1 The Linear Least Squares Problem

Introduction: Determine unknown parameters $c_1,\ldots,c_n$ using $m>n$ measurements with errors

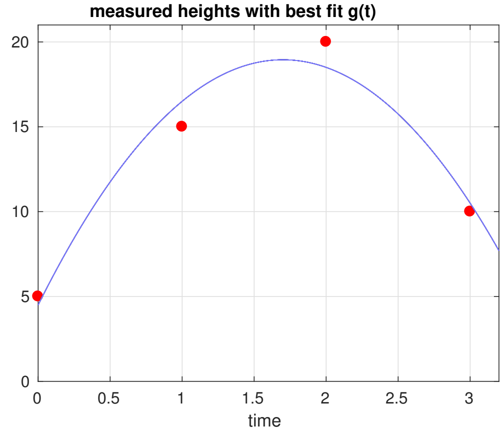
Example: We want to measure the acceleration g caused by gravity. If we throw an object into the air the height y is a quadratic function of time

$$y=g(t)=c_1+c_2t+c_3t^2$$

with three unknown parameters c_1,c_2,c_3 which we want to determine (then we obtain the acceleration from gravity as $g=-2c_3$). In order to determine 3 unknown parameters we need at least 3 measurements. But we have measurement errors, so we want to perform a much larger number m of measurements.

We now perform our experiment: we throw an object into the air and measure its height at m different times $t_1,\ldots,t_m$: We perform $m=4$ measurements and find the following data values (time t in seconds, height y in meters):

t_j	0	1	2	3
y_j	5	15	20	10


measured heights with best fit g(t)

Note that there is no quadratic function $g(t)$ which passes through all four points. We therefore want to find the function $g(t)=c_1+c_2t+c_3t^2$ which gives "the best fit" for the given data points.

In general the output value y depends on the input value x as follows:

$$y=g(t) \qquad \text{with } g(t)=c_1g_1(t)+\cdots+c_ng_n(t)$$

Here the functions $g_1(t),\ldots,g_n(t)$ are known, and we want to determine the unknown parameters $c_1,\ldots,c_n$.

We perform $m\geq n$ measurements and obtain data points $(t_1,y_1),\ldots(t_m,y_m)$ where

$$y_j=g(t_j)+e_j, \qquad j=1,\ldots,m$$

with **measurement errors** e_j. We assume that $e_1,\ldots,e_m$ are **small random errors** (we will be more precise below).

Example: If we want to fit measured points with a straight line, we have $g(x)=c_1\cdot 1+c_2\cdot t$ with $g_1(x)=1$ and $g_2(x)=t$. In this case we need $m\geq 2$ points in order to be able to estimate c_1,c_2. Because of the random errors we should use m as large as possible. Note that for $m>2$ points we will not be able (in general) to find a straight line which passes through all the data points. We would like to find $c_1,\ldots,c_n$ which give the "best fit".

For a certain choice $c_1,\ldots,c_n$ of the parameters we can measure the fit to the data values by the **residual vector** $r=(r_1,\ldots,r_m)^\top$ where

$$r_j:=g(t_j)-y_j=c_1g_1(t_j)+\cdots+c_ng_n(t_j)-y_j, \qquad j=1,\ldots,m.$$

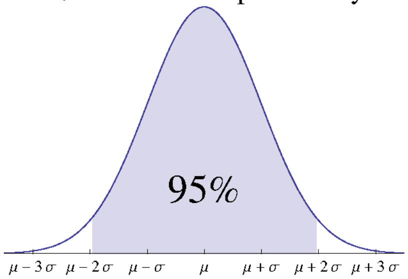
If the function $g(t)$ were the true function, the observed values $y_1,\ldots,y_m$ would have errors $r_1,\ldots,r_m$. Since large values of the errors are unlikely we want to pick $c_1,\ldots,c_n$ such that the residual vector r has a "small size".

Assumption about measurement errors

In order to understand which error vectors $(e_1,\dots,e_m)$ are "likely" or "unlikely" we need to be more precise about the distribution of the errors. We first assume the following (in section 2 we will show that the least squares method also works under a weaker assumption)

Assumption for errors $e_1,\dots,e_m$:

- the errors $e_1,\dots,e_m$ are independent
- the error e_j has **normal distribution** with a standard deviation σ (which is the *same* for all $j=1,\dots,m$): An error of size z occurs with a probability density of $c\,e^{-z^2/(2\sigma^2)}$ which is the well-known bell-shaped curve:

  Most errors observed in practice have (approximatively) a normal distribution, since they are a sum of many independent sources ("central limit theorem"). The standard deviation σ describes how much the error is "spread out". We have that $|z|\le 1.96\sigma$ with probability 95%. Note that large errors ("outliers") are extremely unlikely.

For a certain choice $c_1,\dots,c_n$ we obtain a residual vector $(r_1,\dots,r_N)^\top$. If $(c_1,\dots,c_n)$ were the true values, the observed values $y_1,\dots,y_m$ have a probability density which is given by the product of the individual densities:

$$Ce^{-(r_1^2+\cdots+r_m^2)/(2\sigma^2)}$$

For the "maximal likelihood" (most plausible choice of $c_1,\dots,c_n$) we should therefore minimize $r_1^2+\cdots r_m^2=\|r\|_2^2$ ("least squares method").

The least squares method may give bad results for $c_1,\dots,c_n$ if our assumption is not satisfied. Two typical situations are:

- The errors e_i are normally distributed with known standard deviations σ_i which have different sizes (e.g., the measurement error is larger in certain intervals for x). In this case we need to minimize

  $$\frac{r_1^2}{\sigma_1^2}+\cdots+\frac{r_m^2}{\sigma_m^2}$$

  ("**weighted least squares method**"). We define $\tilde y_j := y/\sigma_j$ and $\tilde a_{jk} := a_{jk}/\sigma_j$, then $\frac{r_1^2}{\sigma_1^2}+\cdots+\frac{r_m^2}{\sigma_m^2}=\left\|\tilde A c-\tilde y\right\|^2$ and we can use the normal algorithm ("ordinary least squares method") with $\tilde A$ and $\tilde y$.
- There are a few very large errors, so called "**outliers**". This can be due to the fact that in addition to the standard error sources (small noise, measurement errors) there can be some rare large errors, e.g., if we accidentally knock against our delicate apparatus while performing our experiment. Since for the normal distribution large errors are extremely rare, outliers have a strong effect on the obtained parameters $c_1,\dots,c_n$ and can spoil the result.

It turns out that the least squares method works well **even when the errors are not normally distributed**. We only need the following **properties for the errors:**

- the errors $e_1,\dots,e_m$ are independent
- the mean (expectation) is zero: $E[e_j]=0$
- the variance is the same for $j=1,\dots,m$: $E[e_j^2]=\sigma^2$

“Least squares method”

We want to find coefficients $c_1,\ldots,c_n$ such that the 2-norm$\|r\|_2$ is minimal, i.e.,

$$F(c_1,\ldots,c_n) := r_1^2+\cdots+r_m^2 = \text{minimal}. \tag{1}$$

Define the matrix $A \in \mathbb{R}^{m\times n}$ by

$$A = \begin{bmatrix} g_1(t_1) & \cdots & g_n(t_1) \\ \vdots & & \vdots \\ g_1(t_m) & \cdots & g_n(t_m) \end{bmatrix}$$

then the residual vector is given by $r = Ac-y$ and $F(c_1,\ldots,c_n) = \|Ac-y\|_2^2$.

Therefore we can pose the least squares problem in the following form: Given a matrix $A \in \mathbb{R}^{N\times n}$ and a right-hand side vector $y \in \mathbb{R}^m$, find a vector $c \in \mathbb{R}^n$ such that

$$\|Ac-y\|_2 = \text{minimal}. \tag{2}$$

We will write $\|\cdot\|$ for $\|\cdot\|_2$ from now on.

Normal Equations

Note that the function $F(c_1,\ldots,c_n)$ is a quadratic function of the coefficients $c_1,\ldots,c_n$. Since this is a smooth function, at a minimum we must have that the partial derivatives satisfy

$$\frac{\partial F}{\partial c_1} = 0, \ldots, \frac{\partial F}{\partial c_n} = 0. \tag{3}$$

Since $F(c_1,\ldots,c_n) = r_1^2+\cdots+r_m^2$ and $r_j = a_{j1}c_1+\cdots+a_{jn}c_n - y_j$ we obtain using the chain rule

$$\frac{\partial F}{\partial c_1} = 2r_1\frac{\partial r_1}{\partial c_1}+\cdots+2r_m\frac{\partial r_m}{\partial c_1} = 2r_1a_{11}+\cdots+2r_ma_{m1} = 2[a_{11}\cdots a_{m1}]\begin{bmatrix} r_1 \\ \vdots \\ r_m \end{bmatrix} \overset{!}{=} 0$$

for the first equation in (3). All n equations in (3) together can therefore be written as

$$\begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & & \vdots \\ a_{n1} & \cdots & a_{mn} \end{bmatrix}\begin{bmatrix} r_1 \\ \vdots \\ r_m \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \qquad \text{i.e.,} A^\top r = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}.$$

These are the so-called **normal equations**. Since $r = Ac-y$ we obtain $A^\top(Ac-y) = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ or

$$A^\top Ac = A^\top y. \tag{4}$$

This leads to the following algorithm:

1. Let $M := A^\top A$ and $b := A^\top y$
2. Solve the $n\times n$ linear system $Mc = b$.

In Matlab we can solve the normal equations and plot the resulting curve as follows:

```
t = [0;1;2;3];  y = [5;15;20;10];  % given data values
A = [t.^0, t, t.^2];               % columns of A contain values of functions 1, t, t^2 for given t-values
M = A'*A;  b = A'*y;               % matrix and rhs vector for normal equations
c = M\b                            % solve normal equations, this gives coefficients c for least squares fit

tp = (0:.01:3.1)';                 % t-values for plotting as column vector
Ap = [tp.^0, tp, tp.^2];           % values of functions 1, t, t^2 for given tp-values
plot(t,y,'o',tp,Ap*c]              % plot data points and least squares fit
```

Note that $\|Ac-y\| = \min$ means that we want to approximate the vector y by a linear combination of the columns of the matrix A. If a column of A is a linear combination of some other columns, this column is "superfluous", and leads to multiple solutions c which all give the same approximation Ac.

Therefore it makes sense to assume that the **columns of the matrix** A **are linearly independent**, i.e.,

$$Ac = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \implies c = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}. \tag{5}$$

This means that the rank of the matrix A is n (the rank is the number of linearly indpendent columns).

Theorem 1. *Assume that the columns of A are linearly independent. Then*

1. The normal equations have a unique solution $c \in \mathbb{R}^n$.
2. This vector c gives the unique minimum of the least squares problem: For $\tilde{c} \in \mathbb{R}^n$ with $\tilde{c} \neq c$ we have

$$\|A\tilde{c} - y\| > \|Ac - y\|$$

Proof. For (1.) we have to show that the matrix $M = A^\top A$ is nonsingular, i.e., $Mc = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} \implies c = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$. Therefore we assume $Mc = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$. By multiplying with $c^\top$ from the left we obtain

$$\underbrace{c^\top A^\top}_{(Ac)^\top} Ac = 0, \qquad \text{i.e., } \|Ac\| = 0$$

which means $Ac = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$. Now our assumption (5) gives $c = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$.

For (2.) we let $\tilde{c} = c + d$ with $d \neq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ and have $\tilde{r} := A\tilde{c} - y = A(c+d) - y = (Ac - y) + Ad = r + Ad$, hence

$$\begin{aligned} \|\tilde{r}\|^2 &= (r+Ad)^\top(r+Ad) = r^\top r + 2(Ad)^\top r + (Ad)^\top(Ad) \\ &= \|r\|^2 + 2d^\top \underbrace{A^\top r}_{\begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}} + \underbrace{\|Ad\|^2}_{>0} > \|r\|^2 \end{aligned}$$

where $A^\top r = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ by the normal equations, and $\|Ad\| > 0$ because of $d \neq \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$ and (5). □

Solving the least squares problem in machine arithmetic

We are given a matrix $A \in \mathbb{R}^{m \times n}$ with n linearly independent columns, and a right hand side vector $y \in \mathbb{R}^m$. We want to compute the vector $c \in \mathbb{R}^n$ which minimizes $\|Ac - y\|_2$ (the 2-norm of the residual).

Loosely speaking this means that we try to “solve the linear system $Ac = y$ as best as possible”: We cannot find c such that $Ac = y$, so the next best thing is to find c with $\|Ac - y\|$ as small as possible.

In Matlab we can solve this problem using the **normal equations** $A^\top Ac = A^\top y$ and use **`M = A'*A; b = A'*y; c = M\b`**

There is a **Matlab shortcut** for this: We can just type **`c=A\y`** (as if we were solving the linear system $Ac = y$).

On a computer with machine arithmetic it turns out that **solving the normal equations can be a numerically unstable algorithm**. We can illustrate this by looking at the special case $m = n$ with a square nonsingular matrix A. In this case $\|r\| = \|Ac - y\|$ is minimized by solving the linear system $Ac = y$, and we have $\|r\| = 0$. Assume that A has a large condition number of about 10^3, then typically $A^\top A$ has a condition number of about 10^6. Therefore by solving the normal equations with matrix $M = A^\top A$ we will lose about 6 digits of accuracy. On the other hand we can just solve $Ac = y$ and only lose about 3 digits of accuracy. Hence in this special case the algorithm with the normal equations is numerically unstable. It turns out that a similar loss of accuracy can also happen for $m > n$ if we use the normal equations.

There is an alternative algorithm for solving the linear least squares problem which is called “**QR decomposition**”.

When you use the shortcut command **`c=A\y`** Matlab actually uses the QR decomposition to compute the vector c. This will give less roundoff error, compared to using `M = A'*A; b = A'*y; c = M\b`.