)$. The vector $\boldsymbol{e}$ can be viewed as the error between the data and the evaluations of the fitted function.

The inner product between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ in $\mathbb{R}^N$ is defined as

$$\boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\top \boldsymbol{y} = \sum_{i=1}^{N} x_i \, y_i \, .$$

For a matrix $A \in \mathbb{R}^{N \times M}$ the null space or kernel is defined as the subspace of all vectors $\boldsymbol{x} \in \mathbb{R}^M$ such that $A\boldsymbol{x} = 0$ and is denoted by null(A). The left null space or cokernel is defined as the null space of $A^\top$. The column space or range or image of A is the space spanned by the columns of A and is denoted by range(A). The row space or coimage of A is the space spanned by the rows of A and is denoted by range$(A^\top)$. The rank of A is the number of linearly independent columns of A and is denoted as rank(A). It is true that rank(A) = rank$(A^\top)$.

In function fitting, we need to specify:

- **model architecture:** the functional form of function $f(x)$. 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 2-norm of the differences between the data and the function's values,

$$\|\boldsymbol{e}\|_2 = \sqrt{\sum_{i=1}^{N} e_i^2} = \sqrt{\sum_{i=1}^{N} \big(y_i - f(x_i)\big)^2}\,, \tag{1.1}$$

  or we can choose to minimize the 1-norm

$$\|\boldsymbol{e}\|_1 = \sum_{i=1}^{N} |e_i| = \sum_{i=1}^{N} \big|y_i - f(x_i)\big|\,, \tag{1.2}$$

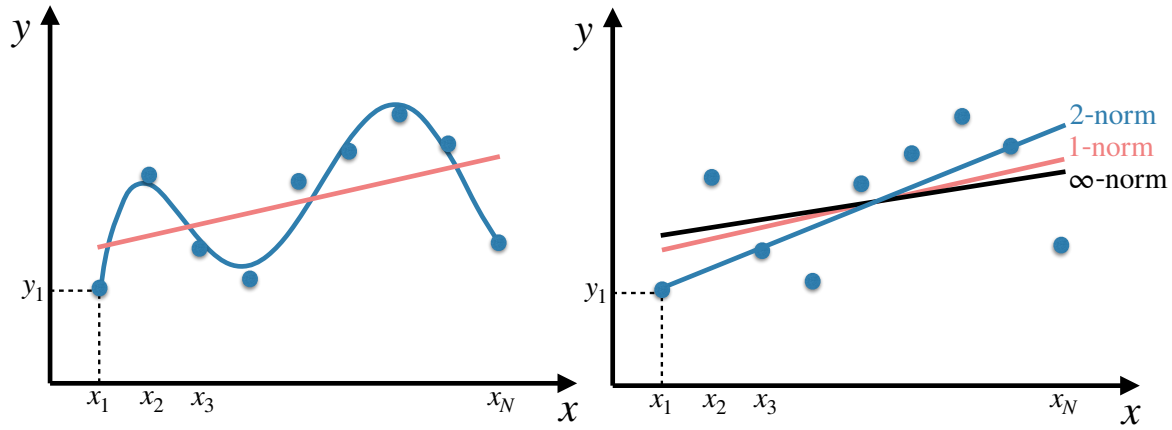


Figure 1.1: Example showing function fitting of given data points with two possible model architectures (left): straight and curved lines. In the right plot, we show three different linear approximations to the data obtained with different norm measures.

or we can choose to minimize the ∞-norm

$$\|\boldsymbol{e}\|_\infty = \max_{i=1,\ldots,N} |e_i| = \max_{i=1,\ldots,N} \left|y_i - f(x_i)\right| . \tag{1.3}$$

In general we can optimize the p-norm of the error

$$\|\boldsymbol{e}\|_p = \left(\sum_{i=1}^{N} e_i^p\right)^{\frac{1}{p}} . \tag{1.4}$$

Notice that

$$\|\boldsymbol{e}\|_2^2 = \boldsymbol{e}\cdot\boldsymbol{e} .$$

Different choices of the norm will lead to different results as shown in Figure 1.1, where the same data points have 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", see Fig. 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 $\{(x_i, y_i)\}_{i=1}^{N}$ with a function $f(x)$ that can be expressed as a linear combination of M linearly independent functions $\varphi_k(x)$,

$$f(x; \boldsymbol{w}) = \sum_{k=1}^{M} w_k \, \varphi_k(x) \tag{1.5}$$

where $\boldsymbol{w} = (w_1, \ldots, w_M)$ are unknown weights to be found. The functions $\varphi_k(x)$ are called basis functions and typically $M \ll N$. Some examples of functions that can be used as basis functions φ_k are given below:

$$\varphi_k(x) = x^{k-1}, \qquad \varphi_k(x) = \cos[(k-1)x],$$
$$\varphi_k(x) = e^{\beta_k x}, \qquad \varphi_k(x) = 1 - \frac{|x - x_k|}{\delta},$$

where β_k and δ are predefined parameters. Note that the functions φ can be nonlinear. The linear in the "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 w_k enter linearly.* The functions φ should not be linearly dependent otherwise some parameters are redundant.

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

$$E(\boldsymbol{w}) = \|\boldsymbol{e}(\boldsymbol{w})\|_2^2 = \sum_{i=1}^{N} e_i^2(\boldsymbol{w}) = \sum_{i=1}^{N} \big(y_i - f(x_i; \boldsymbol{w})\big)^2 \tag{1.6}$$

is **minimised**,

$$\boldsymbol{w}^\star = \arg\min_{\boldsymbol{w}} E(\boldsymbol{w}) \tag{1.7}$$

"Least Squares" refers to the fact that the "best" fit is the one that minimizes the sum of squared 2-norm of the errors e_i. The error e_i is a distance between the observed value y_i and the fitted value $f(x_i)$ provided by the model. The error is also known as the residual and we will use the names interchangeably.

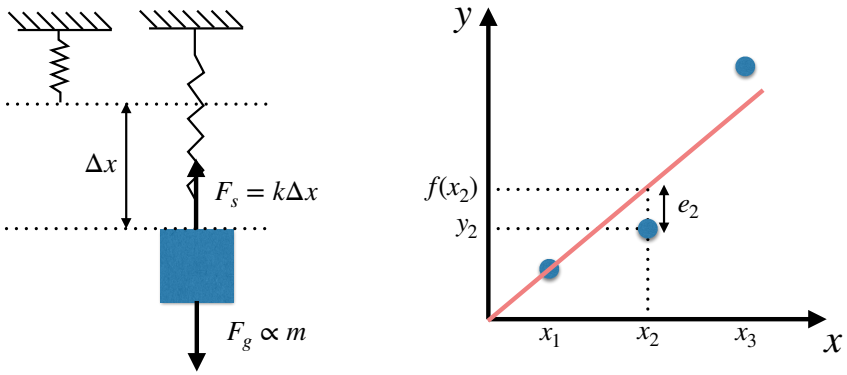
Example 1: A linear spring

We want to determine the elastic constant of a linear spring experimentally. To do so we take a set of measurements of the different displacements Δx observed when hanging different weights m on the spring. From Hook's law we know that the force F needed to extend or compress a spring by some distance Δx is proportional to that distance, i.e the behaviour of the spring can be expressed as $F = k\Delta x$. In our case the force F is the gravitational force exerted by the different masses.

To fit our formalism let us define our measurements of mass as x and our measurements of displacement as y. Using our knowledge of Hook's law we would like to find a function

$f(x)$ that fits best the data y where f is expressed as $f(x) = w\varphi(x)$, and $\varphi(x) = x$. Finding the weight w via least squares fit would give us the best approximation of the elastic constant k (in a least squares sense).

Say we repeat the experiment 3 times we have the dataset: (x_1, y_1), (x_2, y_2), (x_3, y_3) . In this case $M = 1$ and $N = 3$.



We have the following equations:

$$\begin{aligned} kx_1 &= y_1 \\ kx_2 &= y_2 \\ kx_3 &= y_3 \end{aligned}$$

In order to solve this problem, we seek to find a solution to Eq. (1.7) by minimizing

$$E(k) = (y_1 - kx_1)^2 + (y_2 - kx_2)^2 + (y_3 - kx_3)^2$$

$$\frac{dE}{dk} = 0 \Longrightarrow -2\left[(y_1 - kx_1)x_1 + (y_2 - kx_2)x_2 + (y_3 - kx_3)x_3\right] = 0$$

$$k^\star = \frac{x_1y_1 + x_2y_2 + x_3y_3}{x_1^2 + x_2^2 + x_3^2},$$

where we have denoted with $k^\star$ 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 $\boldsymbol{w}$ such that

$$A\boldsymbol{w} = \boldsymbol{y}\,, \tag{1.8}$$

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

$$\overbrace{\begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \dots & \varphi_M(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \dots & \varphi_M(x_2) \\ \vdots & & & \vdots \\ \vdots & & & \vdots \\ \varphi_1(x_N) & \varphi_2(x_N) & \dots & \varphi_M(x_N) \end{bmatrix}}^{A} \overbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_M \end{bmatrix}}^{\boldsymbol{w}} = \overbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_N \end{bmatrix}}^{\boldsymbol{y}} \tag{1.9}$$

We will refer to A as the *least squares matrix* or as the *regression matrix*. To determine $\boldsymbol{w}$ we need to solve the above system of equations. We distinguish between different cases:

M=N The matrix A is square. Since the functions $\varphi_k(x)$ are linearly independent there exists a unique solution given by $\boldsymbol{w} = A^{-1}\boldsymbol{y}$. 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 $A^{-1}A = AA^{-1} = I \Leftrightarrow \det A \neq 0 \Leftrightarrow \operatorname{rank}(A) = N \Leftrightarrow \operatorname{range}(A) = \mathbb{R}^N \Leftrightarrow \operatorname{null}(A) = 0$. Numerical methods to solve such systems are: Gauss elimination, pivoting strategies, LU decomposition, Cholesky decomposition, etc.

M>N The system is underdetermined and has infinitely many solutions.

M<N The system is overdetermined. We can seek an approximate solution $A\boldsymbol{w} \approx \boldsymbol{y}$ with least squares method by requiring that $E(\boldsymbol{w}) = \|\boldsymbol{y} - A\boldsymbol{w}\|_2^2$ is minimal.

Using the matrix notation, the error in Eq. (1.1) can be written as $\boldsymbol{e} = \boldsymbol{y} - A\boldsymbol{w}$. Evaluating the error function Eq. (1.6), we have

$$\begin{aligned} E(\boldsymbol{w}) = \boldsymbol{e}(\boldsymbol{w})^\top \boldsymbol{e}(\boldsymbol{w}) = \\ =&(\boldsymbol{y} - A\boldsymbol{w})^\top(\boldsymbol{y} - A\boldsymbol{w}) \\ =&(\boldsymbol{y}^\top - \boldsymbol{w}^\top A^\top)(\boldsymbol{y} - A\boldsymbol{w}) \\ =&\boldsymbol{y}^\top\boldsymbol{y} - \boldsymbol{y}^\top A\boldsymbol{w} - \boldsymbol{w}^\top A^\top \boldsymbol{y} + \boldsymbol{w}^\top A^\top A\boldsymbol{w} \\ =&\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{w}^\top A^\top \boldsymbol{y} + \boldsymbol{w}^\top A^\top A\boldsymbol{w}\,. \end{aligned}$$

In the last line we have used the fact that $\boldsymbol{y}^\top A\boldsymbol{w}$ 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 $\boldsymbol{w}$[1],

$$\frac{dE(\boldsymbol{w})}{d\boldsymbol{w}} = 0 \Longrightarrow -2\boldsymbol{y}^\top A + 2\boldsymbol{w}^\top A^\top A = 0 \quad \text{transpose} \quad -2A^\top\boldsymbol{y} + 2A^\top A\boldsymbol{w} = 0$$

and obtain the **normal equations**

$$\boxed{A^\top A\,\boldsymbol{w} = A^\top \boldsymbol{y}} \tag{1.10}$$

[1]In order to review matrix differentiation see [1].

Since we assume that the functions φ_k are linearly independent the matrix $A^\top A$ is a symmetric and positive definite matrix. For the symmetry notice that $(AA^\top)^\top = (A^\top)^\top A^\top = AA^\top$. For the positive definiteness we write $\boldsymbol{y}^\top A^\top A\boldsymbol{y} = (A\boldsymbol{y})^\top(A\boldsymbol{y}) = \|A\boldsymbol{y}\|_2^2 > 0$. Thus, $A^\top A$ can be inverted and we can write the least squares solution $\boldsymbol{w}^\star$ as

$$\boxed{\boldsymbol{w}^\star = (A^\top A)^{-1}A^\top \boldsymbol{y}} \tag{1.11}$$

Note that the normal equations transform the initial problem into a linear system with a square matrix.

$$\underset{(N\times M)}{A}\ \underset{(M\times 1)}{\boldsymbol{w}} = \underset{(N\times 1)}{\boldsymbol{y}} \quad\Longrightarrow\quad \underset{(M\times M)}{A^\top A}\ \underset{(M\times 1)}{\boldsymbol{w}} = \underset{(M\times 1)}{A^\top \boldsymbol{y}}$$

In order to see that the solution is indeed a minimizer of E, this we calculate the second derivative of E,

$$\nabla^2 E(\boldsymbol{w}) = 2A^\top A\,,$$

which is symmetric and positive definite for all $\boldsymbol{w}$. We conclude that the least square solution $\boldsymbol{w}^\star$ is a minimum.

Example 2: A classic: Fitting a line

Consider a set of N experimental results $\{(x_i, y_i)\}_{i=1}^N$, which we wish to fit to

$$w_1 + w_2 x_i \approx y_i$$

According to the notation in Eq. (1.5), we use the two first basis functions of the polynomial basis $\varphi_k(x_i) = x_i^{k-1}$. We can rewrite the problem in matrix form as

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_N \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \approx \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} \tag{1.12}$$

According to Eq. (1.10), we then need to solve the normal equations

$$\begin{bmatrix} N & \sum_{i=1}^N x_i \\ \sum_{i=1}^N x_i & \sum_{i=1}^N x_i^2 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N x_i y_i \end{bmatrix} \tag{1.13}$$

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

$$
\begin{aligned}
w_1^\star &= \frac{\left(\sum_{i=1}^N x_i^2\right)\left(\sum_{i=1}^N y_i\right) - \left(\sum_{i=1}^N x_i\right)\left(\sum_{i=1}^N x_i y_i\right)}{N\left(\sum_{i=1}^N x_i^2\right) - \left(\sum_{i=1}^N x_i\right)^2} \\
w_2^\star &= \frac{N\left(\sum_{i=1}^N x_i y_i\right) - \left(\sum_{i=1}^N x_i\right)\left(\sum_{i=1}^N y_i\right)}{N\left(\sum_{i=1}^N x_i^2\right) - \left(\sum_{i=1}^N x_i\right)^2}.
\end{aligned}
\tag{1.14}
$$

1.4 Special case: the orthonormal case

We denote by $\boldsymbol{a}_i$ the columns of a matrix $A \in \mathbb{R}^{N\times M}$. The columns of a matrix **orthonormal** whenever

$$\boldsymbol{a}_i \cdot \boldsymbol{a}_j = \delta_{ij}\,, \quad 1 \le i,j \le M\,,$$

where δ_{ij} is the delta Kronecker function. A square matrix with orthonormal columns is called an **orthogonal matrix** and has the following properties,

$$
\begin{aligned}
A^\top &= A^{-1}\,, \\
A^\top A &= AA^\top = I\,, \\
\|A\boldsymbol{u}\| &= \|\boldsymbol{u}\|\,, \\
(A\boldsymbol{u})^\top(A\boldsymbol{v}) &= \boldsymbol{u}^\top\boldsymbol{v}\,, \quad \forall \boldsymbol{u},\boldsymbol{v} \in \mathbb{R}^M\,.
\end{aligned}
$$

If the regression matrix is orthogonal then from Eq. (1.11) the optimal is given by

$$\boldsymbol{w}^\star = A^\top \boldsymbol{y}\,.$$

Example 3: A matrix with orthogonal columns

Fitting a straight line ($y = w_1 + w_2 x$) leads to orthogonal (but not orthonormal) columns when the measurements x_i average to zero. We can always do this transformation: $\tilde{x} = (x_1 + \ldots + x_N)/N$. Instead of working with $y = w_1 + w_2 x$, we work with $y = w_1 + w_2(x - \tilde{x})$.

$$
\begin{bmatrix} 1 & x_1 - \tilde{x} \\ \vdots & \vdots \\ 1 & x_N - \tilde{x} \end{bmatrix}
\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}
=
\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix}
$$

If we insert this in Eq. (1.13), we observe that we obtain a diagonal matrix on the left hand side

$$
\begin{bmatrix} N & 0 \\ 0 & \sum_{i=1}^N (x_i - \tilde{x})^2 \end{bmatrix}
\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}
=
\begin{bmatrix} \sum_{i=1}^N y_i \\ \sum_{i=1}^N (x_i - \tilde{x}) y_i \end{bmatrix}.
$$

We can easily solve the system to get

$$w_1^\star = \frac{\boldsymbol{a}a_1^\top \boldsymbol{y}}{\boldsymbol{a}_1^\top \boldsymbol{a}_1} = \frac{\sum y_i}{N}, \quad w_2^\star = \frac{\boldsymbol{a}_2^\top \boldsymbol{y}}{\boldsymbol{a}_2^\top \boldsymbol{a}_2} = \frac{\sum (x_i - \tilde{x}) y_i}{\sum (x_i - \tilde{x})^2},$$

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.

1.5 Geometric interpretation

Example 4: A linear spring: continued

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

$$k \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix},$$

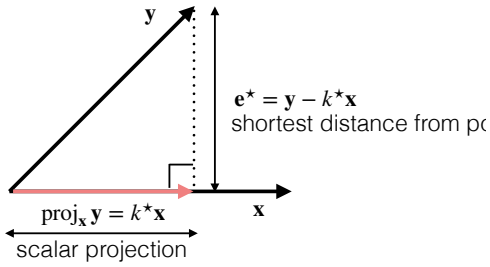
and the solution is

$$k^\star = \frac{x_1 y_1 + x_2 y_2 + x_3 y_3}{x_1^2 + x_2^2 + x_3^2} = \frac{\boldsymbol{x} \cdot \boldsymbol{y}}{\|\boldsymbol{x}\|_2^2},$$

which we rewrote using $\boldsymbol{x} = (x_1, x_2, x_3)$ and $\boldsymbol{y} = (y_1, y_2, y_3)$. From geometry we know that a projection of vector $\boldsymbol{b}$ onto vector $\boldsymbol{a}$ is given by

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

where $\frac{\boldsymbol{a} \cdot \boldsymbol{b}}{\|\boldsymbol{a}\|}$ 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 $k^\star$ such that $\boldsymbol{p}^\star = k^\star \boldsymbol{x}$ is a projection of $\boldsymbol{y}$ onto $\boldsymbol{x}$, which is the point on the line through $\boldsymbol{x}$ closest to $\boldsymbol{y}$. Therefore, the residual $\boldsymbol{e}^\star = \boldsymbol{y} - k^\star \boldsymbol{x}$ is perpendicular to vector $\boldsymbol{x}$.

The geometrical properties of least squares solutions that we have observed in example 4 are true in general. *Since the basis functions $\varphi(x)$ 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 $\boldsymbol{w}^\star$ such that $A\boldsymbol{w}^\star$ is the projection of vector $\boldsymbol{y}$ on the column space of A. The residual or error $\boldsymbol{e} = \boldsymbol{y} - A\boldsymbol{w}^\star$ is perpendicular to that space and has minimal 2-norm, which means that $\boldsymbol{e} = \|\boldsymbol{y} - A\boldsymbol{w}\|_2$ is minimal for $\boldsymbol{w} = \boldsymbol{w}^\star$.* Note that the equation $A\boldsymbol{w} = \boldsymbol{y}$ has a solution only if $\boldsymbol{y}$ is in the column space of A in which case $\boldsymbol{e} = 0$.

To prove that $\boldsymbol{e}^\star$ is perpendicular to the column space of A we write,

$$\begin{aligned} A^\top A\boldsymbol{w}^\star &= A^\top \boldsymbol{y} \\ A^\top(\boldsymbol{y} - A\boldsymbol{w}^\star) &= 0 \\ A^\top \boldsymbol{e}^\star &= 0 \end{aligned}$$

In other words, $\boldsymbol{e}^\star$ is an element of the left null space of A or the null space of $A^\top$.

Suppose that the fitting function has 2 basis functions $M = 2$ and we have $N = 3$ data points. Then the geometric interpretation can be visualised (Figure 1.2) in 3D space, where the 2 columns of matrix A span a plane.

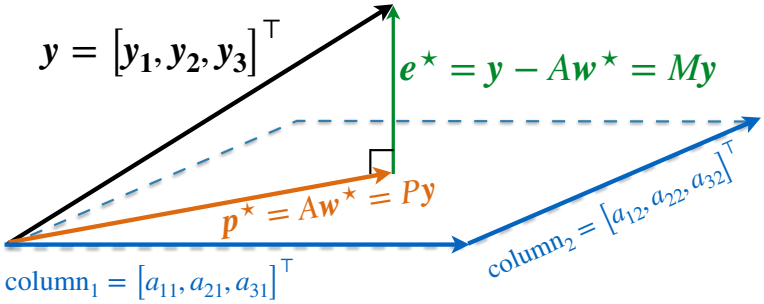


Figure 1.2: Geometric interpretation of the least squares solution with $N = 3$ and $M = 2$.

For an inconsistent system of equations, the vector $\boldsymbol{y}$ lies outside of that plane and the least squares method finds the vector $\boldsymbol{p}^\star = A\boldsymbol{w}^\star$ in the column space of A that is closest in the 2-norm to the data $\boldsymbol{y}$. Searching for the least squares solution $\boldsymbol{w}^\star$, which will minimise the error $\boldsymbol{e}$, is the same as finding the point $\boldsymbol{p}^\star = A\boldsymbol{w}^\star$ that is closer to $\boldsymbol{y}$ than any other point $\boldsymbol{p}$ in the column space.

1.5.1 The Projection Matrix P

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

$$\boldsymbol{p}^\star = A\boldsymbol{w}^\star = A(A^\top A)^{-1}A^\top \boldsymbol{y} = P\boldsymbol{y},$$

where matrix P given by

$$P = A\,(A^\top A)^{-1}A^\top \tag{1.15}$$

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

1. It is symmetric: $P = P^\top$,
2. It is idemptotent: $P^2 = P$.

On the other hand, for any $\boldsymbol{y} \in \mathbb{R}^N$ we can write the residual as

$$\begin{aligned}\boldsymbol{e}^\star &= \boldsymbol{y} - A\boldsymbol{w}^\star \\ &= \boldsymbol{y} - P\boldsymbol{y} \\ &= \boldsymbol{y} - A\,(A^\top A)^{-1}A^\top \boldsymbol{y} \\ &= (I - A\,(A^\top A)^{-1}A^\top)\boldsymbol{y} = M\boldsymbol{y},\end{aligned}$$

where

$$M = I - A\,(A^\top A)^{-1}A^\top\,,$$

which is also a projection matrix ($M = M^\top$ 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. The matrix M is the projection onto the space orthogonal to the column space of A, i.e. the left null space of A.

A fundamental theorem in linear algebra states that when A is a non singular matrix in $\mathbb{R}^{N\times M}$, then every $\boldsymbol{y} \in \mathbb{R}^N$ has a unique decomposition $\boldsymbol{y} = \boldsymbol{y}_r + \boldsymbol{y}_n$ where $\boldsymbol{y}_r$ is an element of the range of A and $\boldsymbol{y}_n$ is an element of the null space of $A^\top$. Observe that

$$\boldsymbol{y} = (P + M)\boldsymbol{y} = A\boldsymbol{w}^\star + \boldsymbol{e}^\star\,,$$

and $A\boldsymbol{w}^\star$ belongs in the range of A and $\boldsymbol{e}^\star$ belongs in the null space of $A^\top$.

1.6 Numerical solutions

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

1.6.1 Condition number

The numbers in a computer are stored with limited digits. Thus, the numerical solution will be affected by the round-off errors. Suppose we have obtained a numerical solution $\tilde{\boldsymbol{w}}$ of $A\boldsymbol{w} = \boldsymbol{y}$, where $\boldsymbol{w}$ is the true solution. We assume A is square and invertible. We define the two errors, the error in the numerical solution $\delta\boldsymbol{w} = \boldsymbol{w} - \tilde{\boldsymbol{w}}$ and the residual of the numerical solution $\delta\boldsymbol{y} = \boldsymbol{y} - \tilde{\boldsymbol{y}}$, where $\tilde{\boldsymbol{y}} = A\tilde{\boldsymbol{w}}$. Notice that since $\delta\boldsymbol{w} \neq 0$, the residual $\delta\boldsymbol{y}$ is not equal to zero as well.

If $\|\delta \boldsymbol{y}\|$ is small, does this mean that $\|\delta \boldsymbol{w}\|$ is also small?

The answer in general is **not necessarily**. To show this we will link the two quantities, the relative difference of the solution $\frac{\|\delta \boldsymbol{w}\|}{\|\boldsymbol{w}\|}$ with the relative residual $\frac{\|\delta \boldsymbol{y}\|}{\|\boldsymbol{y}\|}$. The residual can be written as,

$$\delta \boldsymbol{y} = \boldsymbol{y} - A\tilde{\boldsymbol{w}} = A\boldsymbol{w} - A\tilde{\boldsymbol{w}} = A\,\delta \boldsymbol{w}\,. \tag{1.16}$$

Since we have assumed that A is invertible, we can write

$$\delta \boldsymbol{w} = A^{-1}\delta \boldsymbol{y}\,. \tag{1.17}$$

Taking norms in the above equation and using properties of the norm,

$$\|\delta \boldsymbol{w}\| = \|A^{-1}\,\delta \boldsymbol{y}\| \leq \|A^{-1}\|\|\delta \boldsymbol{y}\|\,. \tag{1.18}$$

Notice also that since $A\boldsymbol{w} = \boldsymbol{y}$, it holds that $\|\boldsymbol{y}\| \leq \|A\|\|\boldsymbol{w}\|$, or equivalently,

$$\frac{1}{\|\boldsymbol{w}\|} \leq \frac{\|A\|}{\|\boldsymbol{y}\|}\,. \tag{1.19}$$

Multiplying Eq. (1.18) and Eq. (1.19) we get the desired result,

$$\frac{\|\delta \boldsymbol{w}\|}{\|\boldsymbol{w}\|} \leq \|A^{-1}\|\|A\|\frac{\|\delta \boldsymbol{y}\|}{\|\boldsymbol{y}\|} = \kappa(A)\frac{\|\delta \boldsymbol{y}\|}{\|\boldsymbol{y}\|}\,. \tag{1.20}$$

We define the **condition number** of a matrix A as

$$\kappa(A) = \|A\|\|A^{-1}\| \tag{1.21}$$

The above derivation and definition is true for any norm. Here we will only study the case of the 2-norm and we will denote the condition number as κ_2. 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. When $\kappa(A)$ is small, the computed solution will be close to the true solution. Finally, the condition number can be interpreted as a measure of how close a matrix is to be singular.

Next, we calculate the condition number in two special cases. If A is orthogonal then it true that $\|A^{-1}\|_2 = 1$, $\|A\|_2 = 1$ and $\kappa_2(A) = 1$. Orthogonal matrices have the best possible condition number.

The 2-norm of a matrix A is equal to $\|A\|_2 = \sqrt{\rho(A^\top A)}$, where ρ is the spectral radius or the largest eigenvalue, see Section 1.6.4. For positive definite matrices $\|A\|_2 = \sigma_M$ is the largest singular value of A. It is easy to see that for positive definite matrices $\kappa_2(A) = \frac{\sigma_1}{\sigma_M}$.

If we now return to the problem of least squares. The condition number for the matrix $A^\top A$ is given by,

$$\kappa_2(A^\top A) = \kappa_2(A)^2\,. \tag{1.22}$$

The proof of this statement needs the singular value decomposition theorem presented in Section 1.6.3 and is postponed for the end of that section. The equality in Eq. (1.22) is only true approximately if we consider other norms, but the approximation is tight.

We would like to find algorithms such that the error is bounded by $\kappa(A)$ and not its square. Next, we present two numerically stable ways to compute the LLS solution. When the regression matrix A has full column rank, the QR decomposition can be applied. When A is rank deficient, the SVD algorithm can be used.

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

$$A = QR = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R_1 \\ 0 \end{bmatrix} = Q_1 R_1 \tag{1.23}$$

where $Q \in \mathbb{R}^{N\times N}$ is an orthogonal matrix, $Q_1 \in \mathbb{R}^{N\times M}$ 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

$$\boldsymbol{w} = R_1^{-1} Q_1^\top \boldsymbol{y}. \tag{1.24}$$

Notice that R_1 is upper triangular we don't actually have to compute its inverse since the solution can be obtained by *back substitution*.

Notice that Eq. (1.24) can be written as $R_1\boldsymbol{w} = Q_1^\top \boldsymbol{y}$ and the condition number of R_1 is given by

$$\|R_1\| = \|Q_1 R_1\| = \|A\|,$$

where the first equality is true because Q_1 is orthogonal. Therefore, the condition of the matrices involved in the solution is the same as the condition of the initial matrix A.

1.6.3 Singular Value Decomposition (SVD)

The singular value decomposition of a matrix $A \in \mathbb{R}^{N\times M}$ with $\text{rank}(A) = \rho$, decomposes A as,

$$A = \begin{bmatrix} U_r & U_n \end{bmatrix} \begin{bmatrix} S & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_r^\top \\ V_n^\top \end{bmatrix} = U\,\Sigma\,V^\top \tag{1.25}$$

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 and $S = \text{diag}(\sigma_1, \ldots, \sigma_\rho)$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho \geq 0$. The numbers σ_i are called *singular values* of A.

The column vectors of U_r and V_r provide a basis for range(A) and range$(A^\top)$, respectively. The column vectors of U_n and V_n provide a basis for null$(A^\top)$ and null(A), respectively. The singular values σ_i give the lengths of the principal axes of the hyper-ellipsoid defined by $A\boldsymbol{w}$, when $\boldsymbol{w}$ lies on the hypersphere $\|\boldsymbol{w}\|^2 = 1$.

Denote by Σ^+ the matrix

$$\Sigma^+ = \begin{bmatrix} S^{-1} & 0 \\ 0 & 0 \end{bmatrix}.$$

We define the **Moore-Penrose pseudoinverse** of A as

$$A^+ = (A^\top A)^{-1} A^\top . \tag{1.26}$$

Using Eq. (1.25) it is easy to see that

$$\boxed{A^+ = V\,\Sigma^+\,U^\top}$$

The LLS solution using the SVD decomposition is given by

$$\boxed{\boldsymbol{w}^\star = V\,\Sigma^+\,U^\top \boldsymbol{y}} \tag{1.27}$$

Equation (1.27) can be written as $\Sigma V^\top \boldsymbol{w}^\star = U^\top \boldsymbol{y}$ and it easy to see that $\kappa(\Sigma V^\top) = \kappa(A)$.

1.6.4 *Norm of a matrix

The norm of a matrix is a map from $\mathbb{R}^{N\times M} \to \mathbb{R}$. The matrix norm satisfies the definition of a norm:

1. $\|A + B\| \leq \|A\| + \|B\|$ for all $A, B \in \mathbb{R}^{N\times M}$,
2. $\|aA\| = |a|\|A\|$ for all $a \in \mathbb{R}$ and $A \in \mathbb{R}^{N\times M}$,
3. $\|A\| = 0 \Leftrightarrow A = 0$ for all $\mathbb{R}^{N\times M}$.

Additionally, for square matrices it holds that,

$$\|AB\| \leq \|A\|\|B\| \text{ for all } A, B \in \mathbb{R}^{N\times N} \tag{1.28}$$

Next, we consider a special case of matrix norms, those that are **induced by a vector norm**. For any matrix $A \in \mathbb{R}^{N\times M}$ we define the matrix norm induced by the vector norm $\|\cdot\|_p$ as,

$$\|A\|_p = \sup_{\|\boldsymbol{x}\|_p \neq 0} \frac{\|A\boldsymbol{x}\|_p}{\|\boldsymbol{x}\|_p} = \sup_{\|\boldsymbol{x}\|_p = 1} \|A\boldsymbol{x}\|_p . \tag{1.29}$$

Here, we will only use the 2−norm of a matrix $A \in \mathbb{R}^{N\times M}$,

$$\|A\|_2 = \sup_{\|\boldsymbol{x}\|_2=1} \|A\boldsymbol{x}\|_2 \,. \tag{1.30}$$

In order to compute the 2−norm of a matrix $A \in \mathbb{R}^{N\times M}$ we consider the singular value decomposition $A = U\Sigma V^\top$ introduced in Section 1.6.3. Using the fact that $U \in \mathbb{R}^{N\times N}$ is an orthogonal matrix we find that,

$$\|A\|_2 = \sup_{\|\boldsymbol{x}\|_2=1} \sqrt{\boldsymbol{x}^\top A^\top A\boldsymbol{x}} = \sup_{\|\boldsymbol{x}\|_2=1} \sqrt{\boldsymbol{x}^\top V\Sigma^\top U^\top U\Sigma V^\top \boldsymbol{x}} \tag{1.31}$$

$$= \sup_{\|\boldsymbol{x}\|_2=1} \sqrt{\boldsymbol{x}^\top V\Sigma^\top\Sigma V^\top \boldsymbol{x}} \,. \tag{1.32}$$

Since $V \in \mathbb{R}^{M\times M}$ is also orthogonal we define $\boldsymbol{y} = V^\top\boldsymbol{x}$ which again has norm equal to 1. Thus, we can compute the 2−norm as

$$\|A\|_2 = \sup_{\|\boldsymbol{y}\|_2=1} \sqrt{\boldsymbol{y}^\top\Sigma^\top\Sigma\boldsymbol{y}} \tag{1.33}$$

$$= \sup_{\|\boldsymbol{y}\|_2=1} \|\Sigma\boldsymbol{y}\|_2 \tag{1.34}$$

$$= \sup_{\|\boldsymbol{y}\|_2=1} \sqrt{\sum_{i=1}^{N} \sigma_i^2 y_i^2} \,. \tag{1.35}$$

Recall from Section 1.6.3 that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_\rho \geq 0$ and thus the supremum is reached for $\boldsymbol{y} = (1, 0, \ldots, 0)$ and is given by

$$\|A\|_2 = \sigma_1 \,. \tag{1.36}$$

Thus, the 2−norm of a matrix $A \in \mathbb{R}^{N\times M}$ is the largest singular value.

1.6.5 *The condition of $A^\top A$

In this section we will prove Eq. (1.22) using the SVD theorem. From Eq. (1.25),

$$A^\top A = V\Sigma^\top U^\top U\Sigma V^\top \,. \tag{1.37}$$

Since U is orthogonal

$$A^\top A = V\Sigma^\top\Sigma V^\top \,. \tag{1.38}$$

We use again the orthogonality of V to write

$$\|A^\top A\|_2 = \|V\Sigma^\top\Sigma V^\top\|_2 = \|\Sigma^\top\Sigma\|_2 = \frac{\sigma_1^2}{\sigma_M^2} = \|A\|_2^2 \,, \tag{1.39}$$

and Eq. (1.22) has been proven.

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.

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 $\sum_i \big(y_i - g(x_i)\big)^2$?

Solution: No, we do not. We minimise

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

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

You are given N data points (x_i, y_i, z_i), $i = 1 \dots N$ in $\mathbb{R}^3$, which were roughly located on a surface. This could mean that $z_i = a\,x_i + b\,y_i + c + \xi_i$, 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 1: Population growth

Let $N(t)$ be the number of people at time t. Let λ be the birth rate. A simplified model describing the population growth is given by the differential equation,

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

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

$$\frac{\mathrm{d}N(t)}{\mathrm{d}t} = \lambda N(t) + u\,. \tag{2.1}$$

The solution of the Eq. (2.1) is given by,

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

where N_0 is 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]),

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

We can reformulate this nonlinear equation into

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

A nonlinear equation $g(x) = h(x)$ can always be written as $g(x) - h(x) = f(x) = 0$. In this case, we transform the problem to finding a root (zero) of the function $f(x)$, i.e. find $x^\star$ such that

$$f(x^\star) = 0 \tag{2.2}$$

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

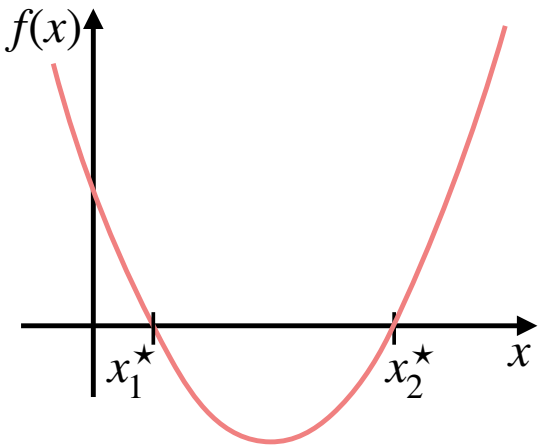


Figure 2.1: Function $f(x)$ is equal to zero at $x_1^\star$ and $x_2^\star$.

In general, there is no formula to find a root of a nonlinear function. The Abel-Ruffini theorem [2] states that there is no algebraic expression for the roots of a polynomial of degree 5 or higher. In order to solve nonlinear equations, we use **iterative schemes** that, given an initial guess x_0, generate a sequence $x_0, x_1, \ldots, x_k$ that converges to $x^\star$ as $k \to \infty$.

In practice, we only perform finite amount of steps and terminate the algorithm, when some stopping criteria are 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 $x^\star$ exactly.

2.2 Preliminaries

Before we discuss different algorithms for nonlinear equations, we need to consider the following:

- When does a solution to our problem exist theoretically?
- Is there a way 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 $A\vec{x} = \vec{b}$ required that A is non-singular. To determine this criterion for a Nonlinear equation, we look at the intermediate value theorem.

Theorem 1 (Intermediate Value Theorem)**.** *If a function f is continuous on interval $[a, b]$, i.e., $f \in C([a, b])$, then for all y between $f(a)$ and $f(b)$ there exists a number $x^\star \in (a, b)$ for which $f(x^\star) = y$.*

Corollary 1 (Bolzano's theorem)**.** *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 exists a $x^\star \in (a, b)$ with $f(x^\star) = 0$.*

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 $x^\star$ is the true solution. We want to answer the following question:

If $\left|f(\tilde{x})\right| \approx 0$, does this mean that $|\tilde{x} - x^\star| \approx 0$?

Notice that this question has been posed before in Section 1.6.1. We define a system (f) to be **well-conditioned** when a small change in the input (x) causes a small change in the output (y). We define a system to be **ill-conditioned** when a small change in the input causes a big change in the output.

We can measure the behaviour of a function f by the condition number,

$$\kappa = \frac{|\delta y|}{|\delta x|} = \frac{|f(x + \delta x) - f(x)|}{|\delta x|}, \tag{2.3}$$

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. We conclude that $\delta y \approx f'(x)\delta x$ and $\kappa = |f'(x)|$. Notice that this is the condition number of the forward evaluation problem, i.e., when given x we evaluate $y = f(x)$.

The root-finding problem is the reverse operation to the function evaluation. For a root $x^\star$ of a function $f(x)$, we need to evaluate $x^\star = f^{-1}(0)$, where f^{-1} is the inverse function of $f(x)$.

Using the previous result and the inverse function theorem, we find the condition number for the root finding problem is given by

$$\kappa = \left|\frac{\partial}{\partial y} f^{-1}(y)\right|\Big|_{y=0} = \frac{1}{\left|f'\left(f^{-1}(0)\right)\right|} = \frac{1}{|f'(x^\star)|} \tag{2.4}$$

We see that, for small $|f'(x^\star)|$ the root-finding problem is ill-conditioned. Small $|f'(x^\star)|$ means that the tangent line is nearly horizontal as illustrated in Fig. 2.2.

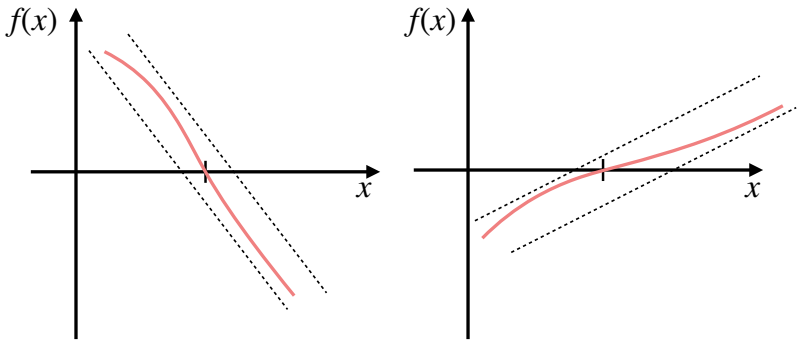


Figure 2.2: Illustration of a well-conditioned (left) and an ill-conditioned (right) function.

Note: If $f'(x^\star) = 0$, 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 Fig. 2.3 for examples).

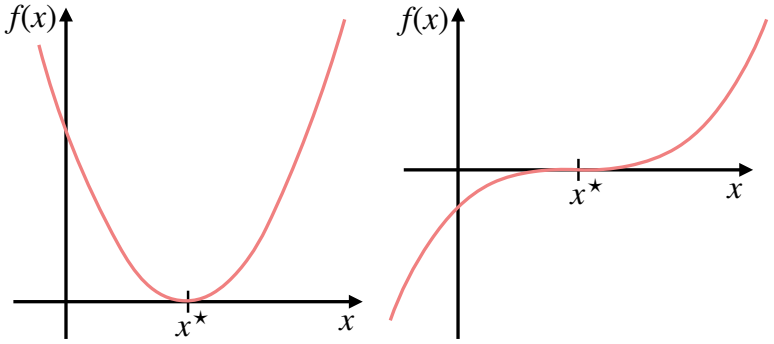


Figure 2.3: Graph of the function $f(x) = x^2 - 2x + 1 = (x-1)^2$ (left), which has a root $x^\star = 1$ with $m = 2$ and $f(x) = x^3 - 3x^2 + 3x - 1 = (x-1)^3$ (right), which has a root $x^\star = 1$ with $m = 3$.

Order of convergence

We want the sequence $\{x_k\}_{k=0}^{\infty}$ produced by the root-finding algorithm to converge to the root $x^\star$ as fast as possible. In order to quantify the speed of convergence, we define the error at the k-th

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

term x_k by

$$e_k = x_k - x^\star \,. \tag{2.5}$$

If the sequence of x_k converges to $x^\star$ as $k \to \infty$ then there exists $r \geq 1$ and $C > 0$, such that the following limit exists,

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^r} = C \tag{2.6}$$

The value r is the **order of convergence** and the constant C is the **rate of convergence** or **asymptotic error constant**. Common cases according to the value of r are:

- $r = 1$: if $0 < C < 1$ linear convergence. If $C = 0$ superlinear. If $C = 1$ sublinear.
- $r = 2$: quadratic convergence.

Example 2: Convergence Rates

1. $x_k = \frac{1}{2^k}$, $\lim_{k\to\infty} x_k = x^\star = 0$

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

2. $x_k = \frac{1}{k!}$, $\lim_{k\to\infty} x_k = x^\star = 0$

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

3. $x_k = \frac{1}{k^2}$, $\lim_{k\to\infty} x_k = x^\star = 0$

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

4. $x_k = \frac{1}{2^{2^k}}$, $\lim_{k\to\infty} x_k = x^\star = 0$

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

2.3 Bisection Method

The bisection method is based on the intermediate value theorem (Theorem 1). The root-finding method starts in an initial interval and proceeds with halving the interval length, until a solution has been isolated within a prescribed accuracy. The next interval is chosen such that the two

points always have opposite sign. The algorithm is illustrated in Fig. 2.4 and described in Algorithm 1.

Algorithm 1 Bisection Method

Input:

a, b, (initial interval)
tol, (tolerance, minimum length of interval)
$k_{\max}$, (maximum number of iterations)

Output:

x_k, (approximate solution after k iterations)

Steps:

```
k ← 1
while (b − a) > tol  and  k < k_max do
    x_k ← (a + b)/2
    if sign(f(a)) = sign(f(x_k)) then
        a ← x_k
    else
        b ← x_k
    end if
    k ← k + 1
end while
```

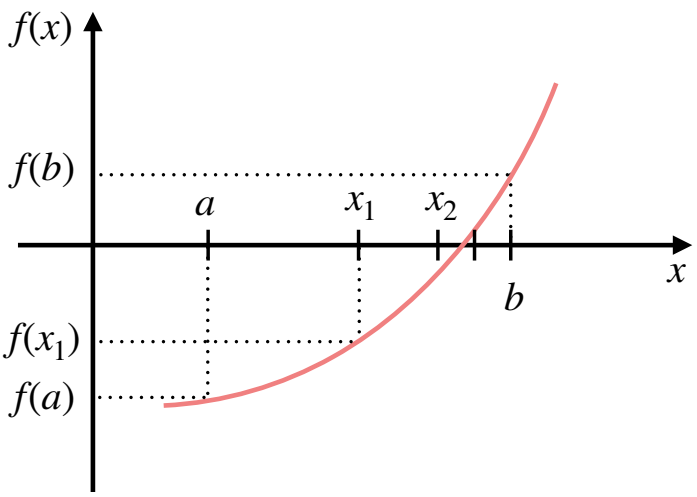


Figure 2.4: Iterative approach used in the bisection method.

Order of convergence

In order to show that the the bisection method converges to the solution, we first notice that,

$$|x_k - x_{k-1}| = \frac{1}{2}|x_{k-1} - x_{k-2}| = \cdots = \frac{1}{2^k}|x_1 - x_0| = \frac{1}{2^k}(b-a)\,. \tag{2.7}$$

The error at the iteration $k+1$ is bounded by,

$$|e_{k+1}| = |x_{k+1} - x^\star| \leq \frac{1}{2}|x_{k+1} - x_k| = \frac{1}{2^{k+1}}(b-a)\,. \tag{2.8}$$

In Fig. 2.5 a sketch shows why this inequality holds. As $k \to \infty$, $2^{-k-1}(b-a) \to 0$ and $|e_{k+1}| \to 0$. Therefore, $x_{k+1} \to x^\star$ and the algorithm converges to the root.

How fast is convergence achieved? Informally, from the ratio of Eq. (2.6),

$$\frac{|e_{k+1}|}{|e_k|^r} \approx \frac{2^{-k-1}(b-a)}{2^{-kr}(b-a)^r} = \frac{(b-a)^{1-r}}{2} 2^{k(r-1)}\,. \tag{2.9}$$

Notice that for $r \geq 1$,

$$\lim_{k\to\infty} 2^{k(r-1)} = \begin{cases} 1, & r = 1 \\ \infty, & r > 1\,. \end{cases} \tag{2.10}$$

Thus, the order of convergence is 1 and the rate of convergence is $1/2$.

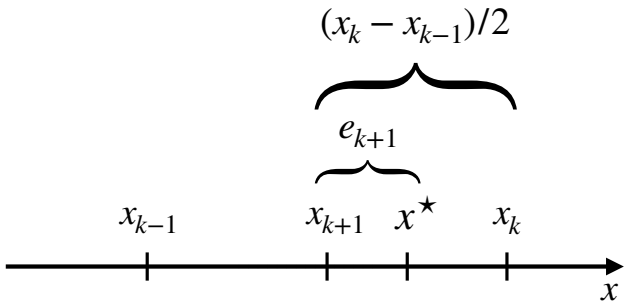


Figure 2.5: The error in the bisection method at $k+1$ step is bounded by half the length of the interval of the previous step.

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,

$$|e_k| = \text{tol} \quad \Rightarrow \quad \frac{b-a}{2^k} = \text{tol} \quad \Rightarrow \quad k = \log_2\left(\frac{b-a}{\text{tol}}\right). \tag{2.11}$$

Notice that the convergence result does not depend on the function f.

Advantages	Disadvantages
The bisection method is certain to converge.	The convergence is slow.
The bisection method makes no use of actual function values, only of their signs.	The initial interval needs to be known beforehand.
The function doesn't need to be differentiable, the only assumption on f is that it is continuous.	Can not be easily generalized to higher dimensions and many unknowns.

Table 2.1: Advantages and disadvantages of the bisection method.

2.4 Newton's Method

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

$$f(x) \approx f(x_k) + f'(x_k)(x - x_k)\,. \tag{2.12}$$

For $x = x^\star$ in the the above approximation we get,

$$0 = f(x^\star) \approx f(x_k) + f'(x_k)(x^\star - x_k)\,. \tag{2.13}$$

Solving Eq. (2.13) for $x^\star$ gives the approximation,

$$x^\star \approx x_k - \frac{f(x_k)}{f'(x_k)}. \tag{2.14}$$

The approximation is exact only for a linear function, since the approximation Eq. (2.13) is in this case exact. Nonetheless, we will use this result in order to update x_k. Given the initial guess x_0, the Newton's method generates a sequence $\{x_k\}_{k=0}^{\infty}$, where,

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}} \tag{2.15}$$

The algorithm is illustrated in Fig. 2.6. Newton's method approximates the nonlinear function f by the tangent line to $f(x_k)$ at x_k. A geometric construction of Newton's method can be made by observing that $f'(x_k) = \tan(\vartheta) = \frac{f(x_k)}{x_k - x_{k+1}}$.

Order of convergence

In order to estimate the error of Newton's method, we assume f contains a unique single root in the interval $[a, b]$. Moreover, we assume that f is twice differentiable on (a, b) and f' continuous

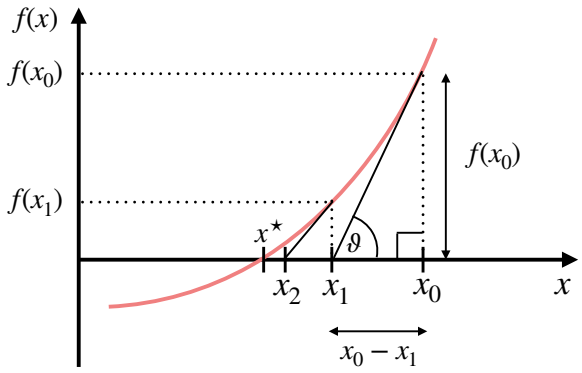


Figure 2.6: Graphical example of Newton iterations as we follow the tangents of $f(x)$.

on (a, b). Since $x^\star$ is a single root it is true that $f(x^\star) = 0$ and $f'(x^\star) \neq 0$. By Taylor's theorem, we expand f around x_k,

$$f(x) = f(x_k) + f'(x_k)(x - x_k) + \frac{1}{2} f''(\xi_k)(x - x_k)^2 \,, \tag{2.16}$$

for ξ_k between x and x_k. Evaluating Eq. (2.16) at $x = x^\star$, and using the fact that $f(x^\star) = 0$ we get,

$$0 = f(x_k) + f'(x_k)(x^\star - x_k) + \frac{1}{2} f''(\xi_k)(x^\star - x_k)^2 \,, \tag{2.17}$$

for ξ_k between $x^\star$ and x_k. Dividing by $f'(x_k)$ and rearranging the terms in Eq. (2.17)

$$0 = \frac{f(x_k)}{f'(x_k)} - x_k + x^\star + \frac{f''(\xi_k)}{2f'(x_k)}(x^\star - x_k)^2 \,. \tag{2.18}$$

Using Eq. (2.15) in Eq. (2.18),

$$0 = x^\star - x_{k+1} + \frac{f''(\xi_k)}{2f'(x_k)}(x^\star - x_k)^2 \,, \tag{2.19}$$

or equivalently

$$x_{k+1} - x^\star = \frac{f''(\xi_k)}{2f'(x_k)}(x^\star - x_k)^2 \,. \tag{2.20}$$

Finally, we take absolute values in the above equation and use the definition of the error in Eq. (2.5) to write,

$$|e_{k+1}| = \frac{f''(\xi_k)}{2f'(x_k)} |e_k|^2 \,. \tag{2.21}$$

In order to find the order of convergence, we write

$$\lim_{k\to\infty} \frac{|e_{k+1}|}{|e_k|^2} = \lim_{k\to\infty} \frac{|f''(\xi_k)|}{2|f'(x_k)|} = \frac{|f''(x^\star)|}{2|f'(x^\star)|} = C < \infty \,, \tag{2.22}$$

since ξ_k convergences to $x^\star$ as x_k convergences to $x^\star$. Thus the order of convergence is 2 and the algorithm converges quadratically. It can be shown that for roots of multiplicity $m > 1$, the convergence (if the method is converging) is linear unless we modify the iteration (Eq. (2.15)) to $x_{k+1} = x_k - mf(x_k)/f'(x_k)$.

Advantages	Disadvantages
Quadratic convergence	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.
	Requires the evaluation of both function and derivative at each iteration.

Table 2.2: Advantages and disadvantages of the Newton's method.

2.5 Secant Method

Evaluating the derivative in Newton's method 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. If we approximate $f'(x_k)$ by

$$f'(x_k) \approx \frac{f(x_k) - f(x_{k-1})}{x_k - x_{k-1}}\,, \tag{2.23}$$

then Newton's method is transformed to

$$x_{k+1} = x_k - f(x_k)\frac{x_k - x_{k-1}}{f(x_k) - f(x_{k-1})} \tag{2.24}$$

The secant method approximates the nonlinear function f by a secant line through the previous two iterations (see Fig. 2.7).

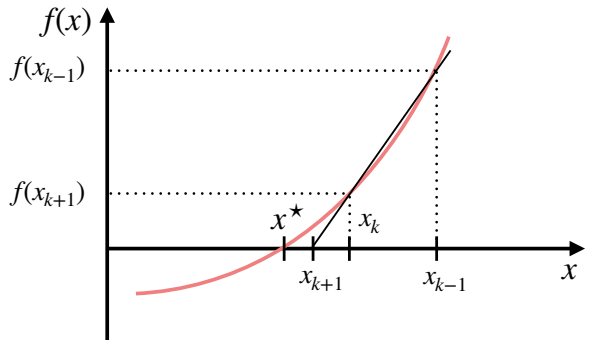


Figure 2.7: Iterative approach used in the secant method.

Order of convergence

It can be shown that the convergence rate of the secant method for simple roots is equal to the golden ratio,

$$\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618\,.$$

For the interested readers, more details about the convergence of the secant method can be found in [3]. The secant method uses a linear approximation of the function f to construct its derivative. This idea can be extended for higher order interpolations. For example, if quadratic interpolation is used (Müller's method) then the method has convergence rate approximately equal to 1.839.

Advantages	Disadvantages
We avoid the evaluation of the derivative.	The convergence rate is not quadratic.
At each step, we need to perform 1 function evaluation.	We need two initial approximations.

Table 2.3: Advantages and disadvantages of the secant method.

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

2.6 Numerical computation of the order of convergence

In this section, we describe the process one needs to follow in order to numerically compute the order of convergence of a root-finding method. From the definition of the order of convergence in Eq. (2.6), it is true that for large enough k,

$$\frac{|e_{k+1}|}{|e_k|^r} \approx C\,,$$

or equivalently, $|e_{k+1}| \approx C|e_k|^r$. It also holds that $|e_{k+2}| \approx C|e_{k+1}|^r$. The ratio of $|e_{k+2}|$ to $|e_{k+1}|$ is equal to,

$$\left|\frac{e_{k+2}}{e_{k+1}}\right| \approx \left|\frac{e_{k+1}}{e_k}\right|^r . \tag{2.25}$$

Notice that the convergence rate C does not appear in Eq. (2.25). By taking the log in Eq. (2.25), we get

$$\log\left|\frac{e_{k+2}}{e_{k+1}}\right| \approx r \log\left|\frac{e_{k+1}}{e_k}\right| .$$

Finally, the convergence rate is approximated by,

$$r \approx \frac{\log\left|\frac{e_{k+2}}{e_{k+1}}\right|}{\log\left|\frac{e_{k+1}}{e_k}\right|} \tag{2.26}$$

In practice, we compute the error $e_k = x_k - x^\star$ at every iteration k of the root-finding method, for a function with known root $x^\star$. Then, we use Eq. (2.26) to evaluate the convergence rate r_k as a sequence. The sequence of r_k will converge to the theoretical convergence rate of the method r.

Exam checklist

- What are the key algorithms for solving Nonlinear 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 the sequence: $x_0 = 0, x_1 = 1, x_2 = 0, x_3 = 1, ...$ Newton's method is stuck in a cycle. The cycle has a period of 2 and is called a 2-cycle.

3. $f(x) = x^2$ using initial value $x_0 = 1$.

   Solution:

   The first derivative is $f'(x) = 2x$, which is 0 at $x^\star$. 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. (2.22) 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 root with multiplicity 2.

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 functions $f_i(\boldsymbol{x})$, $i = 1, \ldots, N$, where $\boldsymbol{x} = (x_1, \ldots, x_M)^\top$ is a vector of M unknowns. We wish to find $\boldsymbol{x}^\star$, such that

$$f_i(\boldsymbol{x}^\star) = 0, \qquad i = 1, \ldots, N \tag{3.1}$$

We write this system of equations as

$$F(\boldsymbol{x}^\star) = \begin{pmatrix} f_1(\boldsymbol{x}^\star) \\ f_2(\boldsymbol{x}^\star) \\ \vdots \\ f_N(\boldsymbol{x}^\star) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{0} \tag{3.2}$$

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

Example 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:

$$p = k_1 e^{k_2 r} + k_3 r, \tag{3.3}$$

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:

$$\begin{aligned} p_1 &= k_1 e^{k_2 r_1} + k_3 r_1, \\ p_2 &= k_1 e^{k_2 r_2} + k_3 r_2, \\ p_3 &= k_1 e^{k_2 r_3} + k_3 r_3. \end{aligned} \tag{3.4}$$

As in the Chapter 2, we will solve this system with iterative algorithms that, given an initial guess $\boldsymbol{x_0}$, compute a sequence with terms $\boldsymbol{x_k}$ that hopefully converges to the true solution $\boldsymbol{x}^\star$. Note that if $N = M = 1$, our problem simplifies to the case discussed in the previous chapter. For $N = 1$ and M arbitrary we find a single non-linear equation for a function $f : \mathbb{R} \to \mathbb{R}$. For $M = 1$ and N arbitrary a set of $i = 1, \ldots, N$ non-linear equations $f_i : \mathbb{R} \to \mathbb{R}$.

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.

For $F : \mathbb{R}^M \to \mathbb{R}^N$ that has bounded derivatives up to second order, for every $\boldsymbol{x}$ and $\boldsymbol{y} \in \mathbb{R}^M$ it holds that

$$f_i(\boldsymbol{x} + \boldsymbol{y}) = f_i(\boldsymbol{x}) + \sum_{j=1}^{M} \frac{\partial f_i(\boldsymbol{x})}{\partial x_j} y_j + O\left(\|\boldsymbol{y}\|^2\right), \tag{3.5}$$

or

$$F(\boldsymbol{x} + \boldsymbol{y}) = F(\boldsymbol{x}) + J(\boldsymbol{x})\, \boldsymbol{y} + O\left(\|\boldsymbol{y}\|^2\right), \tag{3.6}$$

where $J(\boldsymbol{x}) \in \mathbb{R}^{N \times M}$ is the **Jacobian matrix** with elements $J_{ij}(\boldsymbol{x}) = \frac{\partial f_i(\boldsymbol{x})}{\partial x_j}$,

$$J(\boldsymbol{x}) = \begin{pmatrix} \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_1(\boldsymbol{x})}{\partial x_M} \\ \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_2(\boldsymbol{x})}{\partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \dfrac{\partial f_N(\boldsymbol{x})}{\partial x_1} & \dfrac{\partial f_N(\boldsymbol{x})}{\partial x_2} & \cdots & \dfrac{\partial f_N(\boldsymbol{x})}{\partial x_M} \end{pmatrix}. \tag{3.7}$$

Graphically, we can think of $\boldsymbol{x}$ as a point in $\mathbb{R}^M$ and $\boldsymbol{y}$ 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., $f(x+y) = f(x) + f'(x)y + O(y^2)$ 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 M unknowns, the condition number of the root finding problem for root $\boldsymbol{x}^\star$ of F is $\|J^{-1}(\boldsymbol{x}^\star)\|$ where J is the $N \times M$ Jacobian matrix.

3.2 Newton's method

As in the previous chapter, we derive the Newton's method by expanding F around $\boldsymbol{x}_k$ and evaluate the approximation at $x^\star$,

$$\mathbf{0} = F(\boldsymbol{x}^\star) \approx F(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)(\boldsymbol{x}^\star - \boldsymbol{x}_k), \tag{3.8}$$

where we have kept only two terms since we assume that $\|\boldsymbol{x}^\star - \boldsymbol{x}_k\|$ is small. Rewriting Eq. (3.8) and substituting $\boldsymbol{x}^\star$ by $\boldsymbol{x}_{k+1}$ gives

$$J(\boldsymbol{x}_k)(\boldsymbol{x}_{k+1} - \boldsymbol{x}_k) = -F(\boldsymbol{x}_k)\,. \tag{3.9}$$

Setting $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}$ we see that at each step of the method, we have to solve the linear system of equations,

$$A\boldsymbol{z} \;=\; \boldsymbol{b}. \tag{3.10}$$

Here the matrix $A = J(\boldsymbol{x}_k)$ and right hand side $\boldsymbol{b} = -F(\boldsymbol{x}_k)$.

3.2.1 Newton-Raphson method, $N = M$

When $M = N$, then J is a square matrix and, assuming it is invertible, the Newton or Newton-Rapshon method is given by,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J^{-1}(\boldsymbol{x}_k)\, F(\boldsymbol{x}_k) \tag{3.11}$$

In practice, we don't invert $J(\boldsymbol{x}_k)$. Instead we solve

$$J(\boldsymbol{x}_k)\, \boldsymbol{z} = -F(\boldsymbol{x}_k)\,, \tag{3.12}$$

for $\boldsymbol{z}$ and update as $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \boldsymbol{z}$ as in the pseudocode Algorithm 2. 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 $\boldsymbol{x}^\star$.

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

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. (3.12) costs $O(N^3)$ operations when 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 $\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_k\| < \text{tol}$
$k_{\max}$, maximum number of iterations: stop if $k > k_{\max}$

Output:

$\boldsymbol{x}_k$, approximation of solution of $F(\boldsymbol{x}) = \boldsymbol{0}$ within tolerance tol or with $k = k_{\max}$ steps.

Steps:

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

3.2.2 pseudo-Newton method, $M \neq N$

When $M \neq N$ and assuming that J has always full rank the Newton-Raphson method takes the form

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - J^{+}(\boldsymbol{x}_k)\, F(\boldsymbol{x}_k) \tag{3.13}$$

where J^{+} is the Moore-Penrose pseudo-inverse matrix. It has be shown that in this case the algorithm converges to stationary points of $\|F(\boldsymbol{x})\|_2^2$ [4]. Notice that when $M > N$ then $J^{+} = (J^{\top}J)^{-1}J^{\top}$ and when $M < N$ then $J^{+} = J^{\top}(JJ^{\top})^{-1}$.

Example 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 $\boldsymbol{x}^{\star}$. 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 $F(\boldsymbol{x}_k) + J(\boldsymbol{x}_k)(\boldsymbol{x} - \boldsymbol{x}_k) = 0$ 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 $f_x = \partial f(x_k, y_k)/\partial x$, $f_y = \partial f(x_k, y_k)/\partial y$

$$f_x(x - x_k) + f_y(y - y_k) + f(x_k, y_k) = 0$$
$$g_x(x - x_k) + g_y(y - y_k) + g(x_k, y_k) = 0$$

Each one of the two equations represents a plane. The first equation defines a plane tangent to surface $z = f(x, y)$ at point $\big(x_k, y_k, f(x_k, y_k)\big)$, while the second equation defines the tangent plane to $z = g(x, y)$. The intersection of these planes and $z = 0$ gives us the point (x_{k+1}, y_{k+1}), 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 in each iteration

3.3.1 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. (3.12), we solve

$$J_0 \, \boldsymbol{z} = -F(\boldsymbol{x}_k) \, . \tag{3.14}$$

This can be done efficiently by performing an LU decomposition of J_0 once and use it over and over again for different $F(\boldsymbol{x}_k)$. This removes the $O(N^3)$ cost of solving the linear system. We only have to evaluate $F(\boldsymbol{x}_k)$ and can then compute $\boldsymbol{z}$ in $O(N^2)$ operations. This method can only succeed if J is not changing rapidly.

3.3.2 *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. (3.7)). It instead updates $J_0 = J(\boldsymbol{x}_0)$ by using only evaluations of the function F.

After the first step we know: $\Delta\boldsymbol{x} = \boldsymbol{x}_1 - \boldsymbol{x}_0$ and $\Delta F = F(\boldsymbol{x}_1) - F(\boldsymbol{x}_0)$. So, derivatives of the f_i are in the direction of $\Delta\boldsymbol{x}$. Then the next J_1 is adjusted to satisfy

$$J_1 \Delta\boldsymbol{x} = \Delta F\,, \tag{3.15}$$

for example by the rank-1 update

$$J_1 = J_0 + \frac{(\Delta F - J_0\Delta\boldsymbol{x})\,\Delta\boldsymbol{x}^\top}{\Delta\boldsymbol{x}^\top\Delta\boldsymbol{x}}\,. \tag{3.16}$$

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? Consider the minimization problem

$$x^\star = \arg\min_{\boldsymbol{x}} E(\boldsymbol{x})\,, \tag{3.17}$$

where we have M unknowns $\boldsymbol{x} = (x_1, \ldots, x_M)^\top$ and function $E : \mathbb{R}^M \to \mathbb{R}$ is a scalar function of M variables. Note that the problem of maximizing $E(\boldsymbol{x})$ is equivalent to the problem of minimizing $-E(\boldsymbol{x})$.

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

$$\nabla E(\boldsymbol{x}^\star) = \left(\frac{\partial E}{\partial x_1}(\boldsymbol{x}^\star), \ldots, \frac{\partial E}{\partial x_M}(\boldsymbol{x}^\star)\right)^\top = \boldsymbol{0}\,, \tag{3.18}$$

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

$$\nabla^2 E(\boldsymbol{x}^\star) = H(\boldsymbol{x}^\star) = \begin{pmatrix} \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_1^2} & \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_1 \partial x_M} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_M \partial x_1} & \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_M \partial x_2} & \cdots & \frac{\partial^2 E(\boldsymbol{x}^\star)}{\partial x_M^2} \end{pmatrix}. \tag{3.19}$$

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

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

$$F(\boldsymbol{x}) = \nabla E(\boldsymbol{x}) = \mathbf{0}\,, \tag{3.20}$$

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

3.4.1 Newton's method

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

$$J(\boldsymbol{x}) = \nabla^2 E(\boldsymbol{x})\,. \tag{3.21}$$

The Newton's update rule is therefore,

$$\begin{aligned} \nabla^2 E(\boldsymbol{x}_k)\,\boldsymbol{z} &= -\nabla E(\boldsymbol{x}_k)\,, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \boldsymbol{z} \end{aligned} \tag{3.22}$$

3.4.2 *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 from the minimum it is unreliable and can diverge from the minimum. A simpler method is to move in the local gradient descent direction,

$$\begin{aligned} \boldsymbol{z} &= -\nabla E(\boldsymbol{x}_k)\,, \\ \boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \eta \boldsymbol{z}\,. \end{aligned}$$

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 which might be local and not a global minimum.

3.4.3 *Levenberg-Marquardt Method

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

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

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 $E(\boldsymbol{x}_k)$, λ is increased. The Levenberg-Marquardt method is primarily used for the non-linear Least Squares problem.

3.5 *Non-Linear Least Squares

In Chapter 1 we considered a linear least squares problem of function fitting where given a set of data $\{(x_i, y_i)\}_{i=1}^N$ we computed the M unknown parameters in $\boldsymbol{w}$ by solving the system $A\boldsymbol{w} = \boldsymbol{y}$. The fitting function $f(x)$ had parameters $\boldsymbol{w} = (w_1, \ldots, w_M)$ that appeared outside of the basis functions. Now, we consider a case where the unknown parameters appear inside the basis, functions, i.e.,

$$f(x) = \sum_{k=1}^{K} \varphi_k(x; \boldsymbol{w}) \tag{3.23}$$

Notice that now we may have more parameters than basis functions, $M > K$. The cost function can still be expressed as,

$$E(\boldsymbol{w}) = \sum_{i=1}^{N} \big(y_i - f(x_i; \boldsymbol{w})\big)^2 = \big\|\boldsymbol{y} - F(\boldsymbol{x}; \boldsymbol{w})\big\|_2^2 \,. \tag{3.24}$$

We wish to find the $\boldsymbol{w}^\star$ which minimizes $E : \mathbb{R}^M \to \mathbb{R}$.

Example 3: Gaussian basis functions

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

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

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

To employ **Newton's method** we need to evaluate the terms ∇E and $\nabla^2 E$ that appear in Eq. (3.22). The first term is equal to

$$\frac{\partial E}{\partial w_k}(\boldsymbol{w}) = -2 \sum_{i=1}^{N} \big(y_i - f(x_i; \boldsymbol{w})\big) \frac{\partial f(x_i; \boldsymbol{w})}{\partial w_k} \,. \tag{3.25}$$

The second term is

$$\frac{\partial^2 E}{\partial w_k \partial w_\ell}(\boldsymbol{w}) = -2\sum_{i=1}^{N}\left(-\frac{\partial f(x_i;\boldsymbol{w})}{\partial w_k}\frac{\partial f(x_i;\boldsymbol{w})}{\partial w_\ell} + \big(y_i - f(x_i;\boldsymbol{w})\big)\frac{\partial^2 f(x_i;\boldsymbol{w})}{\partial w_k \partial w_\ell}\right). \tag{3.26}$$

Define the vector $\boldsymbol{r}(\boldsymbol{w}) \in \mathbb{R}^N$ with elements $r_i = y_i - f(x_i;\boldsymbol{w})$ the matrix $D \in \mathbb{R}^{N\times M}$ with elements

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

and the matrix $L(\boldsymbol{w}) \in \mathbb{R}^{M\times M}$ with elements

$$L_{k,\ell}(\boldsymbol{w}) = \sum_{i=1}^{N}\big(y_i - f(x_i;\boldsymbol{w})\big)\frac{\partial^2 f(x_i;\boldsymbol{w})}{\partial w_k \partial w_\ell}.$$

Then we can write

$$\nabla E(\boldsymbol{w}) = -2D(\boldsymbol{w})^\top \boldsymbol{r}(\boldsymbol{w}),$$

and

$$\nabla^2 E(\boldsymbol{w}) = 2D(\boldsymbol{w})^\top D(\boldsymbol{w}) - 2L(\boldsymbol{w}),$$

Plugging these expressions into Eq. (3.22) gives us

$$\boxed{\begin{aligned}\Big(D(\boldsymbol{w}_k)^\top D(\boldsymbol{w}_k) - L(\boldsymbol{w}_k)\Big)\,\boldsymbol{z} &= D(\boldsymbol{w}_k)^\top \boldsymbol{r}(\boldsymbol{w}_k),\\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \boldsymbol{z}\end{aligned}} \tag{3.27}$$

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

$$\boxed{\begin{aligned}D(\boldsymbol{w}_k)^\top D(\boldsymbol{w}_k)\,\boldsymbol{z} &= D(\boldsymbol{w}_k)^\top \boldsymbol{r}(\boldsymbol{w}_k),\\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \boldsymbol{z}\end{aligned}} \tag{3.28}$$

Notice that system is similar to the normal equations in Eq. (1.10). For the **Steepest Descent method**, we would have

$$\boxed{\begin{aligned}\boldsymbol{z} &= D(\boldsymbol{w}_k)^\top \boldsymbol{r}(\boldsymbol{w}_k),\\ \boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \eta\,\boldsymbol{z}\end{aligned}} \tag{3.29}$$

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 Chapter 1, 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, represent assumptions that we have made about the structure/architecture of the model.

Recall from the first part of of Chapter 1 that we constructed a fitting function $f(x)$ as

$$f(x) = \sum_{k=1}^{M} \alpha_k \varphi_k(x) = y\,. \tag{4.1}$$

The parameters here are:

- M: the number of terms. M can be larger, equal or smaller than the number of data points
- $\varphi_k(x)$: the basis functions

Here we denote as x_i the abscissae and as y_i the ordinates of the given data points. Furthermore, we denote as α_k our coefficients.

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

Example 1: Polynomial Function Interpolation Revisited

Consider three data points (x_i, y_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

$$\begin{bmatrix} \varphi_1(x_1) & \varphi_2(x_1) & \varphi_3(x_1) \\ \varphi_1(x_2) & \varphi_2(x_2) & \varphi_3(x_2) \\ \varphi_1(x_3) & \varphi_2(x_3) & \varphi_3(x_3) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}, \tag{4.2}$$

where $\varphi_1(x) = 1$, $\varphi_2(x) = x$ and $\varphi_3(x) = x^2$. With the given data, we have to solve the following linear system

$$\begin{bmatrix} 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \tag{4.3}$$

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 because $\alpha_1 = 0$, we can quickly determine $\alpha_3 = 0$ and $\alpha_2 = 1$. To summarize, the interpolating function is given by $f(x) = x$.

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

Imagine now that we have N data points and we use, for example, the 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 $N-1$, $\{\ell_k(x)\}_{k=1}^N$, such that $\ell_i(x_j) = \delta_{ij}$, where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise. This property is satisfied by

$$\ell_k(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_N)}{(x_k-x_1)(x_k-x_2)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_N)} \tag{4.4}$$

A more compact way to write ℓ_k is

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

Using these functions we construct the interpolation function as

$$f(x) = \sum_{k=1}^{N} \alpha_k \, \ell_k(x) \, .$$

In order to find the coefficients we evaluate the polynomial at the point x_i and set it equal to y_i

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

Where we used the constructive property of the polynomials, $\ell_k(x_k) = 1$ and $\ell_k(x_i) = 0$ for $k \neq i$). Finally, the Lagrange interpolation function is given by,

$$\boxed{f(x) = \sum_{k=1}^{N} y_k \, \ell_k(x)} \tag{4.5}$$

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 $g(x)$ that has been sampled at points x_k and for some $x_1 \leq \xi \leq x_N$ is given by

$$\big|g(x) - f(x)\big| = \left| \frac{g^{(n)}(\xi)}{n!} \prod_{k=1}^{n} (x - x_k) \right| \, .$$

  This term can be minimized, if we choose 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 are given by $x_k = -\cos(\frac{2k-1}{2n}\pi)$.

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

Consider N data points (x_i, y_i) with $i = 1 \dots N$. We wish to find a function $f(x)$, such that $y_i \approx f(x_i)$. Assume that the data lie on a straight line, i.e. $y_i = a\,x_i + b$ for some a and b.

- How do you expect the Lagrange functions $\ell_k(x)$ in Eq. (4.4) to look like for $N = 3$? How about larger N?
  Solution: For $N = 3$, the Lagrange functions $\ell_k(x)$ are polynomials of order $N - 1 = 2$ (quadratic). For larger N, the Lagrange functions $\ell_k(x)$ will always be polynomials of order $N - 1$.

- How do you expect the Lagrange interpolator $f(x)$ in Eq. (4.5) 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. Assume we have to interpolate through the set of points $\{(0,1),(2,5),(4,9\}$ that are generated from the linear function $f(x) = 2x + 1$. Using $N = 2$ points:

$$\ell_1(x) = \frac{x-2}{-2} = -\frac{1}{2}(x-2)$$
$$\ell_2(x) = \frac{x-0}{2} = \frac{x}{2}$$
$$f(x) = 1(-\frac{1}{2})(x-2) + 5\frac{x}{2} = 2x+1$$

Using $N = 3$ points:

$$\ell_1(x) = \frac{(x-2)(x-4)}{(0-2)(0-4)} = \frac{(x-2)(x-4)}{8} = \frac{x^2-6x+8}{8}\,,$$
$$\ell_2(x) = \frac{(x-0)(x-4)}{(2-0)(2-4)} = -\frac{x^2-4x}{4}\,,$$
$$\ell_3(x) = \frac{(x-0)(x-2)}{(4-0)(4-2)} = \frac{x^2-2x}{8}\,,$$
$$f(x) = 1\frac{x^2-6x+8}{8} - 5\frac{2x^2-8x}{8} + 9\frac{x^2-2x}{8} = 2x+1\,.$$

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 ($\ell_k(x)$ and $f(x)$) to change if we add a small noise to the data? Adding noise can be modelled as setting $y_i = ax_i + b + \xi_i$, 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 do not change because $\ell_k(x)$ is a function of the x_i points only. But the Lagrange interpolator $f(x)$ is modified. The data without noise is given by $y_i = ax_i + b$ and the data with noise by $y_i^* = ax_i + b + \xi_i$. The corresponding interpolators are given by,

$$f(x) = \sum y_k\,\ell_k(x) = \sum (ax_k + b)\,\ell_k(x)$$
$$f^*(x) = \sum y_k^*\,\ell_k(x) = \sum (ax_k + b + \xi_k)\,\ell_k(x)\,.$$

  The difference between the interpolators is given by,

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

  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.

- Assume we do least squares to fit a linear function $a_L x + b_L$ to the data. Would you expect that $a_L = a$ 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: $a_L = a$, $b_L = b$. If we add noise: $y_i^* = a_L^* x_i + b_L^* + \xi_i$. 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 the case of N data points (x_i, y_i), $i = 1 \dots N$. Estimate the computational complexity in terms of number of basic floating point operations (FLOPs: add, subtract, multiply, divide) to evaluate $f(x)$ in Eq. (4.5) for a given value x. Estimate the computational complexity to evaluate the equivalent $f(x) = \sum_{k=1}^{N} \alpha_k \varphi_k(x)$ in Eq. (4.1) with basis functions $\varphi_k(x) = x^{k-1}$ for 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 $f(x) = \sum_{k=1}^{M} \alpha_k \varphi_k(x)$ with $M < N$?

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

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

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

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

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

To evaluate $f(x) = \sum \alpha_k \varphi_k(x)$, where $\varphi_k(x) = x^{k-1}$, we have

$$f(x) = \alpha_N x^{N-1} + \alpha_{N-1} x^{N-2} + \dots + \alpha_2 x^1 + \alpha_1$$

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 α_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 perform $N-1$ multiplications for all x^{k-1} and $N-1$ multiplications to multiply with all x^{k-1} with a_k. All in all, 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.

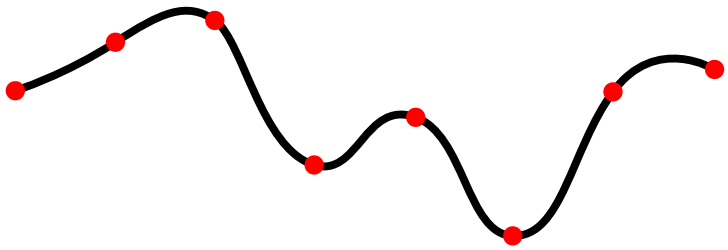


Figure 5.1: Example of a cubic spline fitted through data points (source: wikipedia).

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

$$f'(x) = -f_i'' \frac{(x_{i+1} - x)^2}{2\Delta_i} + f_{i+1}'' \frac{(x - x_i)^2}{2\Delta_i} + C_i, \tag{5.1}$$

$$f(x) = f_i'' \frac{(x_{i+1} - x)^3}{6\Delta_i} + f_{i+1}'' \frac{(x - x_i)^3}{6\Delta_i} + C_i(x - x_i) + D_i. \tag{5.2}$$

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

$$\begin{aligned} y_i = f(x_i) &= f_i'' \frac{\Delta_i^2}{6} + D_i \\ y_{i+1} = f(x_{i+1}) &= f_{i+1}'' \frac{\Delta_i^2}{6} + C_i\Delta_i + D_i \end{aligned} \tag{5.3}$$

This equations can readily be solved for C_i and D_i:

$$C_i = \frac{y_{i+1} - y_i}{\Delta_i} - (f_{i+1}'' - f_i'') \frac{\Delta_i}{6} \tag{5.4}$$

$$D_i = y_i - f_i'' \frac{\Delta_i^2}{6} \tag{5.5}$$

Now we compute $f'(x)$ (using Eq. (5.1) and Eq. (5.4)) once with $x_i \leq x \leq x_{i+1}$ and once $x_{i-1} \leq x \leq x_i$:

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

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

Evaluating the two derivatives at $x = x_i$ and setting them equal we get

$$\frac{\Delta_{i-1}}{6} f''_{i-1} + \left(\frac{\Delta_{i-1}+\Delta_i}{3}\right) f''_i + \frac{\Delta_i}{6} f''_{i+1} = \frac{y_{i+1}-y_i}{\Delta_i} - \frac{y_i - y_{i-1}}{\Delta_{i-1}} \tag{5.6}$$

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 $O(N)$ using the Tridiagonalmatrix-Algorithm (TDMA). To understand this algorithm consider a general tridiagonal matrix

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.7}$$

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

$$\begin{bmatrix} 1 & \frac{c_1}{b_1} & & & 0 \\ & b_2 - a_2\frac{c_1}{b_1} & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} \frac{d_1}{b_1} \\ d_2 - a_2\frac{d_1}{b_1} \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.8}$$

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

$$\begin{bmatrix} 1 & w_1 & & & 0 \\ & 1 & \frac{c_2}{b_2 - a_2 w_1} & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{N-1} \\ 0 & & & a_N & b_N \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} g_1 \\ \frac{d_2 - a_1 g_1}{b_2 - a_2 w_1} \\ d_3 \\ \vdots \\ d_N \end{bmatrix}. \tag{5.9}$$

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

$$w_i = \begin{cases} \dfrac{c_1}{b_1}, & i = 1 \\ \dfrac{c_i}{b_i - a_i w_{i-1}}, & i = 2, 3, \ldots, N-1 \end{cases} \quad \text{and} \quad g_i = \begin{cases} \dfrac{d_1}{b_1}, & i = 1 \\ \dfrac{d_i - a_i g_{i-1}}{b_i - a_i w_{i-1}}, & i = 2, 3, \ldots, N \end{cases} \tag{5.10}$$

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

$$x_i = \begin{cases} g_N, & i = N \\ g_i - w_i x_{i+1}, & i = N-1, N-2, \ldots, 1 \end{cases} \tag{5.11}$$

As we see, the computational complexity is $O(N)$.

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: $E \approx \min \int [f''(x)]^2 dx \implies$ 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 again). 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 polynomial of degree d which is only non-zero for the range $t_i \leq x \leq t_{i+d+1}$ via:

$$
\begin{aligned}
B_{i,0,t}(x) &= \begin{cases} 1 & \text{if } t_i \leq x < t_{i+1}, \\ 0 & \text{otherwise,} \end{cases} \\
B_{i,d,t}(x) &= \frac{x - t_i}{t_{i+d} - t_i} B_{i,d-1,t}(x) + \frac{t_{i+d+1} - x}{t_{i+d+1} - t_{i+1}} B_{i+1,d-1,t}(x).
\end{aligned}
\tag{5.12}
$$

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

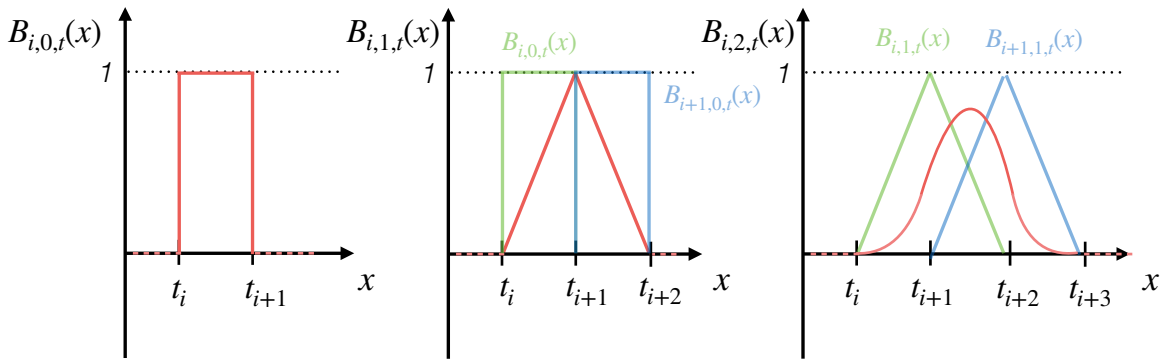


Figure 5.3: Illustration of the construction of the B-splines. The new spline (in red) is obtained from the old (green and blue) by linear interpolation (please see http://geometrie.foretnik.net/files/NURBS-en.swf for accurate plots of the resulting functions).

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:

$$\{\underbrace{t_1, \ldots, t_{d+1}}_{d+1 \text{ equal knots}}, \underbrace{t_{d+2}, \ldots, t_M}_{M-d-1 \text{ internal knots}}, \underbrace{t_{M+1}, \ldots, t_{M+d+1}}_{d+1 \text{ equal knots}}\} \tag{5.13}$$

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. (5.12), repeated 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:

$$S_{d,t}(x) = \sum_{i=1}^{M} \alpha_i B_{i,d,t}(x), \quad \text{for} \quad t_{d+1} \leq x < t_{M+1} \tag{5.14}$$

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. (5.14) 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 *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 $z_k = f(x_k, y_k)$

5.4.1 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

$$f(x, y) = \sum_i \sum_j a_{i,j} \phi_i(x) \phi_j(y) \tag{5.15}$$

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

$$f(x_p, y_q) = Z(x_p, y_q) = \sum_i \sum_j a_{i,j} \phi_i(x_p) \phi_j(y_q) \tag{5.16}$$

Using Lagrange polynomials we can show that

$$f(x_p, y_q) = \sum_i \sum_j Z(x_i, y_j) l_i(x_p) l_j(y_q) \tag{5.17}$$

where

$$l_i(x) = \frac{(x - x_1)(x - x_2) \dots (x - x_{i-1})(x - x_{i+1}) \dots (x - x_M)}{(x_i - x_1)(x_i - x_2) \dots (x_i - x_{i-1})(x_i - x_{i+1}) \dots (x_i - x_M)} \tag{5.18}$$

and

$$l_j(y) = \frac{(y - y_1)(y - y_2) \dots (y - y_{j-1})(y - y_{j+1}) \dots (y - y_N)}{(y_j - y_1)(y_j - y_2) \dots (y_j - y_{j-1})(y_j - y_{j+1}) \dots (y_j - y_N)} \tag{5.19}$$

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

5.4.2 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

  $$f(x,y) = \frac{\sum_{k=1}^{n} z_k \, g(x-x_k, y-y_k)}{\sum_{k=1}^{n} g(x-x_k, y-y_k)} \tag{5.20}$$

  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

  $$g(x,y) \; = \; \frac{1}{(x^2+y^2)^{\mu/2}} \tag{5.21}$$

  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 $(0 \leq x, y \leq 1)$ 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:

  $$\begin{aligned} z(x,y) = {} & y \cdot z(x,1) + (1-y) \cdot z(x,0) && \rightarrow \text{ linear interpolation in } y \\ & + x \cdot z(1,y) + (1-x) \cdot z(0,y) && \rightarrow \text{ linear interpolation in } x \\ & - (1-x) \cdot (1-y) \cdot z(0,0) - (1-x) \cdot y \cdot z(0,1) && \rightarrow \text{ bilinear interpolation} \\ & - (1-y) \cdot x \cdot z(1,0) - x \cdot y \cdot z(1,1) && \rightarrow \text{ bilinear interpolation} \end{aligned} \tag{5.22}$$

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. (5.6)) 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. (5.6).

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

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. (5.6) 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. (5.6) 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. (5.6)?

Solution:

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

- for a straight line:

$$\begin{aligned} \frac{h}{6}f''_{i-1} + \frac{2h}{3}f''_i + \frac{h}{6}f''_{i+1} &= \frac{a + bx_{i+1} - (a + bx_i)}{h} - \frac{a + bx_i - (a + bx_{i-1})}{h} \\ &= \frac{b(x_i + h - x_i)}{h} - \frac{b(x_i - (x_i - h))}{h} = 0 \end{aligned}$$

  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. (5.6).

- for a parabola:

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

$$\frac{1}{6}f''_{i-1} + \frac{2}{3}f''_i + \frac{1}{6}f''_{i+1} = 2c$$

  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. (5.6).

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. (5.14) 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)?

Solution:

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

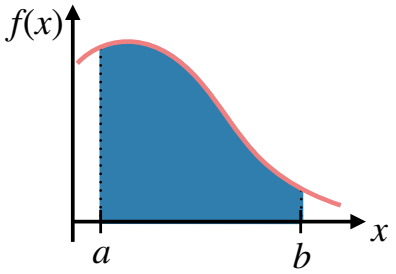
$$S_{d,t}(x_i) = \sum_{i=1}^{N} a_i B_{i,d,t}(x_i) = y_i \quad i = 1, \ldots, N$$

The resulting basis cubic splines (order $d = 3$) are linearly combined in order to construct the final formula using Eq. (5.14). 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

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

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



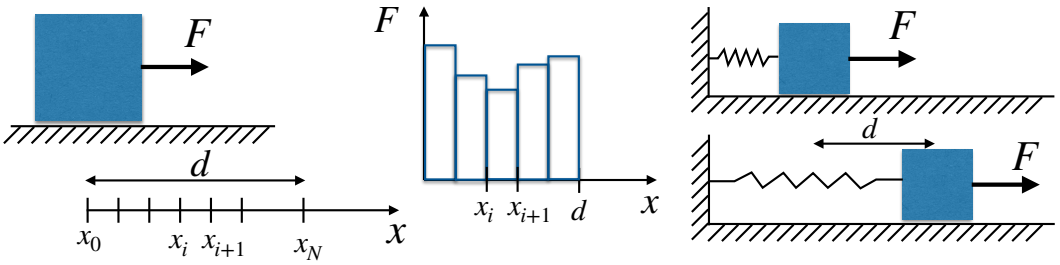
$$I = \int_a^b f(x)\ \mathrm{d}x \tag{6.1}$$

Numerical integration is crucial in various situations:

- The integral cannot be solved analytically. Example: $I = \int_0^1 \sin(\cos(x))\ \mathrm{d}x$
- The 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 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., $F = F(x)$, we can approximately assume that $F(x) = const.$ over a segment Δx (in the interval $[x_i, x_{i+1}]$). The work we perform in this interval is $\Delta A_i = F_i \Delta x_i$

and the total work is

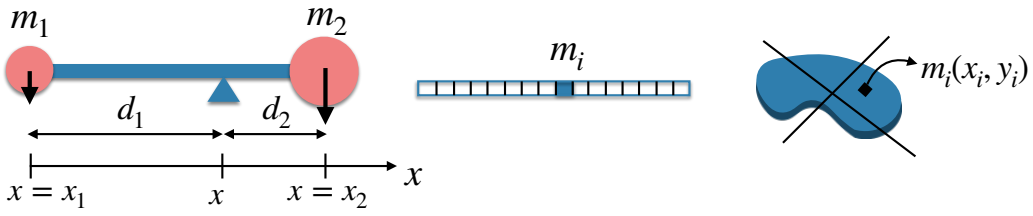
$$A = \sum_{i=0}^{N} \Delta A_i = \sum_{i=0}^{N} F(x_i)\Delta x_i,$$

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 $A = \int_0^d F(x)\ \mathrm{d}x$. 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

$$A = \int_0^d kx\ \mathrm{d}x = \frac{kd^2}{2}.$$

Example 2: Moments

Consider a beam that has point objects of mass m_1 and m_2 attached at its left and right end 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

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

6.1 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

$$\int_a^b f(x)\ \mathrm{d}x = \int_a^c f(x)\ \mathrm{d}x + \int_c^b f(x)\ \mathrm{d}x, \tag{6.2}$$

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

$$I = \int_a^b f(x)\ \mathrm{d}x = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)\ \mathrm{d}x \tag{6.3}$$

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 2-step "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.

$$I \approx \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} p_i(x)\ \mathrm{d}x \tag{6.4}$$

6.2 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:

$$I_{R_i} = f(x_i)\,\Delta_i \tag{6.5}$$

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

$$I_{M_i} = f(x_{i+1/2})\,\Delta_i = f\left(\frac{x_i + x_{i+1}}{2}\right)\,\Delta_i \tag{6.6}$$

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

$$I_{T_i} = \frac{f(x_i) + f(x_{i+1})}{2}\Delta_i \tag{6.7}$$

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

$$I_{S_i} = \frac{f(x_i) + 4\,f((x_i + x_{i+1})/2) + f(x_{i+1})}{6}\Delta_i \tag{6.8}$$

Fig. 6.1 shows an example of such approximations.

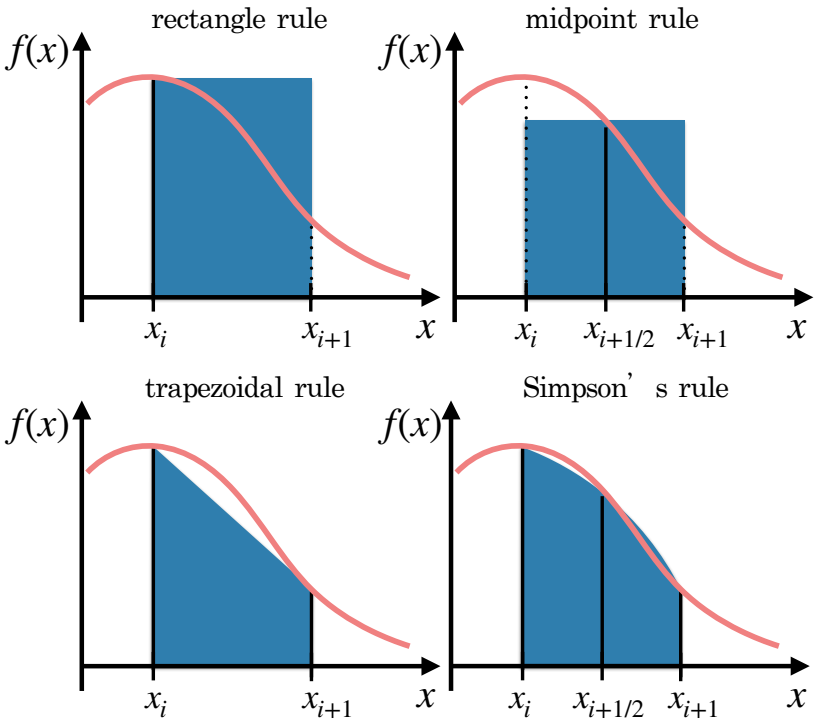


Figure 6.1: Example comparing the rectangle, midpoint, trapezoidal and the Simpson's rule for numerical integration.

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

$$\begin{aligned}
&\textbf{Rectangle Rule:} && I \approx \Delta_x \sum_{i=0}^{N-1} f(x_i), \\
&\textbf{Midpoint Rule:} && I \approx \Delta_x \sum_{i=0}^{N-1} f\left(\frac{x_i + x_{i+1}}{2}\right), \\
&\textbf{Trapezoidal Rule:} && I \approx \frac{\Delta_x}{2}\left(f(x_0) + 2\sum_{i=1}^{N-1} f(x_i) + f(x_N)\right),
\end{aligned} \tag{6.9}$$

For Simpson's approximation, we can rewrite Eq. (6.3) 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:

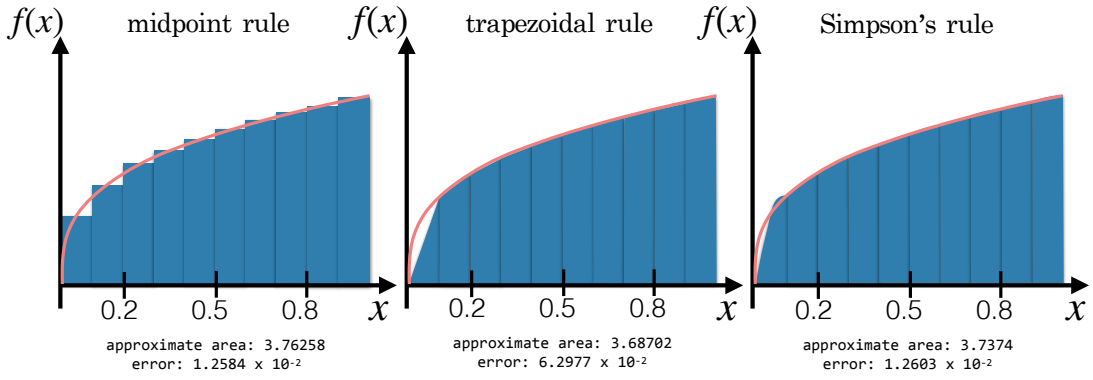
$$I = \int_a^b f(x)\,\mathrm{d}x = \sum_{\substack{i=0 \\ i=\text{even}}}^{N-2} \int_{x_i}^{x_{i+2}} f(x)\,\mathrm{d}x$$

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

Simpson's Rule:
$$I \approx \frac{\Delta_x}{3}\left(f(x_0) + 4 \sum_{\substack{i=1 \\ i=\text{odd}}}^{N-1} f(x_i) + 2 \sum_{\substack{i=2 \\ i=\text{even}}}^{N-2} f(x_i) + f(x_N) \right). \tag{6.10}$$

Example 3: Composite Integration

We wish to evaluate $I = \int_0^1 5\sqrt[3]{x}\,\mathrm{d}x$, i.e., the shaded area in the 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 the most fundamental methods approximate an integral as a weighted sum of f_i:

$$I = \int_a^b f(x)\,\mathrm{d}x \approx \sum_{i=0}^{N} w_i f(x_i), \tag{6.11}$$

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 $i = 1, \ldots, N-1$.

6.2.1 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. (6.3), 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

$$\begin{aligned} p_i(x) &= \sum_{k=0}^{M} f(x_k) l_k^M(x), \\ l_k^M(x) &= \frac{(x-x_0)(x-x_1)\ldots(x-x_{k-1})(x-x_{k+1})\ldots(x-x_M)}{(x_k-x_0)(x_k-x_1)\ldots(x_k-x_{k-1})(x_k-x_{k+1})\ldots(x_k-x_M)}. \end{aligned} \tag{6.12}$$

The integral I is then approximated as

$$I_i = \int_{x_i}^{x_{i+1}} f(x)\ \mathrm{d}x \approx \int_{x_i}^{x_{i+1}} p_i(x)\ \mathrm{d}x = \sum_{k=0}^{M} f(x_k) \int_{x_i}^{x_{i+1}} l_k^M(x)\ \mathrm{d}x. \tag{6.13}$$

We rewrite this as

$$I_i \approx \Delta_i \sum_{k=0}^{M} C_k^M f(x_k), \text{ where } C_k^M = \frac{1}{\Delta_i} \int_{x_i}^{x_{i+1}} l_k^M(x)\ \mathrm{d}x \tag{6.14}$$

Properties of C_k^M:

- Lagrange polynomials fit exactly a constant function: e.g. $f(x) = 1$ must be integrated exactly. In Eq. (6.14), we hence want to have

$$I_i = \int_{x_i}^{x_{i+1}} 1\ \mathrm{d}x = (x_{i+1} - x_i) \sum_{k=0}^{M} C_k^M 1 \tag{6.15}$$

  and it follows that

$$\sum_{k=0}^{M} C_k^M = 1 \tag{6.16}$$

- 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, insert $x_{M-j} = x_i + (M-j)h$ in the right hand side (where $h = \frac{x_{i+1}-x_i}{M}$ 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 we use this on our definition of the coefficient C_k^M and use a substitution of variables, we find that

$$C_k^M = C_{M-k}^M \tag{6.17}$$

Example 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:

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

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

If we insert $C_0^1 = C_1^1 = 1/2$ in Eq. (6.14), 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: $C_0^2 = 1/6$, $C_1^2 = 2/3$, $C_2^2 = 1/6$.

6.2.2 Error analysis

We would like to evaluate the error we obtain in the final value of an integral, when we use different numerical integration rules, i.e. we would like to find an upper bound or an approximation of

$$E_{\mathrm{rule},i} = I_i - I_{\mathrm{rule},i} \tag{6.18}$$

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

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

$$\begin{aligned} f(x) =& f(x_{i+1/2}) + \left(x - x_{i+1/2}\right) f'(x_{i+1/2}) + \frac{1}{2}\left(x - x_{i+1/2}\right)^2 f''(x_{i+1/2}) \\ &+ \frac{1}{6}\left(x - x_{i+1/2}\right)^3 f'''(x_{i+1/2}) + \cdots, \end{aligned} \tag{6.19}$$

We can use this expansion to evaluate I_i:

$$\begin{aligned} I_i &= \int_{x_i}^{x_{i+1}} f(x)\,\mathrm{d}x \\ &= f(x_{i+1/2}) \int_{x_i}^{x_{i+1}} \mathrm{d}x + f'(x_{i+1/2}) \int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right) \mathrm{d}x \\ &\quad + \frac{1}{2} f''(x_{i+1/2}) \int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right)^2 \mathrm{d}x + \frac{1}{6} f'''(x_{i+1/2}) \int_{x_i}^{x_{i+1}} \left(x - x_{i+1/2}\right)^3 \mathrm{d}x + \cdots \\ &= \underbrace{f(x_{i+1/2})\,\Delta_i}_{I_{M_i}} + \underbrace{0 + \frac{1}{24} f''(x_{i+1/2})\Delta_i^3 + 0 + O(\Delta_i^5)}_{E_{M_i}} \end{aligned} \tag{6.20}$$

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:

$$\boxed{E_{M_i} = \frac{1}{24} f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5) + \cdots} \tag{6.21}$$

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. (6.21), we see that

$$\boxed{E_{T_i} = -\frac{1}{12} f''(x_{i+1/2})\Delta_i^3 + O(\Delta_i^5) + \cdots} \tag{6.22}$$

Simpson's rule: We can combine the midpoint and the trapezoidal rule. By comparing Eq. (6.8) with Eq. (6.6) and Eq. (6.7), we see that:

$$I_{S_i} = \frac{2}{3} I_{M_i} + \frac{1}{3} I_{T_i} \tag{6.23}$$

therefore

$$\boxed{E_{S_i} = O(\Delta_i^5) + \cdots}. \tag{6.24}$$

We note 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. (6.3)):

$$\begin{aligned}\sum_{i=0}^{N-1} I_{M_i} &= \sum_{i=0}^{N-1}\left(I_i - \frac{1}{24}f''(x_{i+1/2})\Delta_x^3 + O(\Delta_x^5) + \cdots\right),\\ \left|\sum_{i=0}^{N-1} I_{M_i} - I\right| &< N\frac{1}{24}\max_i(|f''(x_{i+1/2})|)\Delta_x^3 + NO(\Delta_x^5) + \cdots,\\ &= \frac{b-a}{24}\max_i(|f''(x_{i+1/2})|)\Delta_x^2 + O(\Delta_x^4) + \cdots,\end{aligned} \tag{6.25}$$

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 7

Numerical integration II: Richardson and Romberg

What are the 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).

7.1 Richardson Extrapolation

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

$$G \approx G(h) \tag{7.1}$$

In other words, if we calculate any quantity on the computer, the result depends on our choice of the grid spacing h. As the 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 the approximation in terms of a Taylor series for h:

$$G(h) = G(0) + c_1 h + c_2 h^2 + \cdots , \tag{7.2}$$

where $c_1, c_2, \ldots$ are the typical constants we obtain from this expansion, i.e. $G'(0), G''(0)/2, \ldots$ (some constants may be 0). As aforementioned, 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 stated in the beginning of the section, one way to go about that is to split h into $h/2$ so that

$$G(h/2) = G + \frac{1}{2} c_1 h + \frac{1}{4} c_2 h^2 + \cdots . \tag{7.3}$$

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. (7.2) and Eq. (7.3) to reduce the error:

$$G_1(h) = 2\,G(h/2) - G(h) = G + c_2' h^2 + c_3' h^3 + \cdots, \tag{7.4}$$

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

$$G_2(h) = \frac{1}{3}\left(4\,G_1(h/2) - G_1(h)\right) = G + O(h^3), \tag{7.5}$$

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

$$G_n(h) = \frac{1}{2^n - 1}\left(2^n\,G_{n-1}(h/2) - G_{n-1}(h)\right) = G + O(h^{n+1}) \tag{7.6}$$

Example 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:

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

Since the error contains only even terms $(O(1/E^2), O(1/E^4), \ldots)$, we can adapt Eq. (7.6) and write

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

7.2 Error Estimation

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

$$G(h/2) - G(h) = -\frac{1}{2}c_1 h - \frac{3}{4}c_2 h^2 + O(h^3). \tag{7.7}$$

On the other hand, we can rearrange Eq. (7.3) to get

$$\epsilon(h/2) = G - G(h/2) = -\frac{1}{2}c_1 h - \frac{1}{4}c_2 h^2 + O(h^3). \tag{7.8}$$

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

$$\epsilon(h/2) \approx G(h/2) - G(h) \tag{7.9}$$

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.

7.3 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 within a single interval. Then we recalculate within two intervals, four, eight, and so on with results of the series of integrations:

$$I_0^1, I_0^2, I_0^4, \cdots \tag{7.10}$$

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

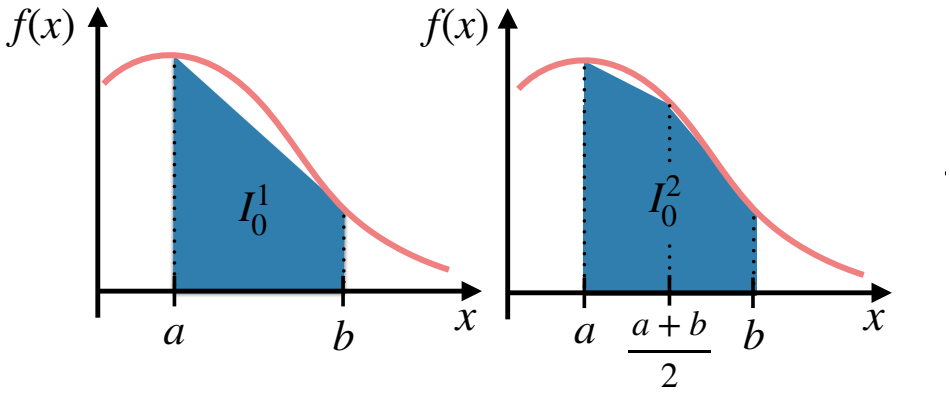


Figure 7.1: Graphical representation of the refinement step for the example of the trapezoidal rule.

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

$$I = \int_a^b f(x)\,\mathrm{d}x = \underbrace{\frac{h}{2}\left[f(a) + f(b) + 2\sum_{j=1}^{n-1} f_j\right]}_{I_0^n} + c_1 h^2 + c_2 h^4 + c_3 h^6 + \cdots, \tag{7.11}$$

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

$$I_0^n = I - c_1 h^2 - c_2 h^4 - c_3 h^6 \tag{7.12}$$

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

$$I_0^{2n} = I - c_1\frac{h^2}{4} - c_2\frac{h^4}{16} - c_3\frac{h^6}{64}\cdots \tag{7.13}$$

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

$$(\star)\ I_1^n = \frac{4I_0^{2n} - I_0^n}{3} = I + \frac{1}{4}c_2h^4 + \frac{5}{16}c_3h^6 \tag{7.14}$$

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

$$I_0^{4n} = I - c_1\frac{h^2}{16} - c_2\frac{h^4}{256} - c_3\frac{h^6}{4096} - \cdots \tag{7.15}$$

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

$$(\star\star)\ I_1^{2n} = \frac{4I_0^{4n} - I_0^{2n}}{3} = I + \frac{1}{64}c_2h^4 + \frac{5}{1024}c_3h^6 \tag{7.16}$$

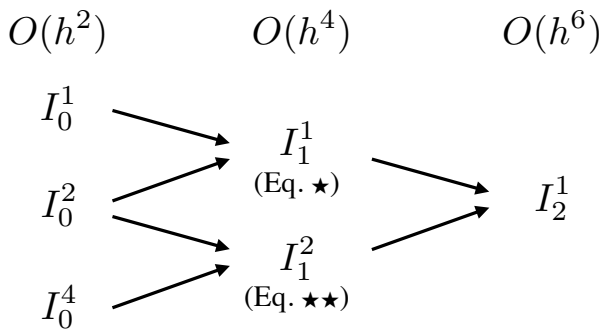


Figure 7.2: Graphical representation of the Romberg integration.

We can now combine I_1^n and I_1^{2n} and eliminate the $O(h^4)$ terms and get a 6th order accurate formula. The process may be continued to get

$$I_k^n = \frac{4^k I_{k-1}^{2n} - I_{k-1}^n}{4^k - 1} \tag{7.17}$$

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 8

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.

8.1 Adaptive Integration

For the numerical calculation of integrals, we want to increase the accuracy where needed and stop refining in the other regions. Thus, we have to identify the regions where refinement is needed. Out of the methods learned in the last lecture we already derived the formal tool to estimate the error for a given quadrature rule in the section about Richardson extrapolation. We found that the error for the approximation of any quantity G for grid spacing h

$$\epsilon(h/2) \approx G(h/2) - G(h) \tag{8.1}$$

Using this, we can 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
```

8.2 Gauss Quadrature

In the spirit of the previous section, namely that there are certain points and regions that are more important when evaluating an 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 a problem of solving a non-linear set of equations.

8.2.1 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

$$\int_a^b f(x)\,\mathrm{d}x \approx c_1 f(a) + c_2 f(b). \tag{8.2}$$

Let the right hand side of (Eq. (8.2)) be exact for integrals of a straight line, e.g.

$$\int_a^b f(x)\,\mathrm{d}x = \int_a^b (a_0 + a_1 x)\,\mathrm{d}x = \left[a_0 x + a_1 \frac{x^2}{2}\right]_a^b = a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right). \tag{8.3}$$

In particular, for $f(x) = a_0 + a_1 x$, we want the right hand sides of (Eq. (8.2)) and (Eq. (8.3)) to be equal,

$$a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) = c_1(a_0 + a_1 a) + c_2(a_0 + a_1 b), \tag{8.4}$$

which can be rewritten as

$$a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) = a_0(c_1 + c_2) + a_1(c_1 a + c_2 b). \tag{8.5}$$

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

$$c_1 + c_2 = b - a, \qquad c_1 a + c_2 b = \frac{b^2-a^2}{2}. \tag{8.6}$$

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

$$c_1 = c_2 = \frac{b-a}{2}, \tag{8.7}$$

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

$$I = \int_a^b f(x)\,\mathrm{d}x \approx c_1 f(x_1) + c_2 f(x_2). \tag{8.8}$$

There are four unknowns x_1, x_2, c_1, and c_2. These are evaluated by requiring that the rule (Eq. (8.8)) 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

$$\begin{aligned}\int_a^b f(x)\,\mathrm{d}x = \int_a^b (a_0 + a_1 x + a_2 x^2 + a_3 x^3)\,\mathrm{d}x &= \left[a_0 x + a_1 \frac{x^2}{2} + a_2 \frac{x^3}{3} + a_3 \frac{x^4}{4}\right]_a^b \\ &= a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) + a_2\left(\frac{b^3-a^3}{3}\right) + a_3\left(\frac{b^4-a^4}{4}\right).\end{aligned} \tag{8.9}$$

Alternatively, applying quadrature rule (Eq. (8.8)), we obtain

$$\int_a^b f(x)\,\mathrm{d}x \approx c_1 f(x_1) + c_2 f(x_2) = c_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + c_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3). \tag{8.10}$$

Equating the right hand sides of (Eq. (8.9)) and (Eq. (8.10)), we have

$$\begin{aligned}&a_0(b-a) + a_1\left(\frac{b^2-a^2}{2}\right) + a_2\left(\frac{b^3-a^3}{3}\right) + a_3\left(\frac{b^4-a^4}{4}\right) \\ &= c_1(a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3) + c_2(a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3) \\ &= a_0(c_1 + c_2) + a_1(c_1 x_1 + c_2 x_2) + a_2(c_1 x_1^2 + c_2 x_2^2) + a_3(c_1 x_1^3 + c_2 x_2^3).\end{aligned} \tag{8.11}$$

Since in (Eq. (8.11)) 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 of equations for c_1, c_2, x_1, and x_2:

$$\begin{aligned} b - a &= c_1 + c_2, \\ \frac{b^2-a^2}{2} &= c_1 x_1 + c_2 x_2, \\ \frac{b^3-a^3}{3} &= c_1 x_1^2 + c_2 x_2^2, \\ \frac{b^4-a^4}{4} &= c_1 x_1^3 + c_2 x_2^3. \end{aligned} \tag{8.12}$$

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

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

$$\begin{aligned} c_1 &= \frac{b-a}{2}, \\ c_2 &= \frac{b-a}{2}, \\ x_1 &= \left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}, \\ x_2 &= \left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}. \end{aligned} \tag{8.13}$$

Solution values for c_1, c_2, x_1, and x_2 as in (Eq. (8.13)), inserted into (Eq. (8.8)), lead to the **two-point Gauss quadrature rule**:

$$\int_a^b f(x)\,\mathrm{d}x \approx \frac{b-a}{2} f\left[\left(\frac{b-a}{2}\right)\left(-\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}\right] + \frac{b-a}{2} f\left[\left(\frac{b-a}{2}\right)\left(\frac{1}{\sqrt{3}}\right) + \frac{b+a}{2}\right]. \tag{8.14}$$

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

8.2.2 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 that 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 $y_i' = f'(x_i)$):

$$f(x) = \sum_{k=1}^{n} U_k(x) y_k + \sum_{k=1}^{n} V_k(x) y_k', \tag{8.15}$$

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

$$U_k(x_j) = \delta_{jk}, \qquad U_k'(x_j) = 0, \tag{8.16}$$

$$V_k(x_j) = 0, \qquad V_k'(x_j) = \delta_{jk}. \tag{8.17}$$

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

$$L_k(x) = \frac{(x-x_1)(x-x_2)\dots(x-x_{k-1})(x-x_{k+1})\dots(x-x_n)}{(x_k-x_1)(x_k-x_2)\dots(x_k-x_{k-1})(x_k-x_{k+1})\dots(x_k-x_n)} \tag{8.18}$$

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

$$U_k(x) = \left[1 - 2L_k'(x_k)(x-x_k)\right] L_k^2(x), \tag{8.19}$$

$$V_k(x) = (x-x_k)L_k^2(x). \tag{8.20}$$

Given the properties of the Lagrange polynomials, we can prove 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.

8.2.3 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

$$x = \frac{2\xi - (a+b)}{b-a} \tag{8.21}$$

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

Let us now use the found interpolation to obtain $\int_{-1}^{1} f(x)\mathrm{d}x$. We approximate f by Hermite polynomials (Eq. (8.15))

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \sum_{k=1}^{n} y_k \int_{-1}^{1} U_k(x)\,\mathrm{d}x + \sum_{k=1}^{n} y_k' \int_{-1}^{1} V_k(x)\,\mathrm{d}x, \tag{8.22}$$

which can be rewritten as follows:

$$\int_{-1}^{1} f(x)\,\mathrm{d}x = \sum_{k=1}^{n} u_k f(x_k) + \sum_{k=1}^{n} v_k f'(x_k), \tag{8.23}$$

where:

$$u_k = \int_{-1}^{1} U_k(x)\,\mathrm{d}x, \qquad v_k = \int_{-1}^{1} V_k(x)\,\mathrm{d}x. \tag{8.24}$$

In order for equation Eq. (8.23) 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

$$v_k = \int_{-1}^{1} (x - x_k) L_k^2(x)\,\mathrm{d}x. \tag{8.25}$$

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

$$C_k = \frac{1}{(x_k - x_1)(x_k - x_2)\cdot\ldots\cdot(x_k - x_{k-1})(x_k - x_{k+1})\cdot\ldots\cdot(x_k - x_n)}, \tag{8.26}$$

$$F(x) = (x - x_1)(x - x_2)\cdot\ldots\cdot(x - x_{n-1})(x - x_n). \tag{8.27}$$

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

$$v_k = C_k \int_{-1}^{1} F(x) L_k(x)\ \mathrm{d}x. \tag{8.28}$$

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

$$0 = \int_{-1}^{1} F(x) L_k(x)\ \mathrm{d}x \tag{8.29}$$

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. (8.29) 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:

$$\int_{-1}^{1} P_n(x) P_m(x)\ \mathrm{d}x = N_n \delta_{nm}, \tag{8.30}$$

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 $+1$.

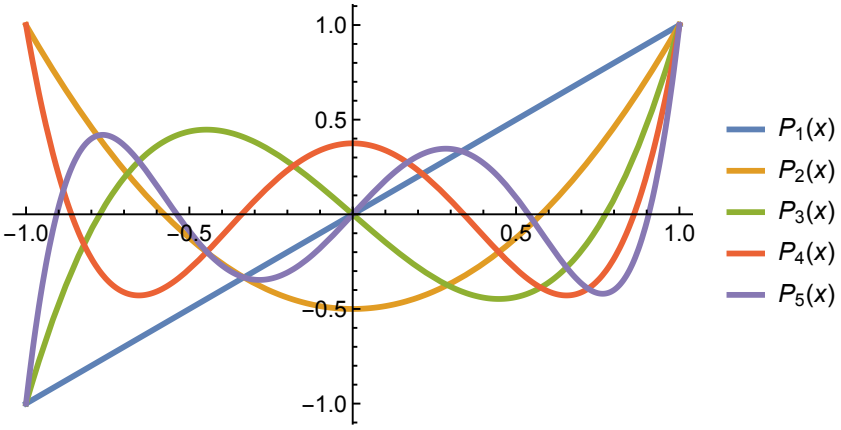


Figure 8.1: Legendre polynomials $P_n(x)$ for $n = 1, \ldots, 5$.

The weights can be computed from Eq. (8.24) for the given abscissa, so we have that

$$\int_{-1}^{1} f(x)\ \mathrm{d}x = \sum_{k=1}^{n} u_k f(x_k) \tag{8.31}$$

We compute x_k as the k-th root of $P_n(x)$ and one can show that the weights in Eq. (8.31) are given by (according to [5]):

$$u_k = \frac{2}{\left(1 - x_k^2\right)\left(P_n'(x_k)\right)^2}. \tag{8.32}$$

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:

$$\varepsilon = \frac{2^{2n+1}(n!)^4}{(2n+1)(2n!)^3} f^{(2n)}(\xi). \tag{8.33}$$

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 preferred.

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 9

Numerical integration IV: Monte Carlo

In this chapter, we discuss another method, in which 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 problems. Luckily, we can resolve these problems by means of the Monte Carlo integration.

9.1 Curse of dimensionality

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

$$f : \mathbb{R}^d \to \mathbb{R}, \qquad \vec{x} = (x^{(1)}, \ldots, x^{(d)}) \mapsto f(x^{(1)}, \ldots, x^{(d)}), \tag{9.1}$$

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 $r = 1, \ldots, d$

$$I = \int_\Omega f(x^{(1)}, \ldots, x^{(d)}) \, \mathrm{d}\vec{x} \tag{9.2}$$

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

$$I = \int_{\Omega_1} \cdots \int_{\Omega_d} f(x^{(1)}, \ldots, x^{(d)}) \, \mathrm{d}x^{(d)} \cdots \mathrm{d}x^{(1)}, \tag{9.3}$$

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

$$I \approx \sum_{\substack{i_1=1 \\ \cdots \\ i_d=1}}^{n} \tilde{w}_{i_1,\ldots,i_d} f(x_{i_1}^{(1)}, \ldots, x_{i_d}^{(d)}), \quad \text{with} \quad \tilde{w}_{i_1,\ldots,i_d} = \prod_{r=1}^{d} w_{i_r}. \tag{9.4}$$

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

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

$$I - I_S = \mathcal{O}(h^4). \tag{9.5}$$

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

$$h = \frac{b_1 - a_1}{n} = \mathcal{O}(n^{-1}) = \mathcal{O}(M^{-1}), \tag{9.6}$$

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

$$h = \frac{(b_1 - a_1)}{n} = \cdots = \frac{(b_d - a_d)}{n} = \mathcal{O}(n^{-1}) = \mathcal{O}(M^{-1/d}). \tag{9.7}$$

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

$$I - I_S = \mathcal{O}(M^{-4/d}). \tag{9.8}$$

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 infeasible.

9.2 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 fundamental concept in probability theory is **random variables** X. For simplicity, let us first regard discrete random variables. They 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 fulfil the property that they sum up to one

$$\sum_i P(x_i) = 1. \tag{9.9}$$

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 definitions 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 throwing k heads in n tosses, given the probability of tossing a single head $\mathbb{P}(X = \text{Head}) = p$

$$P(k) = \binom{n}{k} p^k (1 - p)^{n-k} \tag{9.10}$$

In general we do not only regard discrete, but also continuous spaces. In order to treat them we have to introduce further objects. From a mathematical point of view, this requires us to consider assigning a probability not to each element of a set, but rather to intervals of the space.

9.2.1 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 values 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

$$F_X(x) = P(X \leq x). \tag{9.11}$$

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

$$p(x) = \frac{\mathrm{d}}{\mathrm{d}x} F_X(x). \tag{9.12}$$

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

$$\int p(x)\,\mathrm{d}x = 1 \tag{9.13}$$

It is important to realize that, although they share some properties, the PDF can not be identified directly to the probability of an event as it was the case in the discrete setting. This introduces another important property of the PDF, namely that the probability that a random variable lies within an interval $[a, b]$ is given by

$$P(a \leq X \leq b) = \int_a^b p(x)\,\mathrm{d}x. \tag{9.14}$$

A typical example here is the **uniform distribution** $\mathcal{U}([a, b])$, which is defined in an interval $[a, b]$ and whose PDF reads

$$p_{\mathcal{U}}(x) = \frac{1}{b-a} \tag{9.15}$$

This can be readily generalized to a domain $\Omega \subseteq \mathbb{R}^n$, where we then find $p_{\mathcal{U}}(\vec{x}) = \frac{1}{|\Omega|}$ 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 its mean μ and standard deviation σ. Its pdf reads

$$p_{\mathcal{N}}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{9.16}$$

9.2.2 Expected value and variance

The **expected value** or **expectation** $\mathbb{E}$ of a random variable X over a domain Ω is defined as

$$\mathbb{E}[X] = \langle X \rangle = \int_\Omega x p(x)\,\mathrm{d}x. \tag{9.17}$$

We note that for a given random variable X, $f(X)$ is also 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**

$$\mathbb{E}(x) = \bar{x} = \sum_i x_i P(x_i) \tag{9.18}$$

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

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y] \tag{9.19}$$

The **variance** Var of a random variable X over a domain Ω is defined as

$$\mathrm{Var}[X] = \sigma^2[X] = \mathbb{E}\Big[(X - \mathbb{E}[X])^2\Big], \tag{9.20}$$

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:

$$\sigma^2[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2. \tag{9.21}$$

If two random variables X, Y are **uncorrelated**, i.e. for the expectation value of a product of two variables holds that

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y], \tag{9.22}$$

then the linearity also holds for the variance

$$\sigma^2\Big[\sum_i a_i Y_i\Big] = \sum_i a_i^2 \sigma^2[Y_i]. \tag{9.23}$$

If two random variables are **independent**, the expectation of the product of the random variables can be factored; nonetheless this property is stronger and not all uncorrelated variables are independent.

9.3 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 Fig. 9.1. This means, that we need to integrate the function $f : [0, 1]^2 \to \mathbb{R}$, defined as

$$f(x, y) = \begin{cases} 1 & \text{if } \sqrt{x^2 + y^2} \leq 1, \\ 0 & \text{else,} \end{cases} \tag{9.24}$$

over the domain $[0, 1]^2$.

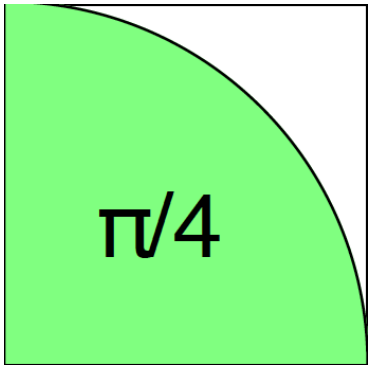


Figure 9.1: The area of the quarter circle with radius $r = 1$ is equal to $\pi/4$.

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 and 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

$$I = |\Omega|\langle f \rangle, \tag{9.25}$$

where we can identify $\langle f \rangle$ as the previously introduced expectation value of the integrand f over Ω, assuming a uniform distribution

$$\langle f \rangle = \frac{1}{|\Omega|} \int_\Omega f(\vec{x})\, \mathrm{d}\vec{x}, \qquad |\Omega| = \int_\Omega \mathrm{d}\vec{x}. \tag{9.26}$$

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 $\vec{x}_i$ (called "samples"), that are drawn from a uniform random distribution in Ω (i.e. regarded as samples from a random variable) and compute the average of all values (perform "sampling")

$$\langle f \rangle \approx \langle f \rangle_M = \frac{1}{M} \sum_{i=1}^{M} f(\vec{x}_i). \tag{9.27}$$

Two interpretations of MC estimation are given in Fig. 9.2, where MC sampling is performed for a one-dimensional function f on domain $|\Omega| = b - a$ and the number of samples is set to $M = 4$.

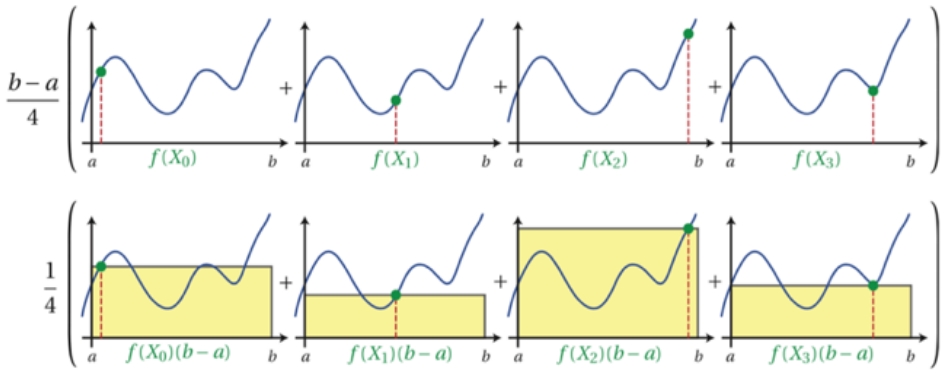


Figure 9.2: Illustration of two interpretations of the Monte Carlo estimator (Eq. (9.27)): computing the mean value (or height) of the function and multiplying by the interval length (top), or computing the average of several rectangle rules with random evaluation locations (bottom).

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 $\vec{x}_i$ are independent, we obtain that the expected value of this estimate $\langle f \rangle_M$ is the exact integral:

$$\mathbb{E}[\langle f \rangle_M] = \mathbb{E}\left[\frac{1}{M}\sum_{i=1}^{M} f(\vec{x}_i)\right] = \frac{1}{M}\sum_{i=1}^{M}\mathbb{E}[f(\vec{x}_i)] = \frac{1}{M}\sum_{i=1}^{M}\langle f \rangle = \frac{1}{M}M\langle f \rangle = \langle f \rangle. \tag{9.28}$$

Such estimators are called unbiased ("true") estimators. Furthermore, as we increase the number of samples M, the estimator $\langle f \rangle_M$ becomes a closer approximation of $\langle f \rangle$. Due to the Strong Law of Large Numbers, in the limit $M \to \infty$ we can guarantee that we have the exact solution:

$$\mathbb{P}\left\{\lim_{M\to\infty}\langle f \rangle_M = \langle f \rangle\right\} = 1. \tag{9.29}$$

9.3.1 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 the random estimate $\langle f \rangle_M$ and the exact deterministic integral $\langle f \rangle$,

$$\varepsilon_M = \sqrt{\mathrm{Var}[\langle f \rangle_M - \langle f \rangle]} \tag{9.30}$$

Using the expression from Eq. (9.21), the linearity of the expectation value and the fact that $\mathbb{E}[\langle f \rangle_M] = \langle f \rangle$ is a deterministic quantity, we can further simplify ε_M to

$$\varepsilon_M = \sqrt{\mathrm{Var}[\langle f \rangle_M]}. \tag{9.31}$$

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 (Eq. (9.21)), use the that $\langle \langle f \rangle_M \rangle = \langle f \rangle$ and that $f(\vec{x}_i)$ and $f(\vec{x}_j)$ are independent, if $i \neq j$:

$$\begin{aligned}
\varepsilon_M^2 = \mathrm{Var}[\langle f \rangle_M] &= \langle \langle f \rangle_M^2 \rangle - \langle \langle f \rangle_M \rangle^2 \\
&= \frac{1}{M^2} \sum_{i,j=1}^{M} \Big(\mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)] - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} \sum_{i=1}^{M} \Big(\mathbb{E}[f(\mathbf{x}_i)^2] - \langle f \rangle^2 \Big) + \frac{1}{M^2} \sum_{\substack{i,j=1 \\ i \neq j}}^{M} \Big(\underbrace{\mathbb{E}[f(\mathbf{x}_i) f(\mathbf{x}_j)]}_{= \mathbb{E}[f(\mathbf{x}_i)] \mathbb{E}[f(\mathbf{x}_j)] = \langle f \rangle^2} - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} \sum_{i=1}^{M} \Big(\langle f^2 \rangle - \langle f \rangle^2 \Big) \\
&= \frac{1}{M^2} M \Big(\langle f^2 \rangle - \langle f \rangle^2 \Big) = \frac{\langle f^2 \rangle - \langle f \rangle^2}{M} = \frac{\mathrm{Var}[f]}{M}.
\end{aligned} \tag{9.32}$$

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, the Monte Carlo method is 1/2-order accurate, i.e.

$$\varepsilon_M = \sqrt{\frac{\mathrm{Var}[f]}{M}} = \mathcal{O}(M^{-1/2}). \tag{9.33}$$

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 is *not always* satisfied:

$$|\langle f \rangle - \langle f \rangle_M| \leq \varepsilon_M. \tag{9.34}$$

Instead, ε_M describes the most probable values of the error, i.e. for large number of samples M, we expect

$$|\langle f \rangle - \langle f \rangle_M| < \begin{cases} \varepsilon_M, & \text{with probability of 68\%,} \\ 2\varepsilon_M, & \text{with probability of 95\%,} \\ 3\varepsilon_M, & \text{with probability of 99\%.} \end{cases} \tag{9.35}$$

9.3.2 Summary

Recipe for Monte Carlo Integration of uncorrelated data is as follows:

1. Sample points $\mathbf{x}_i$ from a 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

$$M, \quad \sum_{i=1}^{M} f(\mathbf{x}_i) \quad \sum_{i=1}^{M} f(\mathbf{x}_i)^2. \tag{9.36}$$

3. Compute the mean as the estimate of the expectation (normalized integral)

$$\frac{I}{|\Omega|} = \langle f \rangle \approx \langle f \rangle_M = \frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}_i). \tag{9.37}$$

4. Estimate the variance using the unbiased sample variance, which is given as follows

$$\mathrm{Var}[f] \approx \frac{M}{M-1} \left(\frac{1}{M} \sum_{i=1}^{M} f(\mathbf{x}_i)^2 - \langle f \rangle_M^2 \right) = \frac{M}{M-1} \Big(\langle f^2 \rangle_M - \langle f \rangle_M^2 \Big), \tag{9.38}$$

   and use it to estimate the error

$$\varepsilon_M = \sqrt{\frac{\mathrm{Var}[f]}{M}} \approx \sqrt{\frac{1}{M-1} \Big(\langle f^2 \rangle_M - \langle f \rangle_M^2 \Big)}. \tag{9.39}$$

9.4 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.,

$$\mathbb{E}_p[f] = \int f(x) p(x) \, \mathrm{d}x. \tag{9.40}$$

The Monte Carlo estimation of this expected value is

$$\mathbb{E}_p[f] \approx \frac{1}{M} \sum_{i=1}^{M} f(x_i) \tag{9.41}$$

where $\{x_i | i = 1, \ldots, M\}$ is a set of M samples drawn from a 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

$$I = |\Omega| \langle f \rangle = |\Omega| \mathbb{E}_p[f], \tag{9.42}$$

where

$$p(x) = \begin{cases} \frac{1}{|\Omega|}, & \text{if } x \in \Omega, \\ 0, & \text{otherwise.} \end{cases} \tag{9.43}$$

Hence, we would need to draw samples from the 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 numbers are available. For other problems where $p(x)$ is non-uniform, we would like to generate non-uniform random numbers to estimate the expected value $\mathbb{E}_p[f]$.

9.4.1 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

$$x = g(u) \tag{9.44}$$

For our uniform random variable U, one has that $p_U(u) = 1$ for $u \in [0, 1]$ and the CDF correspondingly reads

$$F_U(u) = \int_0^u p_U(s)\,\mathrm{d}s = u \tag{9.45}$$

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

$$F_X(x) = u \tag{9.46}$$

Thus, the following transformation is true

$$x = F_X^{-1}(u), \tag{9.47}$$

which means that the values of x are given by the inverse of the CDF $F_X(x)$, which 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

$$x^{(i)} = F_X^{-1}(u^{(i)}) \tag{9.48}$$

where $u^{(i)}$ are samples from the uniform distribution over interval $[0, 1]$.

Another (more formal) 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 $\vec{x} = g(\vec{u})$

$$\int_{g(U)} p(\vec{x})\,\mathrm{d}\vec{x} = \int_U p(\vec{u})|J(\vec{x})|^{-1}\,\mathrm{d}\vec{u}, \tag{9.49}$$

where J denotes the Jacobian and $g(U)$ the transformed domain.

Example 1

Sampling from the Exponential Distribution
Use the inverse transform sampling method to sample from the exponential distribution

$$p(x) = \lambda e^{-\lambda x}, x > 0 \tag{9.50}$$

The CDF of the exponential distribution is

$$F(x) = \int_0^x p(x)\,\mathrm{d}x = \int_0^x \lambda e^{-\lambda x}\,\mathrm{d}x = 1 - e^{-\lambda x} \tag{9.51}$$

Thus the transformation $x = g(u)$ which is obtained from $x = F^{-1}(u)$ or equivalently $F(x) = u$ is derived by solving

$$1 - e^{-\lambda x} = u \tag{9.52}$$

with respect to x to yield

$$x = -\frac{1}{\lambda}\ln(1-u) \tag{9.53}$$

The transformation (9.53) 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

$$x^{(i)} = -\frac{1}{\lambda}\ln(1-u^{(i)}) \tag{9.54}$$

where $u^{(i)}$, $i = 1, \ldots, N$ are samples drawn from the standard uniform distribution.

Example 2

***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

$$\begin{aligned} x_1 &= \sqrt{-2\ln(u_1)}\cos(2\pi u_2) \\ x_2 &= \sqrt{-2\ln(u_1)}\sin(2\pi u_2) \end{aligned} \tag{9.55}$$

To see that this indeed holds we invert the above equation and find

$$\begin{aligned} u_1 &= \exp\left(-\frac{x_1^2 + x_2^2}{2}\right) \\ u_2 &= \frac{1}{2\pi}\tan^{-1}\left(\frac{x_2}{x_1}\right) \end{aligned} \tag{9.56}$$

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

$$\frac{1}{2\pi}\iint\limits_{x_1^2+x_2^2\leq r^2} \exp\left(-\frac{x_1^2+x_2^2}{2}\right) \mathrm{d}x_1\,\mathrm{d}x_2 = \frac{1}{2\pi}\int_0^{2\pi}\int_0^r \exp\left(-\frac{r^2}{2}\right) r\,\mathrm{d}r\,\mathrm{d}\varphi = 1 - \exp\left(-\frac{r^2}{2}\right) \tag{9.57}$$

Setting the right hand side to be u_1 and solve for r we find the first factor of Eq. (9.55). 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

$$\begin{aligned} x_1 &= u_1\left(\frac{-2\ln(r^2)}{r^2}\right)^{1/2} \\ x_2 &= u_2\left(\frac{-2\ln(r^2)}{r^2}\right)^{1/2} \end{aligned} \tag{9.58}$$

Here we use the complex representation of points on a circle.

9.5 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

$$p(x) < \lambda h(x) \tag{9.59}$$

for some $\lambda \in \mathbb{R}$. For graphical explanation, refer to Fig. 9.3. 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 are generated.

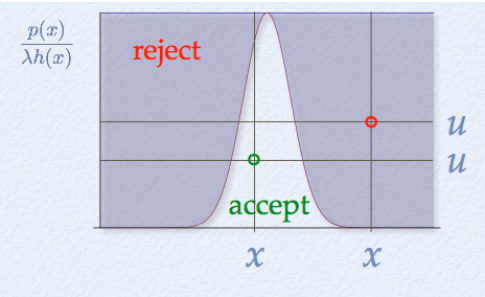


Figure 9.3: Rejection Sampling method

One of the easiest (not always the best or even appropriate) choice for $h(x)$ is a uniform distribution, as depicted in Fig. 9.3. In this case, the method is equivalent to the integration approach. However, 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.

9.6 *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 Fig. 9.4 (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.

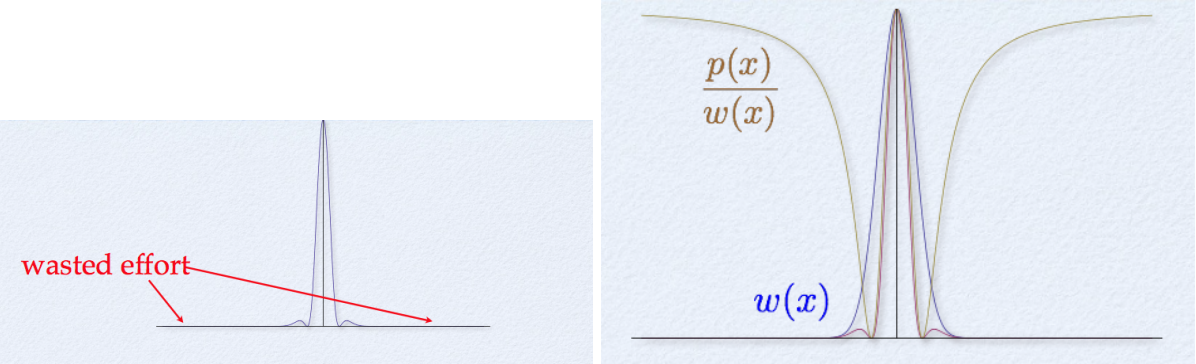


Figure 9.4: Left: Wasted effort while choosing sampling points in areas of low importance. Right: Illustration of usage of importance sampling to overcome difficulties arising from uniform sampling strategies.

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:

$$\langle f \rangle_p = \int_a^b f(x) \frac{p(x)}{w(x)} w(x) \, \mathrm{d}x \approx \frac{1}{M} \sum_{i=1}^{M} f(x_i) \frac{p(x_i)}{w(x_i)}, \tag{9.60}$$

with each x_i being sampled from distribution $w(x)$. For an example distribution $w(x)$, refer to Fig. 9.4 (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 10 Neural Networks

The neural network is a function $\boldsymbol{y}$ from $\mathbb{R}^{n_0}$ to $\mathbb{R}^{n_L}$ parametrized by weight parameters $\boldsymbol{w}$, i.e., $\boldsymbol{y}(\cdot;\boldsymbol{w}) : \mathbb{R}^{n_0} \to \mathbb{R}^{n_L}$. This is not the first time that we come across a function parameterized with weights. The fitting function of the non-linear least squares method is given as a sum of weights times a function of the input data: $\boldsymbol{y}(\boldsymbol{x};\boldsymbol{w}) = \sum_{i=1}^{N} w_i \varphi_i(\boldsymbol{x})$.

10.1 2-layer neural network

The function $\boldsymbol{y}$ has a very specific form that is usually presented by a graph. Such a function is described in Fig. 10.1, where the presented neural network has one input layer, where $\boldsymbol{x} \in \mathbb{R}^{n_0}$, one hidden layer, where $\boldsymbol{z} \in \mathbb{R}^{n_1}$, and an output layer, where $\boldsymbol{y} \in \mathbb{R}^{n_2}$, i.e. three layers in total and two apart from the input layer. The neural network is constructed with all-in-all connections, i.e. all nodes from one layer are being parsed as inputs to every node of the next layer. The notation for the weights applied to every connection w_{ji}^{ℓ} is the following: the superscript ℓ denotes the layer level $\ell = 1, 2$, whereas the subscript ji denotes the destination node j and the source node i. The a_j^{ℓ} are the inputs to every node j of the layer ℓ and they consist of the weighted contributions from all nodes of the previous layer.

How do we pass information from one layer to the other?

For the first layer,

$$a_j^1 = \sum_{i=1}^{n_0} w_{ji}^1 \, x_i + w_{j0}^1 \, . \tag{10.1}$$

We include the constant term (bias in literature) in the summation by extending the input adding the element $x_0 = 1$,

$$a_j^1 = \sum_{i=0}^{n_0} w_{ji}^1 \, x_i \, . \tag{10.2}$$

The output of the first layer,

$$z_j^1 = \varphi_1(a_j^1) \, , \tag{10.3}$$

where φ_1 is function from $\mathbb{R}$ to $\mathbb{R}$ called the *activation function* for layer 1.

For the second layer,

$$a_j^2 = \sum_{i=0}^{n_1} w_{ji}^2 \, z_i^1 \, . \tag{10.4}$$

The output of the second layer, which is also the output of the neural network

$$y_j = z_j^2 = \varphi_2(a_j^2) \, , \tag{10.5}$$

where φ_2 is an activation function, not necessary the same as φ_1.

We can write the output as a function of the input of the neural network, taking into consideration the parsed information from all layers

$$y_j(\boldsymbol{x};\boldsymbol{w}) = \varphi_2\left(\sum_{i=0}^{n_1} w_{ji}^2\ \varphi_1\left(\sum_{k=0}^{n_0} w_{ik}^1\, x_k\right)\right), \quad j = 1,\ldots,n_2\,. \tag{10.6}$$

If we define W^ℓ to be the matrix with elements $W_{ij}^\ell = w_{ij}^\ell$ for $\ell = 1, 2$, then the neural network output can be written as

$$\boldsymbol{y}(\boldsymbol{x};\boldsymbol{w}) = \varphi_2\left(W^2\,\varphi_1\left(W^1\boldsymbol{x}\right)\right), \tag{10.7}$$

where the activation functions act elementwise on the vectors.

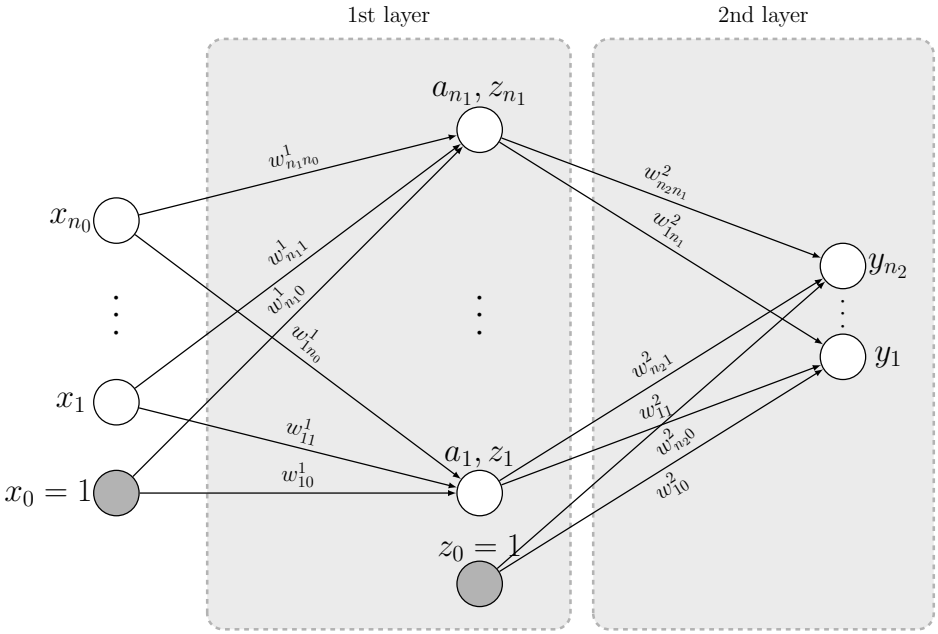


Figure 10.1: Schematic graph of a feedforward neural network with 2 layers. The dimension of the input, hidden and output layer is n_0,n_1 and n_2, respectively.

10.2 L-layer neural network

The neural network described in the previous section had only one intermediate (hidden) layer, but this is not the norm. Generalizing, a neural network can be comprised of a number of intermediate layers. The graph of a generalized neural network is given in Fig. 10.2, where the same notation as explained above is used.

The output of a generalized neural network can be written as a function of the network's input, by parsing information from each layer to the next in the same way as described for the simplified 2-layer network in the previous section.

Let W^ℓ be the weight matrix for every layer, with elements $W^\ell_{ij} = w^\ell_{ij}$ for $\ell = 1, \ldots, L$. The neural network output can be written as

$$\boldsymbol{y}(\boldsymbol{x}; \boldsymbol{w}) = \varphi_L \left(W^L \, \varphi_{L-1} \left(W^{L-1} \, \varphi_{L-2} \left(\cdots W^2 \varphi_1 \left(W^1 \boldsymbol{x} \right) \right) \right) \right) . \tag{10.8}$$

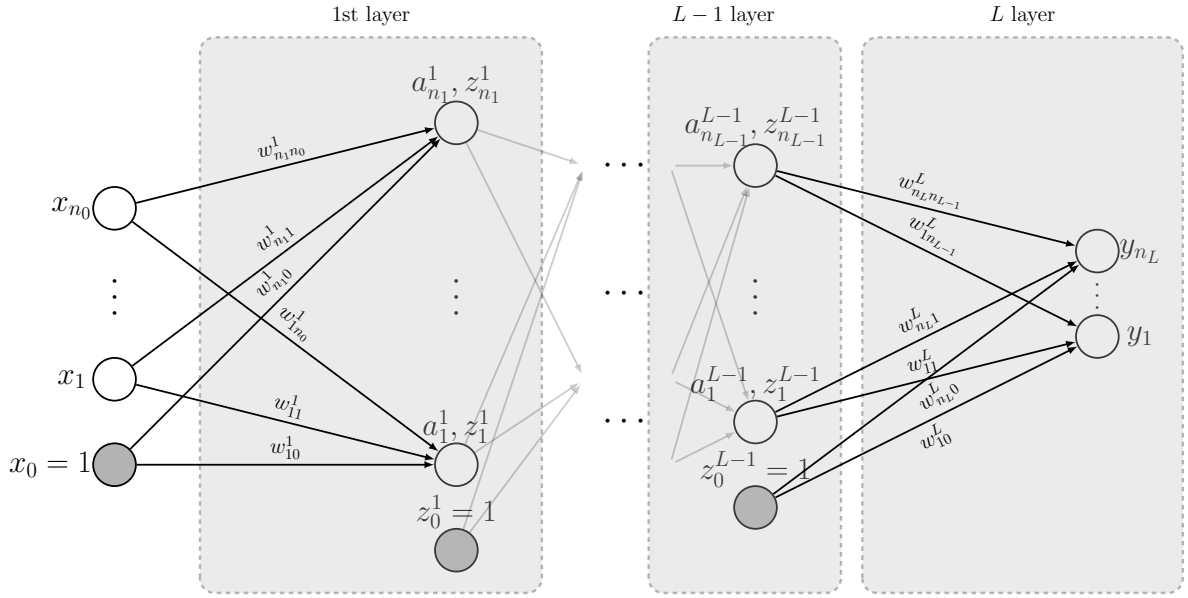
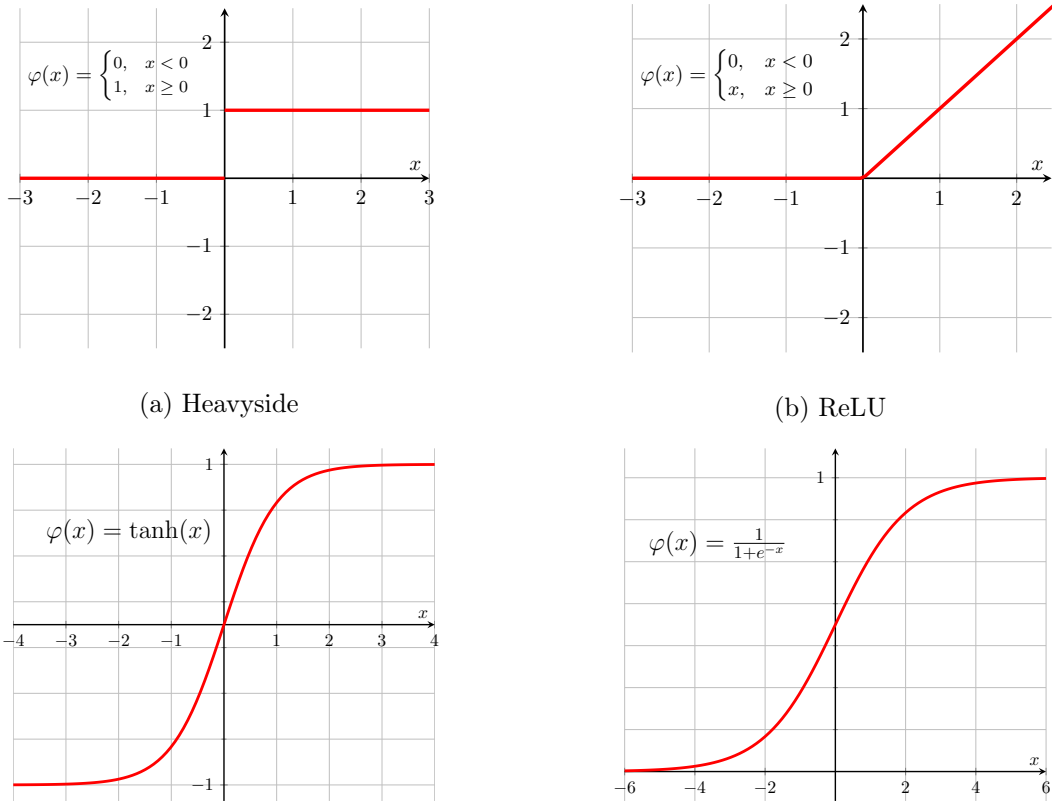


Figure 10.2: Schematic graph of an L layer neural network.

In the above construction we have chosen 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.



(a) Heavyside

(b) ReLU

(c) $\tanh(x)$

(d) logistic

Figure 10.3: Examples of activation functions.

10.3 Activation functions

As mentioned before, the activation function of each layer connects the output of the layer with the layer input, i.e. the weighted combination of the outputs of all nodes of the previous layer. The activation functions between layers can be different.

If the activation functions of all layers of the neural network were linear, the output of the neural network would be simply a linear combination of the network input. This can be shown, simply by substituting all φ_ℓ for $k = 1, \ldots, L$ as a linear function in Eq. (10.8). However, usually, the activation functions are not linear. In the original version of a neural network the Heaviside step function was proposed as an activation function. Later many others were considered, which can be either step- non-globally differentiable functions or smooth globally differentiable functions, with the most prominent ones being the ReLU (Rectified Linear Unit), the hyberbolic tangent and the logistic function, listed in Fig. 10.3. In modern architectures, the preferred choice are smooth functions, such as the logistic or hyberbolic tangent functions.

10.4 Learning

Training the neural network is often called learning and it simply translates to acquiring the weight matrix for all layers of the NN. The application we will apply the learning algorithms for NN is regression. We would like to train the NN on data $d_n = \{\boldsymbol{x}_n, \hat{\boldsymbol{y}}_n\}$, where $n = 1, \ldots, N$ and N is the size of the training data. We aim to optimize the global weights $\boldsymbol{w} = \{W^1, W^2, \ldots, W^L\}$, so that the error between the output of the neural network and the observations is minimal.

For each pair $\{\boldsymbol{x}_n, \hat{\boldsymbol{y}}_n\}$, the local error of the NN output is defined as the L2-error for the problem of regression[1],

$$E_n(\boldsymbol{w}) = \frac{1}{2}|\hat{\boldsymbol{y}}_n - \boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w})|^2 \,, \tag{10.9}$$

where $\boldsymbol{y}(\boldsymbol{x}_n, \boldsymbol{w})$ is the output of the NN for a certain input $\boldsymbol{x}_n$, given weights $\boldsymbol{w}$ and $\hat{\boldsymbol{y}}_n$ is the corresponding observation in the data set.

The cost function E for all the weights in the network is given by the global error

$$E(\boldsymbol{w}) = \sum_{n=1}^{N} E_n(\boldsymbol{w}) \,. \tag{10.10}$$

The learning procedure concludes with a set of weights (parametes) that minimize the global error, i.e.

$$\boldsymbol{w}^\star = \arg\min E(\boldsymbol{w}) \,. \tag{10.11}$$

The numerical scheme we are going to use is a *gradient descent* (GD) method, where derivatives of the cost function E with respect to the weights $\boldsymbol{w}$ are used for the update of the parameters.

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} E(\boldsymbol{w}^{(k)}) \,, \tag{10.12}$$

where the superscript $k = 1, 2, \ldots$ denotes the iteration index, η is a parameter called *learning rate*. The algorithm begins with the weights for the initial iteration $\boldsymbol{w}^{(1)}$ given. The algorithm finds local minima, therefore the end-result might vary with the choice of the initial weights[2].

In the gradient descent update of minima, we utilize the gradient of the overall cost function (the error E) over the parameters to be optimized (the weights $\boldsymbol{w}$). Often, we use an alternative of this, which is called *stochastic gradient descent* (SGD). In the SGD algorithm, the update of the weights in each iteration does not depend on the derivative of the global error, but rather on one of the local errors E_n, related to pair $\{\boldsymbol{x}_n, \hat{\boldsymbol{y}}_n\}$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} E_n(\boldsymbol{w}^{(k)}) \,. \tag{10.13}$$

[1] The L2-error was also used in the procedure of regression with Least Squares. In general, the error functions used for the optimization process depends on the problem type.

[2] The intuition behind Eq. (10.14) comes from a particle dynamics setting. In particle notation, if $\boldsymbol{w}$ is the vector of position and E is the potential, the gradient of the potential corresponds to the velocity of the particle $\dot{\boldsymbol{w}} = -\nabla E(\boldsymbol{w})$. The solution of this ODE returns a stable local minimum of the particle

The choice of the E_n for each iteration can be done: (i) sequentially or even (ii) randomly.

A method in between the GD and the SGD is the so-called *batch stochastic gradient descent* (batchSGD). In this case, the gradient for the solution update is applied neither on the global error, nor on one local error for each iteration. The sum is computed on a subset $\mathcal{I} \subset \{1, 2, \ldots, N\}$,

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} \sum_{n \in \mathcal{I}} E_n(\boldsymbol{w}^{(k)}) \,, \tag{10.14}$$

The set $\mathcal{I}$ can be chosen randomly, with or without replacement.

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 too low, the algorithm will need maybe too many iterations to reach the minimum. On the other hand, if the learning rate is too high, we might fail to reach the minimum and oscillate between suboptimal values (see Fig. 10.4). The right learning rate is one of the crucial hyper-parameters to be dealt with in deep learning.

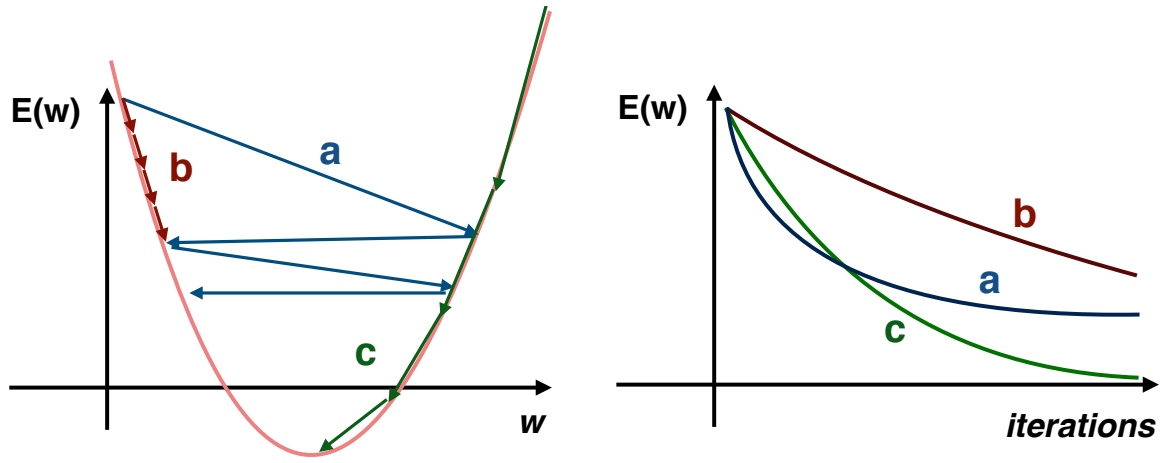


Figure 10.4: The relationship between the error and the number of iterations for different choices of the learning rate η. Case (a) corresponds to a very high learning rate, case (b) to a very slow η and case (c) to a desired value of η.

10.5 Backpropagation

As we saw in the previous section, for the update of the weights of the neural network we need the derivatives of errors of the NN. A way to efficiently minimize the cost function is the so-called **back-propagation method**, i.e., by computing the derivative of E with respect to the each of the weights w_{ji}^{ℓ}, using the chain rule. In order to make our derivation more general, for each layer of the NN, we use a semi-local representation of a node, with its input coming from a single previous node of output $\tilde{\tilde{z}}_i$, given weight w_{ji} and its output z_i is parsed into all nodes of the next

layer, given weights $\tilde{w}_{mj}$, $m = 1, \dots, k$, as seen in Fig. 10.5. Then we can drop the superscript of the weights and the computation of the gradient $\nabla_{\boldsymbol{w}} E_n(\boldsymbol{w})$ shrinks to $\frac{\partial E_n}{\partial w_{ji}}$.

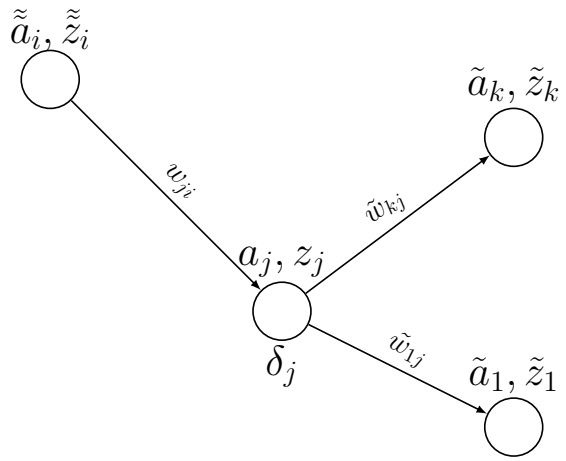


Figure 10.5: The local neighborhood of a node in a general neural network

The calculation of the gradient of the error $\frac{\partial E_n}{\partial w_{ji}}$ proceeds with applying the chain rule

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j \, \tilde{\tilde{z}}_i \,, \tag{10.15}$$

since $z_j = \varphi(a_j)$ and $a_j = \sum_k w_{jk} \tilde{\tilde{z}}_k$. The values of all $\tilde{\tilde{z}}_i, z_i, \tilde{z}_i$ (the output values $\boldsymbol{z}$ and the weighted input $\boldsymbol{a}$ for each layer) can be computed in a previous step, with a forward propagation of the NN.

The δ_i corresponds to the derivative of the error with respect to the weighted input of the node and its computation requires another application of the chain rule,

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial \tilde{a}_k} \frac{\partial \tilde{a}_k}{\partial a_j} = \sum_k \tilde{\delta}_k \, \frac{\partial \tilde{a}_k}{\partial a_j} \,, \tag{10.16}$$

where $\tilde{a}_k = \sum_j \tilde{w}_{kj} z_j = \sum_j \tilde{w}_{kj} \varphi(a_j)$. Therefore, using another chain rule we can compute

$$\frac{\partial \tilde{a}_k}{\partial a_j} = \varphi'(a_j) \tilde{w}_{kj},$$

where $\varphi'(a_j)$ the derivative of the activation function of the node. Finally, we compute

$$\delta_j = \varphi'(a_j) \sum_k \tilde{w}_{kj} \tilde{\delta}_k \,. \tag{10.17}$$

We see that the computation of δ_j depends on the $\tilde{\delta}_k$'s of the nodes of the next layer. In this way, for every layer, the computation of the gradient of the error depends on the layer on its right.

If we now move all the way to the last layer of the NN, with input z_i, weight w_{ji} and output y_j, and we follow the same procedure,

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} = \delta_j \, z_i \, . \tag{10.18}$$

The computation of δ_j proceeds, assuming that at the last layer we have a linear activation function, i.e. $a_j = y_j$

$$\delta_j = \frac{\partial E_n}{\partial a_j} = \frac{\partial}{\partial y_j}\frac{1}{2}||\boldsymbol{y}(\boldsymbol{x}_n;\boldsymbol{w}) - \hat{\boldsymbol{y}}_n||^2 = \frac{\partial}{\partial y_j}\frac{1}{2}\sum_k \left(y_k(\boldsymbol{x}_n;\boldsymbol{w}) - \hat{y}_{ni}\right)^2 = y_j(\boldsymbol{x}_n;\boldsymbol{w}) - \hat{y}_{nj} \, , \tag{10.19}$$

where $y_j(\boldsymbol{x}_n;\boldsymbol{w})$ is the $j-th$ output of the NN and $\hat{\boldsymbol{y}}_{nk}$ the $k-$th element of the $n-$th observation data. Here, k is a running index: the index surviving the differentiation is where $k = j$. We see that, since now there is no layer on the right, it is possible to express the gradient of the error exactly as the local error.

We observe that at the last layer, the computation of the gradients $\frac{\partial E_n}{\partial w_{ji}}$ and $\frac{\partial E_n}{\partial a_j}$ do not depend on the neural network. Therefore, the essence of the back-propagation algorithm, is that we first calculate the δ_j at the last layer and we then back-propagate to acquire the δ_j's at every previous layer.

10.6 Overfitting

Every model with N free parameters should be able to fit exactly N data points. When passing trough all the data points, however, the model is likely to fit behavior of noise and, as a consequence, 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.

For the example of regression with neural networks we were dealing with in the previous sections, if the data we are trying to fit $\{\boldsymbol{x}_n, \hat{\boldsymbol{y}_n}\}$ represents a linear relation, the usage of linear squares results in a non-zero error between the fit and the results, but the model achieves a good generalization over the rest of the domain. On the other hand, if we fit the data using a higher order polynomial, we end up with zero error of the fitting curve and the data, but with bad generalization.

This behaviour of overfitting can be avoided, if we require the error of data and fit always to be higher than zero. We can achieve this, by splitting our data set in two subsets: (a) a training set, which consists of, say, 90% of the data and (b) a test set, which consists in that case of 10% of the data. **Both** sets are used in the SGD algorithm, i.e the error in every iteration is computed using both the training and test sets. If we plot the evolution of the error over the course of

iterations of the SGD algorithm, we observe that, although the error for the training set is being continuously reduced, the error for the test set will reduce and after one point increase again. This point, where both errors are kept low enough is selected to be the termination point of the algorithm (see Fig. 10.6). We note that, in order to make our result statistically independent of the choice of training and test sets, we normally follow the same procedure multiple times, each time selecting a different training and test set. In the end, the termination point of the algorithm is determined from the curve of the mean error over all the realizations.

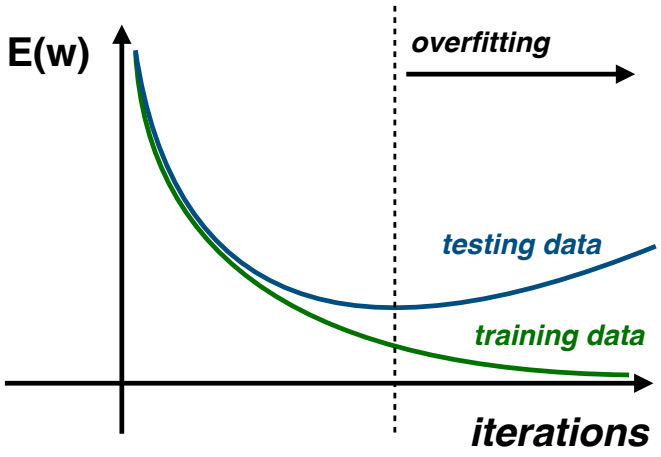


Figure 10.6: The error evaluation of the SGD algorithm for the training and test sets over the course of iterations. Over the point denoted with the dashed line, the algorithm is overfitting the data. The dashed line denotes the termination point of the algorithm

Exam checklist

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

LECTURE 11

Dimensionality reduction

When dealing with large datasets, it is beneficial to substitute the original set with a smaller dataset, that still captures the original's characteristics. This process reduces required storage and processing time of the data set. Usual applications are in the field of big data, such as recommendation systems, quantitive finance, medical data analysis or image processing. We are first going to present Principal component analysis (PCA), a powerful tool based on linear algebra. PCA can be used to extract essential features or structures of a data set that can not readily be observed. Then we will see how the same goal can be achieved using neural networs, namely Autoencoders. We will see that this also allows to readily generalize the dimensionality reduction technique to non-linear problems which are hard to tackle with PCA.

11.1 Principal component analysis

Principal component analysis is a linear transformation, projecting data $\mathbf{x}_n \in \mathbb{R}^D$ for $n = 1, \dots, N$ onto the subspace explaining most of the variance, respectively minimizes the reconstruction loss. This lower dimensional representation is called latent representation. In the following, we want to describe two ways to understand this process; the first being the Maximum Variance formulation and the second the Minimal Error formulation. We will then conclude with the general procedure to obtain the principal components and show extensions to capture non-linear problems using Kernel PCA.

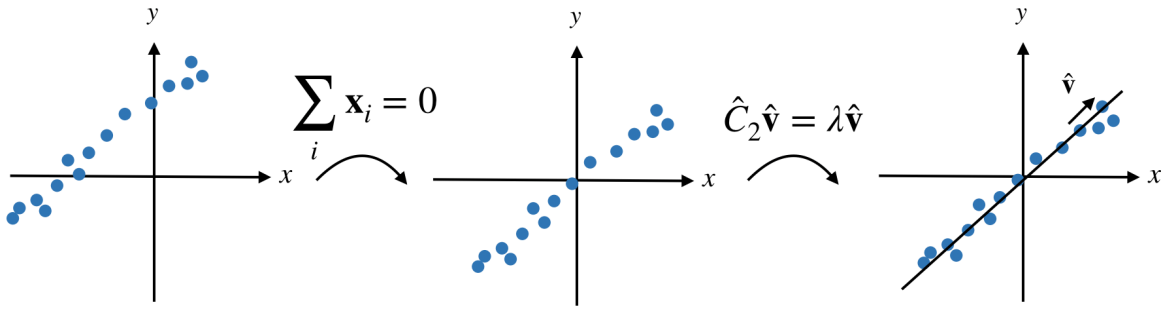


Figure 11.1: Sketch showing the principal workings of principal component analysis. We bring the data set into the mean-zero form.Then PCA derives the principal axis of information of the data.

In order to set the stage, assume we are given a dataset $X^\top = (\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathbb{R}^{D \times N}$ with N

vectors with D elements

$$\mathbf{x}_n \in \mathbb{R}^D, \quad n = 1, 2, \ldots, N \tag{11.1}$$

PCA assumes that the data has zero mean (i.e. $\sum \mathbf{x}_n = 0$). Otherwise we transform the data, such that this property is satisfied

$$\tilde{\mathbf{x}}_n = \mathbf{x}_n - \bar{\mathbf{x}}\,, \quad \text{where} \quad \bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n\,,\ n = 1, 2, \ldots, N \tag{11.2}$$

As we will see, PCA rotates the data in such a way that the new axes are aligned to the direction with the most spread in the variance. As we will further see, PCA gives us a systematic way to project the data in a way that minimizes the resulting error when transformin back to the original representation.

11.1.1 Maximum Variance Formulation

We want to find the first principal component, that is the direction in $\mathbf{v}_1^\star \in \mathbb{R}^D$, so that the variance of the data projected to this vector is maximized. The variance of the projection of the data σ_1^2 in the direction $\mathbf{v}_1$ is given by

$$\begin{aligned} \sigma_1^2 = \frac{1}{N-1}\sum_{n=1}^{N}(\mathbf{x}_n^\top \mathbf{v}_1)^2 &= \frac{1}{N-1}\sum_{n=1}^{N}\mathbf{v}_1^\top \mathbf{x}_n \mathbf{x}_n^\top \mathbf{v}_1 \\ &= \mathbf{v}_1^\top \frac{1}{N-1}\sum_{n=1}^{N}\mathbf{x}_n \mathbf{x}_n^\top \mathbf{v}_1 \\ &= \mathbf{v}_1^\top C \mathbf{v}_1 \end{aligned} \tag{11.3}$$

Here, we introduce $C \in \mathbb{R}^{D\times D}$, which is the sample covariance matrix

$$C = \frac{1}{N-1}\sum_{n=1}^{N}\mathbf{x}_n \mathbf{x}_n^\top \tag{11.4}$$

We, thus, conclude that the property characterizing the first principal component is given by

$$\mathbf{v}_1^\star = \underset{\mathbf{v}_1}{\operatorname{argmax}}\ \mathbf{v}_1^\top C \mathbf{v}_1 \tag{11.5}$$

However, this form does not translate to a well-posed problem, as we realize that a vector with $\|\mathbf{v}_1\| \to \infty$ will be a valid solution. To avoid this, we restrict our attention to unit vectors $\mathbf{v}_1^\top \mathbf{v}_1 = 1$. This results in a constraint optimization problem, which can be solved using Lagrangian multipliers. The resulting objective function to be maximized is

$$\mathcal{L}(\mathbf{v}_1, \lambda_1) = \mathbf{v}_1^\top C \mathbf{v}_1 - \lambda_1\big(\mathbf{v}_1^\top \mathbf{v}_1 - 1\big). \tag{11.6}$$

We can compute the maximum of this function by setting the first derivative to zero

$$\frac{\partial \mathcal{L}(\mathbf{v}_1, \lambda_1)}{\partial \mathbf{v}_1}\Big|_{\mathbf{v}_1^\star, \lambda_1^\star} = 0 \quad \Longrightarrow \quad C\mathbf{v}_1^\star = \lambda_1^\star \mathbf{v}_1^\star \tag{11.7}$$

We realize that the projection that maximizes the variance $\mathbf{v}_1^\star$ corresponds to an eigenvector of the data covariance matrix C. The covariance matrix C is positive definite, therefore its eigenvalues are all positive, therefore, $\lambda_1 \geq 0$. Further we find, that

$$\sigma_1^{\star 2} = \mathbf{v}_1^{\star\top} C \mathbf{v}_1^\star = \lambda_1^\star \mathbf{v}_1^{\star\top} \mathbf{v}_1^\star = \lambda_1^\star, \tag{11.8}$$

Hence, the maximum of the objective function corresponds to the eigenvector that corresponds to the maximum eigenvalue.

We can now continue by computing the second principal component by looking for a vector $\mathbf{v}_2$, that is orthonormal to $\mathbf{v_1}$ and maximizes the residual variance when projecting along this new axis. It can be shown that the second principle axis is the eigenvector corresponding to the second largest eigenvalue $\lambda_2 < \lambda_1$ of the covariance matrix. This procedure is then continued and we find that the principal components correspond to the eigenvectors of the covariance matrix sorted by the size of their eigenvalue. Formally, we can write the whole procedure as

$$\underset{\mathbf{v}_1,\ldots,\mathbf{v}_D,\ \mathbf{v}_i^T\mathbf{v}_j=\delta_{ij}}{\operatorname{argmax}} \quad \mathbf{v}_1^T C \mathbf{v}_1 + \mathbf{v}_2^T C \mathbf{v}_2 + \cdots + \mathbf{v}_D^T C \mathbf{v}_D \tag{11.9}$$

11.1.2 Minimum Error Formulation

Another way to understand the PCA method is the so-called minimum error formulation. Here, we introduce a complete orthonormal set of D-dimensional basis vectors $\{\mathbf{v}\}_{i=1,2,\ldots,N}$. Recall that orthonormal means that the vectors satisfy

$$(\mathbf{v}_i)^T \mathbf{v}_j = \delta_{ij} = \begin{cases} 1\,, & i = j \\ 0\,, & \text{else} \end{cases} \tag{11.10}$$

Furthermore a basis is called complete if any vector $\mathbf{v} \in \mathbb{R}^D$

$$\mathbf{v} = \sum_{i=1}^{D} \alpha_i \mathbf{v}_i, \tag{11.11}$$

where $\alpha_i \in \mathbb{R}$ are the components in the new basis.

We want to use this basis to rewrite our data

$$\mathbf{x}_n = \sum_{i=1}^{D} \alpha_{ni} \mathbf{v}_i \tag{11.12}$$

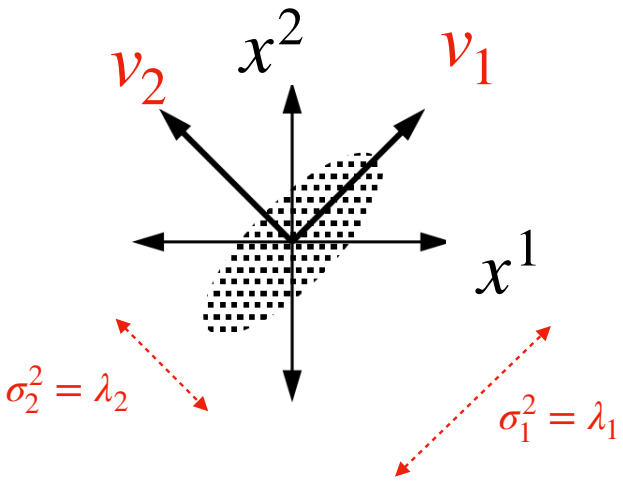


Figure 11.2: The eigenvalues of the covariance matrix represent the variance captured by the PCA component.

Note that this corresponds to a change of the coordinate system. We find that we can compute the coefficients by taking an inner product with $\mathbf{v}_j$:

$$\mathbf{x}_n^T \mathbf{v}_i = \sum_{j=1}^{D} \alpha_{nj} \mathbf{v}_j^\top \mathbf{v}_i = \sum_{j=1}^{D} \alpha_{nj} \delta_{ij} = \alpha_{ni} \tag{11.13}$$

This way, we can obtain our data vectors as

$$\mathbf{x}_n = \sum_{i=1}^{D} (\mathbf{x}_n^\top \mathbf{v}_i) \mathbf{v}_i \tag{11.14}$$

Let us now assume that for each of the basis vectors we assume we only take the first $M \leq D$ terms of this expansion and add a residual with coefficients β_i for $i = M+1, \ldots, D$ independent of the data-point to complement this incomplete expansion

$$\tilde{\mathbf{x}}_n = \sum_{i=1}^{M} \alpha_{ni} \mathbf{v}_i + \sum_{i=M+1}^{D} \beta_i \mathbf{v}_i \tag{11.15}$$

We now aim to minimize the distortion created by this expansion, i.e. the mean square error between the original data and the expansion,

$$J(\{\beta_i\}_{i=M+1,\ldots,D}) = \sum_{n=1}^{N} (\mathbf{x}_n - \tilde{\mathbf{x}}_n)^2. \tag{11.16}$$

If we substitute the expansion for $\mathbf{x}_n$ and $\tilde{\mathbf{x}}^n$ we get,

$$J = \sum_{n=1}^{N} \left(\sum_{i=M+1}^{D} (\alpha_{ni} + \beta_i)\mathbf{v}_i \right)^2 \tag{11.17}$$

$$= \sum_{n=1}^{N} (\alpha_{ni} + \beta_i)^2, \tag{11.18}$$

where we used the orthogonality condition $\mathbf{v}_i^\top \mathbf{v}_j = \delta_{ij}$. Taking the derivative with respect to β_i we find

$$\left.\frac{\partial J}{\partial \beta_i}\right|_{\beta_i^\star} = 2\sum_{n=1}^{N} \alpha_{ni} + 2N\beta_i^\star \stackrel{!}{=} 0 \quad \Longrightarrow \quad \beta_i^\star = \frac{1}{N}\sum_{n=1}^{D} \alpha_{ni} = \frac{1}{N}\sum_{n=1}^{D} \mathbf{x}_n^\top \mathbf{v}_i = \bar{\mathbf{x}}^\top \mathbf{v}_i \tag{11.19}$$

Plugging in the found optimum we find that we can write

$$J\left(\{\beta_i^\star\}_{i=M+1,\ldots,D}\right) = \frac{1}{N}\sum_{n=1}^{N}\sum_{i=M+1}^{D} [(\mathbf{x}_n - \bar{\mathbf{x}})^\top \mathbf{v}_i]^2 \tag{11.20}$$

$$= \sum_{i=M+1}^{D} \mathbf{v}_i^\top \frac{1}{N}\sum_{n=1}^{N} (\mathbf{x}_n - \bar{\mathbf{x}})(\mathbf{x}_n - \bar{\mathbf{x}})^\top \mathbf{v}_i \tag{11.21}$$

$$= \sum_{i=M+1}^{D} \mathbf{v}_i^\top C \mathbf{v}_i \tag{11.22}$$

We realize that we arrive at a similar result as before, but on a different notation. Here, we conclude that we minimize the mean square error of the expansion, by taking $\mathbf{v}_i$ for $i = M+1, \ldots, N$ to be the eigenvectors of the covariance matrix which are associated with the smallest eigenvalues. This is equivalent to the previous statement, namely that the principal subspace $\mathbf{v}_i$ for $i = 1, \ldots, M$ is spanned by the eigenvectors associated to the largest eigenvalues of the covariance matrix.

11.1.3 Computation of the Principal Components

To summarize, we want to perform a transformation $\mathbf{y}_n = V^T \mathbf{x}_n$ of a sample data set $X = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, where $\mathbf{x}_n \in \mathbb{R}^D$ using the principal components $V = (\mathbf{v}_1, \ldots, \mathbf{v}_D) \in \mathbb{R}^{D\times D}$ and $\mathbf{v}_i^T \mathbf{v}_j = \delta_{ij}$. The sample set is considered in the mean deviation form i.e. $\frac{1}{N}\sum_{n=1}^{N} \mathbf{x}^n = 0$. The Principal Component Analysis then consists of the following steps:

1. Construct the centered data matrix $X \in \mathbb{R}^{N\times D}$
2. Construct the covariance matrix $C = \frac{1}{N-1} X^T X$, $C \in \mathbb{R}^{D\times D}$
3. Perform an eigenvector decomposition of the covariance matrix,

$$C = V\Lambda V^{-1}, \quad \Lambda = \operatorname{diag}\left(\left\{\lambda_i | i \in \{1, \ldots, D\}\right\}\right), \quad V^{-1} = V^T \in \mathbb{R}^{D\times D} \tag{11.23}$$

4. Sort the eigenvectors in decreasing eigenvalue order $\lambda_1 \geq \lambda_2 \geq \dots \lambda_D$ to get the transformation $\mathbf{y}^n = V^T \mathbf{x}^n$ where $V = (\mathbf{v}_1, \dots, \mathbf{v}_D)$ are the sorted eigenvectors, with $\mathbf{v}_i$ the eigenvector corresponding to the eigenvalue λ_i and $\mathbf{v}_i^T \mathbf{v}_j = \delta_{i,j}$

The PCA algorithm depends on the eigenvector decomposition of the covariance matrix. For an $D \times D$ matrix the associated cost for this operation is $\mathcal{O}(D^3)$. If only the first principal component is of interest, a famous method to compute the largest eigenvalue and the associated eigenvector is called the *power method*. However, its convergence is only linear and depends on the factor $|\lambda_2/\lambda_1|$, that is, the convergence is very slow for cases where the largest eigenvalue is close to the second largest eigenvalue. If the complete solution to the eigenvalue problem is required, eigenvalue-revealing factorizations are usually applied, such as Schur factorization.

11.1.4 Dimensionality reduction

Until now we have seen how to transform the dataset $X = (\mathbf{x}_1, \dots, \mathbf{x}_N) \in \mathbb{R}^{D \times N}$ into a new coordinate system, giving $Y = (\mathbf{y}_1, \dots, \mathbf{y}_N) \in \mathbb{R}^{D \times N}$ using the principal components $V = (\mathbf{v}_1, \dots, \mathbf{v}_D) \in \mathbb{R}^{D \times D}$

$$Y = V^\top X \tag{11.24}$$

Now we want to see how we can use this method to compress the original dataset. Therefore we restrict the tranformation matrix V by only using the first $r < D$ principal components, giving $V_r = (\mathbf{v}_1, \dots, \mathbf{v}_r) \in \mathbb{R}^{D \times r}$. This allows us to compute a compressed representation of the data $Y_r \in \mathbb{R}^{r \times N}$

$$Y_r = V_r^\top X \tag{11.25}$$

We can reconstruct the data using

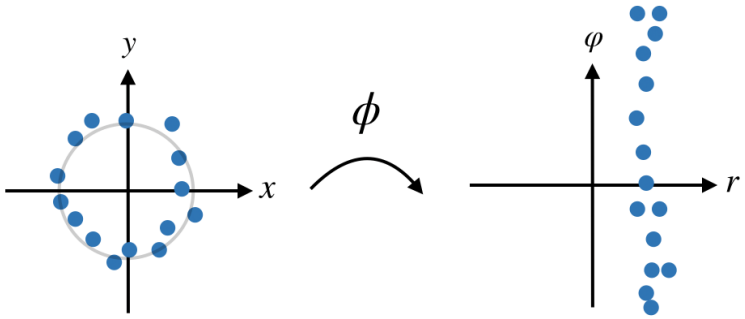
$$\tilde{X} = V_r Y_r = V_r V_r^\top X \tag{11.26}$$

As we have seen in the previous sections, this minimizes the reconstruction loss and maximizes the retained variance. The percentage of retained variance can be computed as

$$\text{percentage of retained variance} = \frac{\sum_{i=1}^{r} \lambda_i}{\sum_{i=1}^{D} \lambda_i} \tag{11.27}$$

11.1.5 Nonlinear Problems / Kernel PCA

Consider the simple nonlinear problem of data distributed on a circle. Clearly in Cartesian coordinates the task of finding a subspace which gives a small error when projecting onto it is not trivial to solve. Even more, it yields infinitely many equally good (resp. bad) solutions. If we however transform the problem into polar coordinates it is again easy to solve. This insight can



be introduced into PCA using Kernel functions ϕ, the usage of which transforms the data into feature space. In feature space, we assume that the data is centered, i.e.

$$\sum_{i=1}^{N} \phi(\mathbf{x}_i) = 0 \tag{11.28}$$

and we redefine the covariance matrix as

$$C = \frac{1}{N} \sum_{i=1}^{N} \phi(\mathbf{x}_i)\phi(\mathbf{x}_i)^\top \tag{11.29}$$

The PCA problem is now the same for the modified covariance matrix. Although this solves the problem of analyzing non-linear problems we have to hand-craft the kernel function, which might not be easy. A way out is offered in the following section, where we solve the problem using neural networks.

11.2 Auto-associative NN

We have seen that the method of PCA can be used for dimensionality reduction by mapping the input, i.e. the original data $\mathbf{x}_n \in \mathbb{R}^D$ to a compressed representation of the data $\mathbf{y}_n \in \mathbb{R}^r$ for $r < D$. We have seen that the compressed space is computed as $\mathbf{y}_n = V_r^\top \mathbf{x}_n$. The matrix V_r, contains the first r principal components that minimize the mean square error between the input and the projection of the output onto the overall space

$$\underset{V_r=(\mathbf{v}_1,\ldots,\mathbf{v}_r),\ \mathbf{v}_i^T\mathbf{v}_j=\delta_{ij}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \left|\left| \mathbf{x}^n - V_r \underbrace{V_r^T \mathbf{x}_n}_{\mathbf{y}_n} \right|\right|_2^2. \tag{11.30}$$

This can be translated into the formalism we learned in the previous chapter on Neural Network. Therefore consider a Neural Network (NN) mapping the input $\mathbf{x}_n \in \mathbb{R}^D$ onto an output $\tilde{\mathbf{x}}^n \in \mathbb{R}^D$ through an intermediate feature space $\mathbf{y}_n \in \mathbb{R}^r$ using a matrix $W \in \mathbb{R}^{r\times D}$

$$\mathbf{y}_n = W\mathbf{x}_n, \qquad \tilde{\mathbf{x}}_n = W^T\mathbf{y}_n. \tag{11.31}$$

The NN is comprised then from one Encoder-Layer, compressing the input $\mathbf{x}_n$ to $\mathbf{y}_n$ and one Decoder-Layer, mapping $\mathbf{y}_n$ to $\tilde{\mathbf{x}}_n$.

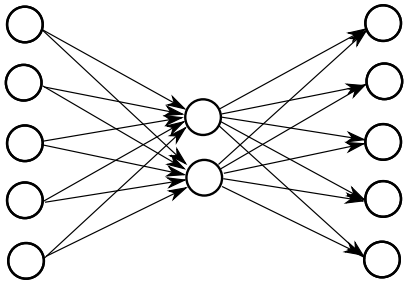


Figure 11.3: Example of an autoencoder. From left to right: Input layer $\mathbf{x}_n$ (circles), matrix W (arrows), hidden layer $\mathbf{y}_n$, matrix $W^\top$, output layer $\tilde{\mathbf{x}}_n$.

The optimal weights of the NN minimize the cost function, which consists of the error between the input and the output of the NN

$$\mathbf{w}^\star = \underset{\mathbf{w}}{\operatorname{argmin}} \frac{1}{N} \sum_{n=1}^{N} \|\mathbf{x}_n - W^T \underbrace{W\mathbf{x}_n}_{\mathbf{y}_n}\|_2^2 \tag{11.32}$$

It can be shown that the loss is convex on the parameters, therefore has a global unique minimum. If $M = 1$ the single-layer NN recovers the first principal component, i.e. $\mathbf{w}^\star = \mathbf{v}_1$. For a larger latent space, the matrix W found by the NN and PCA won't necessarily be the same, but the subspace spanned by the respective vectors in W will. In particular, there is no guarantee that the columns of W are orthogonal or have a unit norm, as there is no such constraint in the formulation. We note that the matrix in the encoder and the decoder part can vary, however, the subspaces found are still the same as the ones for PCA.

11.2.1 Nonlinear Problems

One of the main advantages of Autoencoders over PCA is that autoencoder can be easily augmented to capture non-linear problems, by adding non-linear activation functions φ (sigmoid, tanh, ReLU, ELU etc.), i.e.

$$\mathbf{y}_n = \varphi(W\mathbf{x}_n), \qquad \tilde{\mathbf{x}}_n = W^T\mathbf{y}_n. \tag{11.33}$$

We can also add more intermediate layers to increase the expressibility of the network

$$\mathbf{y}_n = \varphi_L(W_L\varphi_{L-1}(\cdots W_2\varphi_1(W_1\mathbf{x}_n)), \qquad \tilde{\mathbf{x}}_n = W_1^\top\varphi_2(\cdots W_{L-1}^\top\varphi_L(W_L^\top\mathbf{y}_n)). \tag{11.34}$$

This allows us to use the important property of neural networks, namely that they are universal function approximators. Deeper networks increase expressiveness but are easier to overfit and memorize the training dataset.

Bibliography

[1] R. J. Barnes, "Matrix differentiation." [Online]. Available: https://atmos.washington.edu/~dennis/MatrixCalculus.pdf

[2] R. G. Ayoub, "Paolo ruffini's contributions to the quintic," *Archive for History of Exact Sciences*, vol. 23, no. 3, pp. 253–277, 1980. [Online]. Available: http://www.jstor.org/stable/41133596

[3] P. Díez, "A note on the convergence of the secant method for simple and multiple roots," *Applied Mathematics Letters*, vol. 16, no. 8, pp. 1211 – 1215, 2003. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893965903901194

[4] A. Ben-Israel, "A newton-raphson method for the solution of systems of equations," *Journal of Mathematical Analysis and Applications*, vol. 15, no. 2, pp. 243 – 252, 1966. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0022247X66901156

[5] M. Abramowitz, I. A. Stegun, and R. H. Romer, "Handbook of mathematical functions with formulas, graphs, and mathematical tables," *American Journal of Physics*, vol. 56, no. 10, p. 958, 1988.