Introduction to Statistical Methods
MSci Skills PH3010 – 2017/2018

Glen D. Cowan
Physics Department

ROYAL HOLLOWAY UNIVERSITY OF LONDON

Last revised: November 3, 2017

1. Script on least lquares fitting
2. Slides on least squares (weeks 1 and 2) and machine learning (week 3)
3. Sample code

Parameter Fitting and Model Testing with the Method of Least Squares

1 Introduction

This module will provide an introduction to parameter fitting and model testing with the method of least squares, which are simple but powerful tools in the analysis of experimental data. The basic problem is illustrated in Fig. 1. Suppose we have measurements of a quantity y, each of which is carried out corresponding to a fixed value of some other variable x, called the control variable. Roughly speaking the goal is to find a smooth curve that passes close to the data points, called "fitting" a curve to the data.

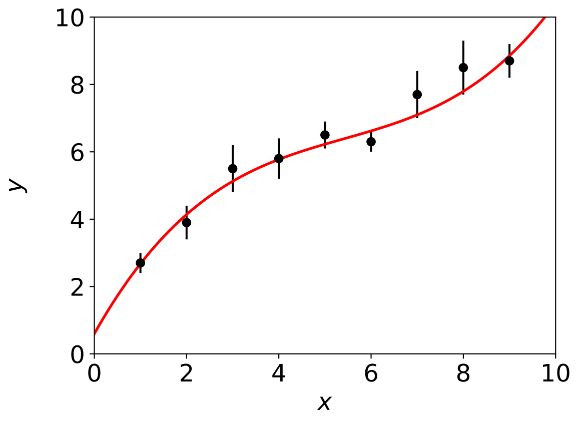


Figure 1: Typical data values for the least-squares problem: measurements y with error bars σ are carried out at known values of a control variable x. The curve has been fitted to the data.

In Sec. 2 we will formulate this problem more precisely and state how the method of least squares is used to find the fitted curve. Then in Secs. 3, 4, 5 and 6 we will examine in greater detail several aspects of this problem: determining the fitted curve itself, quantifying its statistical uncertainty, testing whether the hypothesized functional form of the curve provides an adequate description of the data, and extending the procedure to the case of correlated data. Finally we present a set of exercises including analysis of historical data from Galileo and Ptolemy.

2 The method of least squares

Referring back to Fig. 1 we will suppose here that the x values are known very precisely, but the measured y values are subject to random fluctuations. For any given measurement we do not know exactly how far or in what direction the observed value has departed from its "true" value, but we will assume we have some measure σ of the typical size of the fluctuations. Suppose we have N measurements each labeled with an index i, and thus the data consist of the set of values (x_i, y_i, σ_i) with $i = 1, \dots, N$. These are displayed in Fig. 1 with the σ_i values as error bars.

Looking at the data in Fig. 1, one can believe that if it were possible to carry out the measurements without the random errors, i.e., in the case where σ is zero or negligibly small, then all of the (x, y) points would lie on a smooth curve. That is, there could be some function $f(x)$ that gives the "true" values, and the observed values differ from these as a result of the fluctuations.

More precisely we will treat the y_i as *independent random variables.* That is, if one were to repeat the measurement of y_i many times under the same circumstances (i.e., at the same x_i), the observed y_i values would come out different each time, and would follow a distribution with a certain width. To characterize this width we give the standard deviation σ_i. Note that in the real problem we only have one value y_i for a given x_i; here the repeated measurements used to interpret the meaning of the standard deviation are purely hypothetical. The further statement that the set of y_i values are independent means that the measured value of any one of them has no influence on the fluctuations of any of the others.

If we only have the set of (x, y, σ) measurements, how can we figure out the curve $f(x)$? Given the available data, there is no way to give a definitive answer to this question. But we can *estimate* the curve using statistical methods. The approach we will take here requires that we hypothesize a formula for $f(x)$ that contains some adjustable parameters, i.e., constants whose values are not yet determined. We will usually use Greek letters for the parameters and may write them as a vector if we have more than one, e.g., $f(x; \boldsymbol{\theta})$ with $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$. For example, we could suppose that $f(x)$ is a straight line, i.e.,

$$f(x; \boldsymbol{\theta}) = \theta_0 + \theta_1 x \ , \tag{1}$$

where here $\boldsymbol{\theta} = (\theta_0, \theta_1)$. The problem of estimating the curve is now reduced to estimating the unknown parameters. In this example, each possible choice for the values will yield a possible line, and the question is how to decide which point in the parameter space is best.

Suppose we pick an arbitrary point (θ_0, θ_1), which corresponds to a given curve, as shown in Fig. 2. Our initial choice is evidently not very good, as the amount by which the data points miss the curve, $y_i - f(x_i; \boldsymbol{\theta})$, is quite large for most of the points.

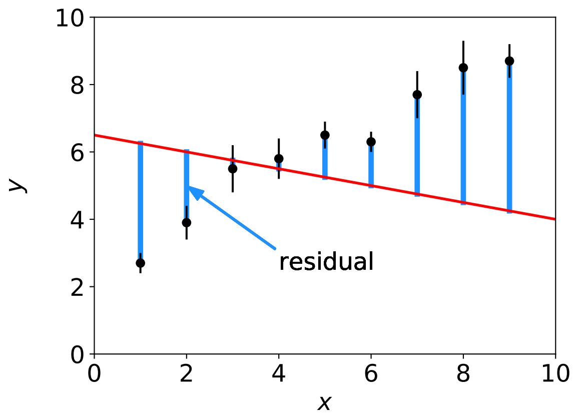


Figure 2: The line corresponding to an arbitrarily chosen point in the parameter space (θ_0, θ_1) together with the data points. The thick vertical bars show the residuals $y_i - f(x_i; \boldsymbol{\theta})$.

Notice that for some of the points in Fig. 2, the difference or "residual" $y_i - f(x_i; \boldsymbol{\theta})$ is large but also the error bar σ_i is large. For these points one expects a bigger fluctuation in the measured value. Some of the measurements have small error bars, and so these should lie closer to the hypothesized curve. So to characterize how far a measured y_i is from the predicted value $f(x_i; \boldsymbol{\theta})$, we can measure the residual $y_i - f(x_i; \boldsymbol{\theta})$ in units of σ_i, i.e., we use

$$\frac{y_i - f(x_i; \boldsymbol{\theta})}{\sigma_i} \,, \tag{2}$$

sometimes called the *normalized* residual. The "best" curve should give small values for the normalized residuals, but these can be positive or negative for different data points, so we cannot simply minimize their sum. In the method of *Least Squares* (LS), we define the best parameter estimates to be those that minimize the sum of squares of the normalized residuals, a quantity called the chi-squared,

$$\chi^2(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \boldsymbol{\theta}))^2}{\sigma_i^2} \,. \tag{3}$$

The values of the parameters that give the minimum chi-squared, called the *estimators* of the parameters, are written with hats, e.g., $\hat{\theta}_0$ and $\hat{\theta}_1$. Figure 3 shows the example of the fitted line, i.e., the curve $f(x; \boldsymbol{\theta})$ evaluated with the LS estimators for $\boldsymbol{\theta}$.

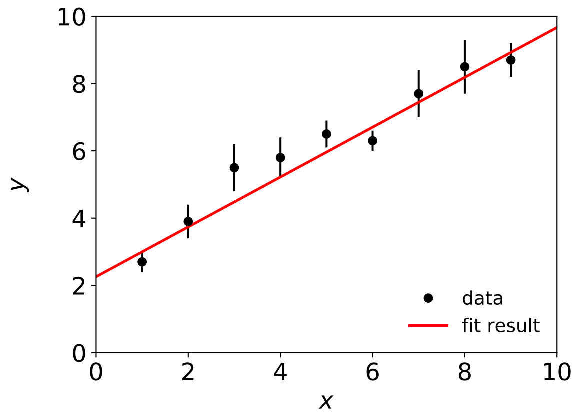


Figure 3: Least-squares fit of a straight line (see text).

One can derive the method of least squares from a more general starting point called the *method of maximum likelihood* in the special case where the data values y_i are statistically independent and each follow a Gaussian distribution (see, e.g., Ref. [2]). But it is equally valid to simply use the minimum of the $\chi^2(\boldsymbol{\theta})$ to define the estimators and it is important to emphasise that it is not possible in general to construct estimators that are "best" in every relevant sense. In practice it turns out that LS estimators have desirable properties in several respects, as we will see in examining the following questions:

1. How do we find the estimators, i.e., how do we minimize $\chi^2(\boldsymbol{\theta})$?
2. How do we quantify the statistical uncertainty in the estimated parameters that stems from the random fluctuations in the measurements, and how is this information used in an analysis problem, e.g., using error propagation?
3. How do we assess whether the hypothesized functional form $f(x; \boldsymbol{\theta})$ adequately describes the data?

3 Finding the minimum of $\chi^2(\boldsymbol{\theta})$

Depending on the problem there can be a variety of different methods employed to find the estimators, i.e., to minimize the quantity $\chi^2(\boldsymbol{\theta})$. We will illustrate using a simple example first the case where we can write down the estimators in closed form in Sec. 3.1, and then by carrying out the minimization numerically in Sec. 6.

3.1 Finding LS estimators in closed form

The curve $f(x;\boldsymbol{\theta})$ that we are fitting to the data does not of course have to be a straight line; it can be any function. It turns out that the method of least squares is particularly easy for problems where $f(x;\boldsymbol{\theta})$ is a linear function *of the parameters* (it does not matter whether it is a linear function of the control variable x). As an example one could have a polynomial of order M, i.e.,

$$f(x;\boldsymbol{\theta}) = \sum_{n=0}^{M} \theta_n x^n \; . \tag{4}$$

For this type of problem it is possible to find the values of the parameters that minimize the chi-squared in closed form, and we will show this here for the special case of a first-order polynomial, i.e., a straight line. Here the the function $\chi^2(\boldsymbol{\theta})$ takes on the form

$$\chi^2(\theta_0, \theta_1) = \sum_{i=1}^{N} \frac{(y_i - \theta_0 - \theta_1 x_i)^2}{\sigma_i^2} \; . \tag{5}$$

To find its minimum we set the derivatives of χ^2 with respect to both parameters equal to zero,

$$\frac{\partial \chi^2}{\partial \theta_0} = \sum_{i=1}^{N} \frac{-2(y_i - \theta_0 - \theta_1 x_i)}{\sigma_i^2} = 0 \; , \tag{6}$$

$$\frac{\partial \chi^2}{\partial \theta_1} = \sum_{i=1}^{N} \frac{-2x_i(y_i - \theta_0 - \theta_1 x_i)}{\sigma_i^2} = 0 \; . \tag{7}$$

These two equations can be rewritten using matrix notation as

$$\begin{pmatrix} \sum_{i=1}^N \frac{1}{\sigma_i^2} & \sum_{i=1}^N \frac{x_i}{\sigma_i^2} \\ \sum_{i=1}^N \frac{x_i}{\sigma_i^2} & \sum_{i=1}^N \frac{x_i^2}{\sigma_i^2} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^N \frac{y_i}{\sigma_i^2} \\ \sum_{i=1}^N \frac{x_i y_i}{\sigma_i^2} \end{pmatrix} \; , \tag{8}$$

which has the general form

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix} \; . \tag{9}$$

By comparing Eqs. (8) and (9) we can read off the values of a, b, c, d, e and f. Then using the fact that a 2×2 matrix A and its inverse A^{-1} can be written

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \; , \qquad A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix} \; , \tag{10}$$

and applying A^{-1} to both sides of Eq. (9), we can find solutions

$$\hat{\theta}_0 = \frac{de - bf}{ad - bc} \, , \tag{11}$$

$$\hat{\theta}_1 = \frac{af - ec}{ad - bc} \, . \tag{12}$$

Here we write the solutions with hats because these are the estimators for the parameters θ_0 and θ_1.

We can extract a few lessons from this example. First, we see that the solutions can be obtained by solving a system of linear equations using standard matrix techniques. Furthermore, the data values y_i enter into the estimators through the quantities e and f, and therefore from Eqs. (11) and (12) we see that the estimators $\hat{\theta}_0$ and $\hat{\theta}_1$ are linear functions of the y_i. This remains true for an arbitrary number of parameters, as long as the function that we are fitting, $f(x;\boldsymbol{\theta})$, is linear in the parameters (see, e.g., Ref. [2]). Finally we see that although we can write down expressions for the solutions in closed form, they are somewhat involved even for the simple case of two parameters. If the number of parameters is higher then this approach may become impractical, and in any case if $f(x;\boldsymbol{\theta})$ is nonlinear in any of the parameters then we must use other methods to find the minimum of $\chi^2(\boldsymbol{\theta})$.

3.2 Numerical minimization of $\chi^2(\boldsymbol{\theta})$

In most problems of practical interest it is too difficult or impossible to find formulas for the estimators in closed form as done in the previous section. Rather, we must find the minimum of $\chi^2(\boldsymbol{\theta})$ (called the *objective function*) numerically. The study of algorithms used to carry out this task constitutes an entire branch of applied mathematics called *mathematical optimization.*

Here we will not go into the details of these algorithms other than to mention some of their general principles. The basic idea is to evaluate the objective function $\chi^2(\boldsymbol{\theta})$ at different points in the parameter space and in this way to try to figure out where it has its minimum. Some algorithms make use of the derivatives of the objective function with respect to the parameters if these are known. One must start at a certain point in the parameter space, say, $\boldsymbol{\theta}_{\rm start}$. Sometimes default values can be used for this; in other problems one may need to have a rough preliminary estimate.

For example, starting with some initial guess for the parameters, hold all but one of them constant, say, θ_0, and determine the value of the others which minimizes the χ^2. This can be done by finding the solution to $\partial\chi^2/\partial\theta_0 = 0$ using, e.g., the Newton–Raphson method. Then hold all parameters constant at the newly found point except the next one, say, θ_1, and determine its value so as to minimize the χ^2. After cycling through all of the parameters one will have moved to a point closer to the minimum. Repeat the procedure as many times as necessary until the change in the parameters from one iteration to the next drops below some threshold, or alternatively, until the change in the χ^2 between iterations falls below a specified value.

In fact the most successful algorithms do not minimize by varying separately individual parameters but rather by moving along certain well-chosen directions in the parameter space, e.g., in the direction where the objective function is decreasing most rapidly (called the method of steepest descent) or using so-called conjugate gradient directions. More details on these types of algorithms can be found in Chapter 10 of Ref. [1].

Here we will show a simple example of numerical minimization using routines in Python. Let us suppose we have the data points shown above in Fig. 1 and we want to fit a straight line. A simple but relatively flexible tool in Python is the routine `curve_fit`, which is part of the package `scipy.optimize`. To use this we first need to import the relevant packages:

```
import numpy as np
from scipy.optimize import curve_fit
```

We need to define the function that we will be fitting to the data. For this example suppose it is a straight line:

```
def func(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x
```

Then we need to have the data values (x_i, y_i, σ_i) in the form of NumPy arrays. In practice we would probably read the data values in from a file, but for this example let us just suppose we have the arrays defined by

```
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

To carry out the least-squares fit we usually want to supply initial values for the parameters. If we have reasonable guesses then these can be used; if they are not supplied then the program will take the initial values equal to 1.0. Looking at the graph we see that the slope and y intercept are both of order unity, so we define

```
p0 = np.array([1.0, 1.0])
```

In practice the result should be insensitive to the initial values, but for some special problems one may need to choose this point with some care. To actually fit the curve we use the SciPy's routine `curve_fit`:

```
thetaHat, cov = curve_fit(func, x, y, p0, sig, absolute_sigma=True)
```

Notice that it is necessary here to include the argument `absolute_sigma=True` in order for the statistical errors of the estimated parameters to have the interpretation that we will require. This results in $\hat{\theta}_0 = 2.26 \pm 0.29$ and $\hat{\theta}_1 = 0.741 \pm 0.057$, as represented by the fitted curve shown earlier in Fig. 3. Here the indicated statistical errors correspond to the standard deviations of the estimators, as described in the next section.

4 Statistical errors of the fitted parameters

In this section we will examine the statistical uncertainties in the estimated parameters, which stem from the random fluctuations in the original measurements. First in Sec. 4.1 we will discuss in some detail how to interpret the statistical errors and how to find them using the Monte Carlo method. In Sec. 4.2 we will show a much simpler way to estimate the errors using derivatives of the function $\chi^2(\boldsymbol{\theta})$, and in Sec. 4.3 we show how to use the statistical uncertainties with error propagation.

4.1 Meaning of statistical errors and estimation by Monte Carlo method

First let us consider carefully the statement that the estimators $\hat{\boldsymbol{\theta}}$ (e.g., $\hat{\theta}_0$ and $\hat{\theta}_1$ in the example above) have statistical errors. What this means is that if we were to repeat the entire experiment with a new, statistically independent set of measurements y_i with $i = 1, \ldots, N$, then the fluctuations in the data would in general be different from before and therefore the numerical values of the estimates $\hat{\boldsymbol{\theta}}$ would also differ. Equivalently we can say that the estimators $\hat{\boldsymbol{\theta}}$ are functions of random variables $y_1, \ldots, y_N$, and they are therefore random variables themselves.

Although we may only carry out the real experiment once, we can simulate it many times with the Monte Carlo method. Figure 4 shows the outcomes of many independent simulations of the same set of measurements of the straight-line fit described above, each of which is characterized by two fitted parameters, $\hat{\theta}_0$ and $\hat{\theta}_1$.

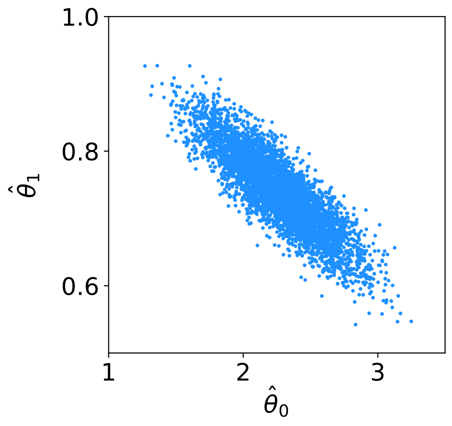


Figure 4: Estimates $\hat{\theta}_0$ and $\hat{\theta}_1$ from repeated simulations of the straight-line fit, each based on a statistically independent set of measurements.

Figures 5 show the projections of the points from the scatter plot onto the $\hat{\theta}_0$ and $\hat{\theta}_1$ axes, which give histograms of the corresponding quantities. The widths, or more precisely, the standard deviations of these distributions are used to quantify the statistical errors in the parameters, which we can write as $\sigma_{\hat{\theta}_0}$ and $\sigma_{\hat{\theta}_1}$ (note the hats).

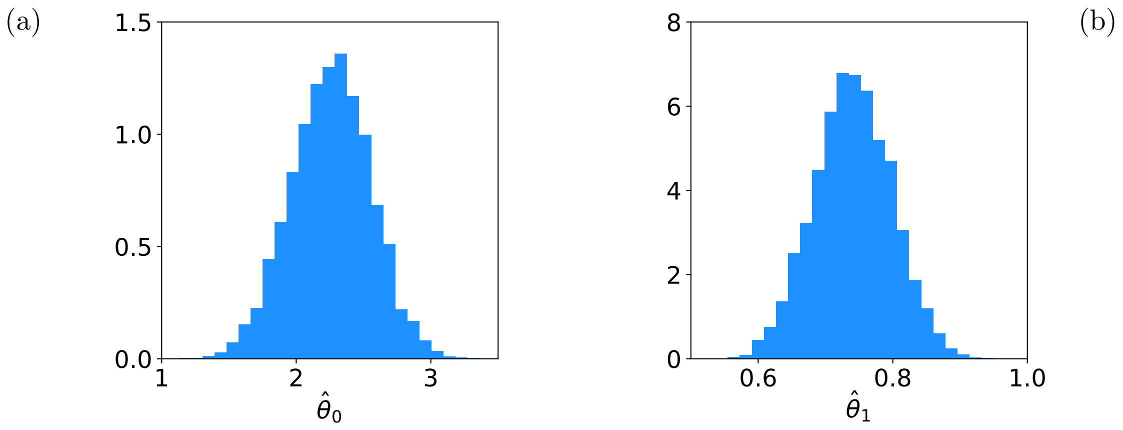


Figure 5: Projections of the scatter plot from Fig. 4 onto (a) the $\hat{\theta}_0$ and (b) the $\hat{\theta}_1$ axes.

Returning to the scatter plot in Fig. 4, we see another interesting feature, namely, that the cloud of points is tilted. This shows that if the value of, say, $\hat{\theta}_0$ comes out higher than average, then there is a higher probability for the value of $\hat{\theta}_1$ to be lower than its average.

That is, the quantities $\hat{\theta}_0$ and $\hat{\theta}_1$ are *correlated*, in this case the correlation being negative.[1]

Before we quantify the statistical errors in the estimators we need to review briefly how to characterise the typical amount of fluctuation in random variables. Recall that to do this for a single random variable, say, u, one defines the *variance* $V[u]$ as the mean of its square minus the square of its mean, i.e.,

$$V[u] = \langle u^2 \rangle - \langle u \rangle^2 \; . \tag{13}$$

Often we use the notation $V[u] = \sigma_u^2$, and the square root of the variance, σ_u, is the standard deviation. In a similar way, to characterise the amount of variation in a pair of random variables, say, u and v, we define the *covariance* as the mean of their product minus the product of the means,

$$\mathrm{cov}[u, v] = \langle uv \rangle - \langle u \rangle \langle v \rangle \; . \tag{14}$$

If u and v have some units such as kg or cm, then the covariance has the units given by their product. It is often convenient to use the dimensionless *correlation coefficient* ρ, defined as the covariance of the two variables divided by the product of their standard deviations,

$$\rho = \frac{\mathrm{cov}[u, v]}{\sigma_u \sigma_v} \; . \tag{15}$$

One can show that the correlation coefficient for any pair of variables lies in the range $-1 \leq \rho \leq 1$. The extreme values of $\rho = \pm 1$ corresponding to a 100% positive $(+1)$ or negative (-1) correlation (i.e., perfect *anticorrelation*). The corresponding scatter plot is then infinitesimally thin and tilted up $(\rho = +1)$ or down $(\rho = -1)$. If $\rho = 0$ then the variables are uncorrelated, and their scatter plot is not tilted.

If one has more than two random variables, e.g., a set of m estimators $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_m)$, then there is a covariance for each pair, and these numbers can be arranged in a *covariance matrix*,

$$U_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j] \; . \tag{16}$$

From the definition of covariance it is easy to check that the matrix is square and symmetric. The diagonal elements simply give the variances

$$U_{ii} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_i] = \sigma_{\hat{\theta}_i}^2 \; , \tag{17}$$

and so by taking their square roots one finds the standard deviations $\sigma_{\hat{\theta}_i}$. The set of correlation coefficients also form a matrix ρ_{ij}, given by

$$\rho_{ij} = \frac{U_{ij}}{\sigma_{\hat{\theta}_i} \sigma_{\hat{\theta}_j}} \; . \tag{18}$$

It is easy to verify that the diagonal elements are $\rho_{ii} = 1$ for all i, which is to say that a random variable is always 100% positively correlated with itself.

[1]Often one hears statements such as "the parameters are correlated", but this is not strictly correct. Here we are treating the parameters θ_0 and θ_1 as unknown constants, not as random variables. It is the estimators, $\hat{\theta}_0$ and $\hat{\theta}_1$, which are functions of the random data, that vary upon repetition of the experiment, and therefore we should say that the *estimators* of the parameters are correlated.

From the scatter plot of the simulated estimates $\hat{\theta}_0$ and $\hat{\theta}_1$ we see that they have a negative correlation. Using standard formulas or NumPy routines to estimate covariance from a sample of observed values (see, e.g., Refs. [2, 3]), we find

$$\begin{aligned}
\sigma_{\hat{\theta}_0} &= 0.29 \,, && (19)\\
\sigma_{\hat{\theta}_1} &= 0.057 \,, && (20)\\
\mathrm{cov}[\hat{\theta}_0, \hat{\theta}_1] &= -0.0142 \,, && (21)\\
\rho &= -0.86 \,. && (22)
\end{aligned}$$

4.2 Statistical errors from derivatives of $\chi^2(\boldsymbol{\theta})$

The Monte Carlo technique described in the previous section provides estimates of the covariance of the estimators but it requires a certain amount of work to set up the simulations and work out the required numbers. The covariance matrix can also be determined directly from the single real data set, with no need of simulated data, by using the second derivatives of $\chi^2(\boldsymbol{\theta})$ with respect to the parameters.

Derivation of the required formulas is nontrivial and goes beyond the scope of this module. We claim here without proof that the inverse of the covariance matrix for a set of estimators $U_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j]$ can be estimated by

$$U_{ij}^{-1} = \frac{1}{2} \left(\frac{\partial^2 \chi^2}{\partial \theta_i \partial \theta_j} \right)_{\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}} \,. \tag{23}$$

In general this formula is an approximation but it is exact or at least close to being so in most cases of practical interest. When using the Python routines described above, the `curve_fit` function finds the minimum of $\chi^2(\boldsymbol{\theta})$ numerically and then by taking small steps about the minimum, it finds the matrix of second derivatives to get U^{-1}. It then inverts this and returns the covariance matrix U.

For the example of the straight line shown above, `curve_fit` returns the covariance matrix

$$U = \begin{pmatrix} 0.08537 & -0.01438 \\ -0.01438 & 0.003275 \end{pmatrix} , \tag{24}$$

where the first row/column corresponds to the estimator $\hat{\theta}_0$ and the second to $\hat{\theta}_1$. By taking the square root of the diagonal elements one finds the standard deviations $\sigma_{\hat{\theta}_0} = 0.29$ and $\sigma_{\hat{\theta}_1} = 0.057$, and combining with the covariance one finds the correlation coefficient $\rho_{01} = -0.86$. All of these numbers agree to good level of precision with the values found using the Monte Carlo calculation in Sec. 4.1.

4.3 Using the covariance of estimators with error propagation

Suppose we have carried out a least-squares fit and found a set of estimators $\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \ldots, \hat{\theta}_m)$ and their covariance matrix $U_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j]$. Very often we then form some function u of the estimators as the quantity of interest, e.g., $u(\hat{\boldsymbol{\theta}}) = (\hat{\theta}_1 \hat{\theta}_2 - 2\hat{\theta}_3)^2$. Because the estimators $\hat{\theta}_i$

are themselves random variables, a function of them is also a random variable with a certain variance. To characterise the statistical uncertainty in the function, we need to “propagate” the covariance of the $\hat{\theta}_i$ through to u. An analogous procedure is carried out with more than one function, e.g., u and v, whereby one finds their variances as well as the covariance $\mathrm{cov}[u, v]$.

One way to carry out this procedure would be to simulate the entire experiment many times, just as in Sec. 4.1. Each time we determine the estimates $\hat{\boldsymbol{\theta}}$, with these evaluate the function $u(\hat{\boldsymbol{\theta}})$, and record its value, e.g., in a histogram. By repeating a sufficient number of times one obtains the distribution of u and thus its standard deviation can be found.

Although the Monte Carlo procedure is robust in the sense that it is essentially guaranteed to produce a meaningful result, it may require some effort to set up. So it is useful to have a simpler approximate method, called (linear) *error propagation*. This is based on a linear approximation to the function in the neighborhood of the estimates actually found. For a pair of functions $u(\hat{\boldsymbol{\theta}})$ and $v(\hat{\boldsymbol{\theta}})$ one finds for the covariance (see, e.g, Ref. [2]),

$$\mathrm{cov}[u, v] \approx \sum_{i,j=1}^{m} \frac{\partial u}{\partial \hat{\theta}_i} \frac{\partial v}{\partial \hat{\theta}_j} U_{ij} \;, \tag{25}$$

where $U_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j]$ is the covariance matrix of the estimators of the parameters and the derivatives are evaluated at the estimated values of the parameters. In the case where there is only one function, say, u, its variance is given by $\sigma_u^2 = \mathrm{cov}[u, u]$, i.e.,

$$\sigma_u^2 \approx \sum_{i,j=1}^{m} \frac{\partial u}{\partial \hat{\theta}_i} \frac{\partial u}{\partial \hat{\theta}_j} U_{ij} \;. \tag{26}$$

In the special case where the covariance may be diagonal, i.e., $U_{ij} = \delta_{ij}\sigma_{\hat{\theta}_i}^2$, then one can carry out one of the two sums to find

$$\sigma_u^2 \approx \sum_{i=1}^{m} \left(\frac{\partial u}{\partial \hat{\theta}_i} \right)^2 \sigma_{\hat{\theta}_i}^2 \;. \tag{27}$$

It should be emphasised, however, that estimators for parameters are correlated in general, i.e., the covariance matrix usually has nonzero off-diagonal elements, and thus one must use the full covariance in propagating the errors.

The approximation on which Eqs. (25), (26) and (27) are based is that the functions should be linear in the neighborhood of the estimated values, where the size of the relevant neighborhood is determined by the standard deviations of the corresponding estimators. Suppose, for example, we have a function of the form $u = \hat{\theta}_1/\hat{\theta}_2$, and that $\hat{\theta}_2 = 10 \pm 1$. The function is (exactly) linear in $\hat{\theta}_1$, and although it is nonlinear in $\hat{\theta}_2$, this is approximately linear if we only consider the small region around 10 ± 1. So in this case linear error propagation should be a good approximation. If, on the other hand, we have $\hat{\theta}_2 = 10 \pm 10$, then the function is very nonlinear if we consider this larger range. In this case linear error propagation can give wildly wrong results. The Monte Carlo approach, however, could be used in such a case to infer the correct distribution of the function u.

5 Goodness-of-fit from the minimum of $\chi^2(\boldsymbol{\theta})$

In the example from Sec. 2 we hypothesized that the data shown in Fig. 3 could be described by a straight line. Having carried out the fit we then found the values of the parameters for the slope and y-intercept. But how do we know that this hypothesis is valid? Perhaps we should have used a curve with more adjustable parameters, such as the third-order polynomial fitted to the same data in Fig. 1. Or perhaps the data come out as in Fig. 6 below; in this case the hypothesis of a straight line is clearly not very good. Beyond a visual comparison of the fitted curve with the data, how can we quantify the “goodness of fit”?

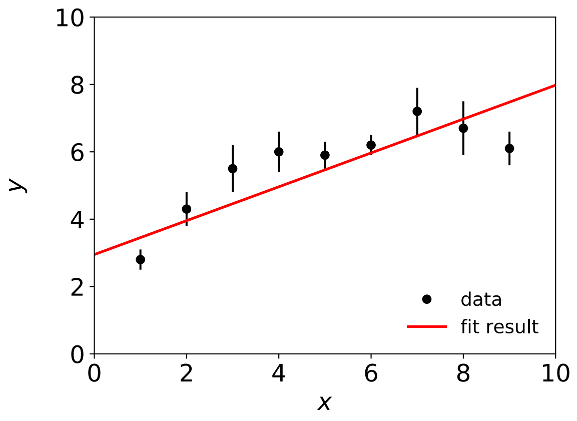


Figure 6: Fit resulting in a poor level of agreement between the hypothesized fit function and the data.

Before we go any further we should head off a common misunderstanding. One might think that if the hypothesized function is bad, then this should be reflected in large values of the statistical errors for the fitted parameters. This is not true. The level of agreement between data and hypothesis and the size of statistical errors are in general very separate questions, and in the particular example shown here they are completely unrelated. In fact, to obtain the data points shown in Fig. 6, the points from the earlier example (Fig. 3) were simply shifted up or down; their error bars were left unchanged. And as a result, the standard deviations of the estimators of the parameters shown in Fig. 6 are exactly the same as before.

The reason for this perhaps surprising behaviour lies in the meaning of the statistical errors. What they reflect is the level with which the parameter estimates would fluctuate upon repetition of the experiment with a new set of measured y_i. It could be that each repetition results in poor agreement, as found in Fig. 6, but that the fitted parameters do not vary much for each repetition, so the standard deviations of the estimators may be small. What we would like to quantify, however, is something different, namely the level to which a straight line provides an adequate description of the data.

One way to address this question is to look at the value of the χ^2 at its minimum, i.e., at $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$,

$$\chi^2_{\min} = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \hat{\boldsymbol{\theta}}))^2}{\sigma_i^2} \; . \tag{28}$$

From the form of this equation we can immediately see that if the fitted function is in good agreement with the data, then the contribution of each term in the sum is small and therefore $\chi^2_{\min}$ should be small, so this statistic can be used to reflect the goodness-of-fit. But we should not be surprised to see some nonzero value even if the fit function is correct, because of the statistical fluctuations in the data. To make a quantitative statement about the goodness-of-fit we need to know how large of a $\chi^2_{\rm min}$ to expect in the case where the hypothesized fit function is correct.

Recall that the standard deviation σ_i reflects the typical (more precisely, the root mean square) level of departure of the measurement y_i from its mean. That is to say, if the fitted value $f(x_i;\hat{\boldsymbol{\theta}})$ were equal to the true mean of y_i, then one would expect $|y_i - f(x_i;\hat{\boldsymbol{\theta}})|$ to be similar to σ_i and therefore each normalized residual (cf. Eq. (2)) should be of order unity,

$$\left| \frac{y_i - f(x_i;\hat{\boldsymbol{\theta}})}{\sigma_i} \right| \sim 1 \; . \tag{29}$$

Since there are N terms in the sum, one would naively expect $\chi^2_{\rm min}$ to be roughly equal to N. In fact this cannot be quite right, since we have adjusted some number of parameters, so the fitted values $f(x_i;\hat{\boldsymbol{\theta}})$ get pulled even closer to y_i than would be obtained from the "true" function. For example, if one had N data points and $m = N$ parameters, then one would be able to adjust their values so that the curve goes exactly through each of the points and one would find $\chi^2_{\rm min} = 0$.

The minimized value $\chi^2_{\rm min}$ is of course a function of the data and as such it is itself a random variable. If one were to repeat the experiment many times, its value would follow a certain distribution. One can show that if the hypothesised form of the function $f(x;\boldsymbol{\theta})$ is correct, and if the y_i are Gaussian distributed, and if certain additional conditions are satisfied that usually hold in practice, then the statistic $\chi^2_{\rm min}$ will follow a probability distribution function (pdf) called, not unsurprisingly, the chi-squared distribution. Calling the variable $\chi^2_{\rm min}$ here z to simplify the notation, the chi-squared pdf is given by

$$f_{\chi^2}(z;n) = \frac{1}{2^{n/2}\Gamma(n/2)} z^{n/2-1} e^{-z/2} \; . \tag{30}$$

Here the parameter n is called the "number of degrees of freedom" and takes on positive integer values, $n = 1, 2, \ldots$. If there are N measurements $y_1, \ldots, y_N$ and m fitted parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$, then the number of degrees of freedom is given by $n = N - m$.

The distribution of $\chi^2_{\rm min}$ for the example of the straight-line fit from repeated simulations of the measurement is shown as a histogram in Fig. 7. The fit had $N = 9$ measured y values and $m = 2$ fitted parameters, so the relevant number of degrees of freedom is $n = 7$. Also shown on the plot as a solid curve is the chi-squared pdf for 7 degrees of freedom.

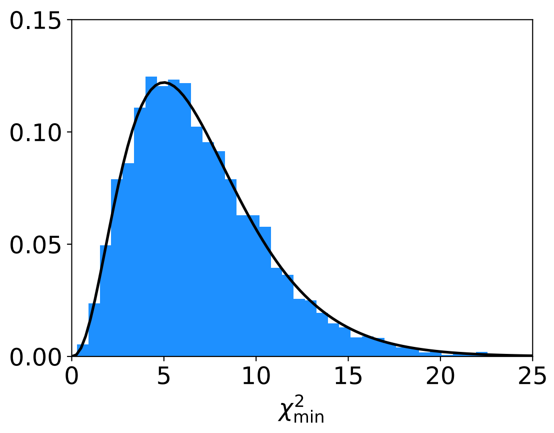


Figure 7: Distribution of the value of $\chi^2_{\rm min}$ from Monte Carlo simulations of the measurement and fit of a straight line. The solid curve shows the chi-squared pdf for $n = 7$ degrees of freedom.

The mean and standard deviation of a chi-squared distributed variable for n degrees of freedom are given by

$$\begin{aligned} \langle z \rangle &= n \, , & (31) \\ \sigma_z &= \sqrt{2n} \, . & (32) \end{aligned}$$

Since the mean is equal to the number of degrees of freedom, we can immediately get a good idea of the goodness-of-fit simply by comparing the $\chi^2_{\rm min}$ obtained to $n = N - m$, the number of measurements minus number of fitted parameters. Sometimes this is given as a ratio, $\chi^2_{\rm min}/n$, called the *chi-squared per degree of freedom.* If this ratio comes out close to unity, the fit is "good"; if it is very large, the fit is bad, and if it is much less than one then it means that the data points have come out closer to the hypothesised curve than what one would expect, given the random fluctuations that should normally be present.

For the fit shown in Fig. 6, there are 9 measurements and 2 fitted parameters, so if the straight-line hypothesis is true, one would expect a value of $\chi^2_{\rm min}$ of around 7. In the fit shown, a value $\chi^2_{\rm min} = 20.9$ was found, or a chi-squared per degree of freedom of almost 3. This can be contrasted with the fit shown in Fig. 3, which has $\chi^2_{\rm min} = 8.2$, for the same number of degrees of freedom.

The fact that the value of $\chi^2_{\rm min}$ in Fig. 6 came out quite high does not prove that the straight-line hypothesis is wrong. It can happen, as a result of of the random fluctuations in the data, that the value of $\chi^2_{\rm min}$ comes out sometimes high and sometimes low. After all it is a random variable, as illustrated by its distribution shown in Fig. 7. We can take a step further in quantifying the goodness-of-fit by giving the probability, under assumption of the hypothesis used (here a straight line), to find a value of $\chi^2_{\rm min}$ as high as the one found or higher. This can be found by integrating the chi-squared pdf from the value of $\chi^2_{\rm min}$ observed to infinity,

$$p = \int_{\chi^2_{\rm min}}^{\infty} f_{\chi^2}(z; n) \, dz \, , \tag{33}$$

where $f_{\chi^2}(z; n)$ is the chi-squared pdf for n degrees of freedom from Eq. (30). The probability p obtained from Eq. (33) is called the p-value of the hypothesis. If it comes out very small, it means that there is only a very small probability, if the hypothesis is true, to find data at least as incompatible with it or moreso. We may therefore be tempted to think of the p-value as something like the probability that the hypothesis is true, but this is not a correct statement. There is an important distinction between these quantities, which we will not consider in detail here (further discussion can be found, e.g., in Ref. [2]). For now we will simply emphasize that the p-value is a measure of compatibility between the data and a given hypothesis, and if it comes out very small then the hypothesis is disfavoured.

For the straight-line fit shown in Fig. 3, we had $\chi^2_{\rm min} = 8.2$ and 7 degrees of freedom. Using these values in Eq. (33) and carrying out the integral with the Python routine `scipy.stats.chi2.sf` gives a p-value of 0.31. Thus if the straight-line hypothesis is correct, there is a substantial probability to get an even higher value of $\chi^2_{\rm min}$, and so we would not reject the hypothesis in this case. But in the fit in Fig. 6 we found $\chi^2_{\rm min} = 20.9$, corresponding to a p-value of 0.0039. It is still possible that the straight-line hypothesis is correct, but if it is, there is only a 4 in a thousand chance to see a $\chi^2_{\rm min}$ at least as high as the one we found.

If we find a very low p-value (or equivalently, $\chi^2_{\min}$ substantially greater than the number of degrees of freedom), then we may wish to try another hypothesis. Using the same data as in Fig. 6 we can try, for example, a second-order polynomial,

$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 \ . \tag{34}$$

The fit using this function is shown in Fig. 8. It gives $\chi^2_{\min} = 3.5$ for 6 degrees of freedom, corresponding to a p-value of 0.75, and thus a much better level of agreement between the data and hypothesis.

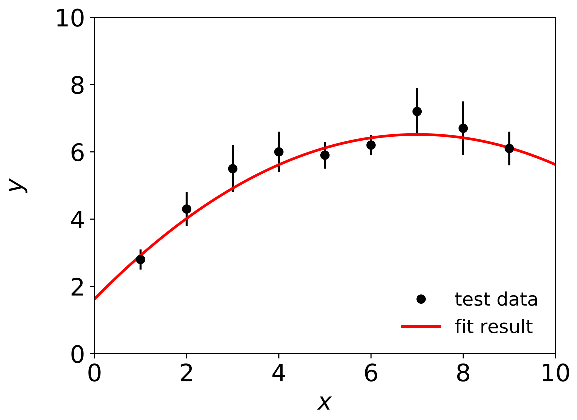


Figure 8: Fit to the same data values as in Fig. 6 but here using a fit function with three adjustable parameters.

6 Least squares with correlated measurements

In the method of least squares as described above, the quantity $\chi^2(\boldsymbol{\theta})$, whose minimum defines the estimators $\hat{\boldsymbol{\theta}}$, is given by

$$\chi^2(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{(y_i - f(x_i;\boldsymbol{\theta}))^2}{\sigma_i^2} \ . \tag{35}$$

As mentioned in Sec. 2, one can show that this recipe follows from a more general principle, namely, the method of maximum likelihood, for the special case where the measured quantities y_i are independent and Gaussian distributed. In practice the measured quantities we deal with very often satisfy these two criteria, and this is an important reason why the method of least squares is so widely used.

It is not uncommon, however, that measured values are Gaussian but they are correlated, and hence not independent. That is, each measurement does not simply have a variance $V[y_i]$, but the set of N measurements is described by an $N \times N$ covariance matrix $V_{ij} = \mathrm{cov}[y_i, y_j]$. The diagonal elements of this matrix give the variances, $V_{ii} = \mathrm{cov}[y_i, y_i] = \sigma_i^2$, and the off-diagonal elements (the covariances) are related to the correlation coefficients through

$$\rho_{ij} = \sigma_i \sigma_j V_{ij} \ . \tag{36}$$

In such a problem, one can generalise the method of least squares by replacing the sum of squares given above by

$$\chi^2(\boldsymbol{\theta}) = \sum_{i,j=1}^{N} (y_i - f(x_i;\boldsymbol{\theta}))V_{ij}^{-1}(y_j - f(x_j;\boldsymbol{\theta})) , \tag{37}$$

where V^{-1} is the inverse of the covariance matrix V. As in the uncorrelated case, this version of $\chi^2(\boldsymbol{\theta})$ can be derived from the method of maximum likelihood where the data follow a *multivariate* Gaussian distribution (see, e.g., [2]).

If we use Eq. (37) for the case where the measurements are in fact uncorrelated, then the covariance matrix is diagonal, i.e., $V_{ij} = \delta_{ij}\sigma_i^2$, and thus its inverse is simply $V_{ij}^{-1} = \delta_{ij}/\sigma_i^2$. Substituting this expression into Eq. (37) allows one to carry out one of the two sums, and the formula for $\chi^2(\boldsymbol{\theta})$ becomes again a sum of squares as in Eq. (35).

Exercises

Here we give several exercises that illustrate the methods described above. Exercise 1 is similar to the straight-line fit shown above and requires an extended error analysis. Exercises 2 and 3 use historical data on projectile motion from Galileo and on refraction of light from Ptolemy. The goodness-of-fit is analyzed to choose between competing hypotheses. Exercise 4 explores algorithms that can be used to find the minimum of $\chi^2(\boldsymbol{\theta})$ numerically.

Exercise 1: Polynomial fit and error analysis

Consider the following set of (x, y, σ) data points:

```
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

1(a): Using these data carry out the least-squares fit of an Mth order polynomial, i.e., with $M + 1$ adjustable parameters, for $M = 1, 2, 3$.

1(b): For each fit, use the error propagation formula Eq. (26) to find the standard deviation of the fitted function σ_f as a function of x. Note that to do this you will need to compute the derivatives of $f(x;\hat{\boldsymbol{\theta}})$ with respect to the components of $\hat{\boldsymbol{\theta}}$. Display the fitted curve plus-or-minus one standard deviation as a shaded band, and extend the x axis to at least 20. (The shaded band can be made with the function `matplotlib.fill_between`.) The result for $M = 1$ is shown in Fig. 9. Note how the size of the error band increases when one goes to x values outside the region where data are available; investigate how this behaviour changes as the order of the polynomial is increased.

1(c): Consider the fit with $M = 3$ and define the difference

$$\Delta_{ab}(\hat{\boldsymbol{\theta}}) = f(a;\hat{\boldsymbol{\theta}}) - f(b;\hat{\boldsymbol{\theta}}) . \tag{38}$$

Using error propagation, find the standard deviation of $\Delta_{ab}(\hat{\boldsymbol{\theta}})$ for $a = 5$ and $b = 6, 10, 20$. Compare these values you find with the standard deviation of f that you plotted for this fit as a shaded band.

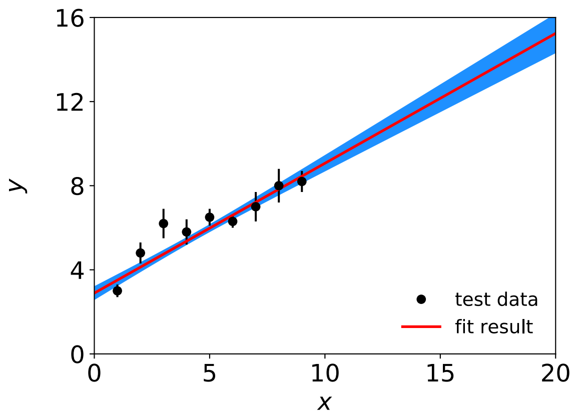


Figure 9: Result of least-squares fit with the one-sigma error shown as a shaded band.

Exercise 2: Analysis of Galileo's ball and ramp data

In this exercise we consider an experiment performed by Galileo using a ball and an inclined ramp as shown in Fig. 10. The ball starts at a height h above the edge of the ramp, and its trajectory is forced to be horizontal before it falls over the edge. The horizontal distance d from the edge to the point of impact is measured for different values of h. Five data points obtained by Galileo in 1608 are shown in Table 1 (from Ref. [4]).

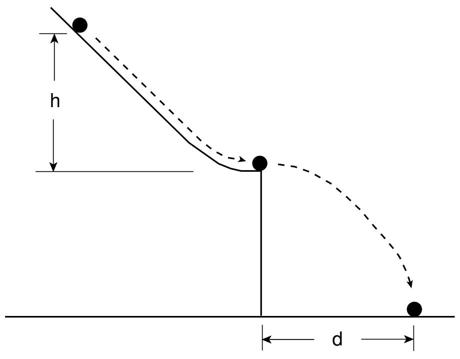


Figure 10: The configuration Galileo's ball and ramp experiment.

Table 1: Galileo's data on horizontal distance before impact d for five values of the starting height h. The units are punti (points); one punto is slightly less than 1 mm.

h	d
1000	1500
828	1340
800	1328
600	1172
300	800

We will assume the heights h are known with negligible error, and that the horizontal distances d have uncertainties of $\sigma = 15$ punti (points); one punto is slightly less than 1 mm. It is not actually known what the measurement uncertainties were, but 1–2% is plausible. In addition, we know that if $h = 0$, then the horizontal distance d will be zero, i.e. if the ball is started at the very edge of the ramp, it will fall straight down to the floor.

We could proceed by applying the known laws of mechanics to the system and deriving the relationship between d and h. For purposes of this exercise, however, we will pretend we do not yet know Newton's laws (as was the case for Galileo), and we will simply try different hypotheses and compare their predictions with the data.

Consider the following three hypotheses for the functional relationship between d and h: a linear relation with a single free parameter α,

$$d = \alpha h \,, \tag{39}$$

a quadratic relation with two parameters, α and β,

$$d = \alpha h + \beta h^2, \tag{40}$$

and something which is a nonlinear function of the parameters,

$$d = \alpha h^{\beta} \,. \tag{41}$$

The goals of this exercise are to investigate these three hypotheses. More specifically, you should write a Python program to do the following:

2(a) Carry out a least-squares fit of the parameters for each hypothesis. Find the estimated parameter values, their standard deviations and if relevant their covariance and correlation coefficient.

2(b) Produce plots of the fitted curves along with the data points and their errors.

2(c) Quantify the level of agreement between the data and each hypothesis with both a p-value and the chi-squared per degree of freedom. Discuss which if of the hypotheses are disfavoured and which are preferred.

2(d) Use Newton's Laws to derive the relation between d and h and predict the values of the parameters. Discuss how you would compare this prediction to the fit results obtained above. What parameters of the system can be related using your theoretical prediction to the fitted parameters? What important experimental uncertainties could render such a comparison difficult?

Exercise 3: Analysis of refraction data from Ptolemy

The astronomer Claudius Ptolemy performed experiments on the refraction of light using a circular copper disc submerged to its centre in water, as illustrated in Fig. 11. Angles of refraction $\theta_{\rm r}$ for 8 values of the angle of incidence $\theta_{\rm i}$ obtained by Ptolemy around 140 A.D. are shown in Table 2 (from Ref. [5]).

For purposes of this exercise we will take the angles of incidence to be known with negligible error and treat the angles of reflection as independent Gaussian-distributed measurements with standard deviations of $\sigma = \frac{1}{2}^{\circ}$. (This is a reasonable guess given that the angles are reported to the nearest half degree. Note that we can absorb an error in $\theta_{\rm i}$ into an effective error in $\theta_{\rm r}$.)

Until the discovery of the correct law of refraction (see below), a commonly used hypothesis was

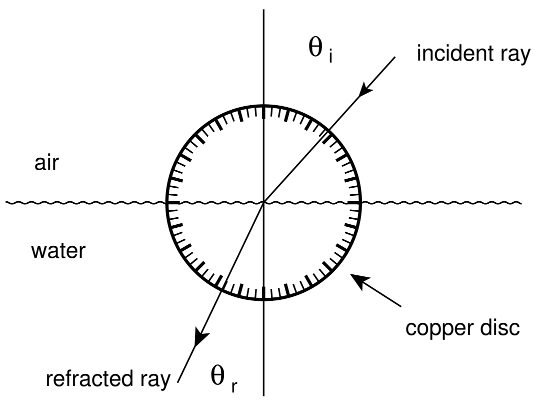


Figure 11: The apparatus used by Ptolemy to investigate the refraction of light.

Table 2: Angles of incidence and refraction (in degrees).

θ_i	θ_r
10	8
20	$15\frac{1}{2}$
30	$22\frac{1}{2}$
40	29
50	35
60	$40\frac{1}{2}$
70	$45\frac{1}{2}$
80	50

$$\theta_\mathrm{r} = \alpha\theta_\mathrm{i}, \tag{42}$$

although it is reported that Ptolemy preferred the formula

$$\theta_\mathrm{r} = \alpha\theta_\mathrm{i} - \beta\theta_\mathrm{i}^2. \tag{43}$$

3(a): Find the least-squares estimates of the parameters for both hypotheses and determine the minimized χ^2. Comment on the goodness-of-fit for both hypotheses. Is it plausible that all of the data values are based on actual measurements?[2]

The law of refraction discovered by the Persian mathematician and physicist Ibn Sahl in the 10th century and rediscovered by others including Snell in 1621 is

$$\theta_\mathrm{r} = \sin^{-1}\left(\frac{\sin\theta_\mathrm{i}}{r}\right), \tag{44}$$

where $r = n_\mathrm{r}/n_\mathrm{i}$ is the ratio of indices of refraction of the two media.

3(b): Determine the least-squares estimate for r and find value of the minimized χ^2. Comment on the validity of the Gaussian assumption for θ_r with $\sigma = \frac{1}{2}^\circ$.

For all of the fits, report the parameter estimates, their standard deviations, and where relevant the covariance and correlation coefficient. Make plots of the fitted curves and the data.

[2]See R. Feynman, R. Leighton and M. Sands, *The Feynman Lectures on Physics*, Vol. I, Addison-Wesley, Menlo Park, 1963, Section 26-2.

Exercise 4: Algorithms for numerical minimization

In most applications we use a specialised software package such as `scipy.optimize` to minimize an objective function like $\chi^2(\boldsymbol{\theta})$. It is nevertheless an interesting exercise to create such a routine yourself. The goal here is to implement a minimization algorithm in Python and to compare its results to those obtained from `curve_fit`.

Because the mathematics of function minimization can be treated separately from the method of least squares, here we will change notation and suppose that we have an objective function $f(\mathbf{x})$ of an n-dimensional argument $\mathbf{x} = (x_1, \ldots, x_n)$. The goal is to find the point in $\mathbf{x}$-space that gives the minimum of $f(\mathbf{x})$. We will assume in doing so that the analyst supplies a starting point $\mathbf{x}_\mathrm{s}$, which can be based on an initial guess.

All methods that achieve this goal require that one be able to evaluate the objective function f at an arbitrary point $\mathbf{x}$. If, in addition, one is able to evaluate the derivatives of f with respect to the components of $\mathbf{x}$ then algorithms can be used that will find the minimum much faster. In some cases one may know the derivatives in closed form; in other situations one may have to estimate them numerically. For the functions we have considered in the examples above it is relatively simple to calculate the first and second derivatives of $\chi^2(\boldsymbol{\theta})$ with respect to the parameters, and so for the method we implement here we will suppose that these are available.

One simple strategy to minimize $f(\mathbf{x})$ is to consider all of the components of $\mathbf{x}$ as fixed except one. Starting from the point $\mathbf{x}_0$, we vary only this chosen component to minimize f. We then repeat this procedure for all of the components of $\mathbf{x}$ in turn.

Consider first the function f as depending only on one non-constant component, which here for simplicity we will call x, without a subscript. Suppose that at a particular step of the algorithm the value of x is x_0, i.e., here the index refers to a step of the algorithm, not to the component. We can expand $f(x)$ in a Taylor series to second order about its initial value x_0, i.e.,

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2!} f''(x_0)(x - x_0)^2 \ . \tag{45}$$

To minimize this function with respect to x, we set its derivative equal to zero,

$$f'(x) \approx f'(x_0) + f''(x_0)(x - x_0) = 0 \ . \tag{46}$$

Solving for x gives

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)} \ . \tag{47}$$

This x is used to updated the corresponding component of $\mathbf{x}$, which in turn gives an updated estimate of the position of the minimum of $f(\mathbf{x})$. Having updated one component, we then cycle through all of the remaining ones and minimize with respect to each while holding the rest constant.

After carrying out minimizations with respect to all of the components we can evaluate the objective function and see how much it changed from its value at the previous full step. The procedure is repeated until the change in $f(\mathbf{x})$ from one iteration to the next falls below a desired threshold, which determines the final numerical accuracy.

One can easily generalise this algorithm to minimize not with respect to the individual coordinates, but rather along an arbitrary direction in $\mathbf{x}$-space. If we have available the derivatives of f, i.e,

$$\nabla f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) , \tag{48}$$

then we can generalise the procedure described above to minimize f along the direction of $-\nabla f$, i.e., the direction in which the function is decreasing most rapidly. This is called the *method of steepest descent.*

Although the method of steepest descent should eventually find the minimum of the objective function, in many cases it turns out to be faster to choose what are called *conjugate gradient* directions; more details can be found in Chapter 10 of Ref. [1].

4(a): The goal of this exercise is to create a Python program that will find the minimum of the objective function $\chi^2(\boldsymbol{\theta})$ numerically. Implement the algorithm with minimization along the coordinate directions as described above and test it by fitting a polynomial to the data shown in Sec. 6. Use different orders for the polynomial and thus different numbers of fitted parameters. Compare the results you obtain with those from the routine `curve_fit` and comment on any differences.

4(b): Extend exercise 4(a) by using the method of steepest descent.

References

[1] S. Brandt, *Statistical and Computational Methods in Data Analysis*, Springer, New York (1997).

[2] G. Cowan, *Statistical Data Analysis*, Oxford University Press, 1998.

[3] RHUL Physics Department, *Mathematical Formula Sheet for Physics* (2017); available on moodle.

[4] Stillman Drake and James Maclachlan, Galileo's discovery of the parabolic trajectory, *Scientific American* **232** (March 1975) 102; Stillman Drake, *Galileo at Work*, University of Chicago Press, Chicago (1978).

[5] Olaf Pedersen and Mogens Pihl, *Early Physics and Astronomy: A Historical Introduction*, MacDonald and Janes, London, 1974.

G. Cowan
RHUL Physics
Version 1.0 / September 2017

PH3010 / MSci Skills Miniproject

Statistical Data Analysis

Week 1: The Method of Least Squares

Autumn term 2017

Glen D. Cowan
RHUL Physics

The Statistical Data Analysis Module

This module on Statistical Data Analysis contains two parts:

Curve fitting with the method of least squares (weeks 1, 2)

Introduction to Machine Learning (week 3)

You will be given a number of exercises that should be written up in the form of a mini-project report. The standard rules apply:

Use the LaTeX template from the PH3010 moodle page.

Word limit is 3000, not including appendices.

All code should be submitted as an appendix.

The exercises for the least-squares part of the module are at the end of the script on moodle. Core exercises are numbers 1, 2, 3.

There may be some adjustment of the assigned exercises depending on how fast we are able to get through the material. (Exercise 4 may become optional.)

Outline (part 1)

Today:

Basic ideas of fitting a curve to data

The method of least squares

Finding the fitted parameters

Find the statistical errors of the fitted parameters

Using error propagation

Start of exercises

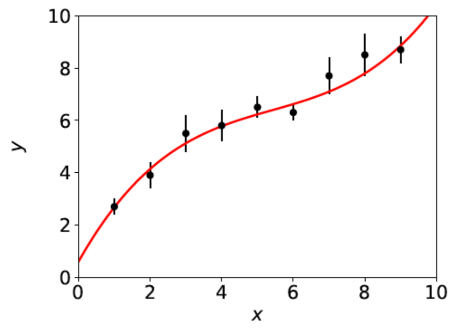
Next week:

goodness-of-fit, fitting correlated data, more exercises

Curve fitting: the basic problem

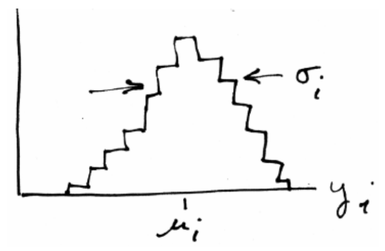
Suppose we have a set of N measured values y_i, $i = 1,.., N$.

Each y_i has an "error bar" σ_i, and is measured at a value x_i of a control variable x known with negligible uncertainty:



Roughly speaking, the goal is to find a curve that passes "close to" the data points, called "curve fitting".

Measured values → random variables



We will regard the measured y_i as independent observations of random variables (r.v.s).

Idea of an r.v.: imagine making repeated observations of the same y_i , and put these in a histogram:

The distribution of y_i has a mean μ_i and standard deviation σ_i.

We only know the data values y_i from a single measurement, i.e., we do not know the μ_i (goal is to estimate this).

y_i = measured value
x_i = control variable value
μ_i = "true value" (unknown)
σ_i = error bar ← suppose these are known

Fitting a curve

The standard deviation σ_i reflects the reproducibility (statistical error) of the measurement y_i.

If σ_i were to be very small, we can imagine that y_i would we be very close to its mean μ_i, and lie on a smooth curve given by some function of the control variable, i.e., $\mu_i = f(x_i)$.

Goal is to find the function $f(x)$. Here we will assume that we have some hypothesis for its functional form, but that it contains some unknown constants (parameters), e.g., a straight line:

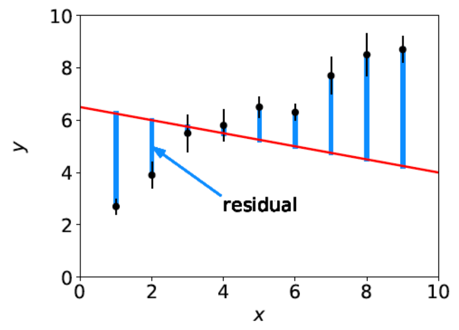
$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

vector of parameters $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_m)$

Curve fitting is thus reduced to estimating the parameters.

Least Squares: main idea

Consider fitting a straight line ad suppose we pick an arbitrary point in parameter space (θ_0, θ_1), which gives a certain curve:



Here the curve does not describe the data very well, as can be seen by the large values for the residuals:

$$\text{residual of } i\text{th point} = y_i - f(x_i;\boldsymbol{\theta})$$

Minimising the residuals

If a measured value y_i has a small σ_i, we want it to be closer to the curve, i.e., measure the distance from point to curve in units of σ_i:

$$\text{normalized residual of } i\text{th point} = \frac{y_i - f(x_i;\boldsymbol{\theta})}{\sigma_i}$$

Idea of the method of Least Squares is to choose the parameters that give the minimum of the sum of squared normalized residuals, i.e., we should minimize the "chi-squared":

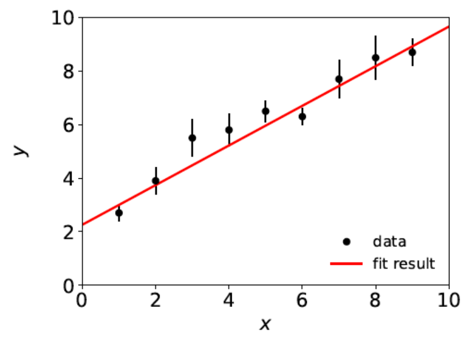
$$\chi^2(\boldsymbol{\theta}) = \sum_{i=1}^{N} \frac{(y_i - f(x_i;\boldsymbol{\theta}))^2}{\sigma_i^2}$$

Least squares estimators

The values that minimize $\chi^2(\boldsymbol{\theta})$ are called the least-squares *estimators* for the parameters, written with hats:

$$\hat{\boldsymbol{\theta}} = (\hat{\theta}_0, \hat{\theta}_1)$$

The fitted curve is thus “best” in the least-squares sense:



$$\hat{\theta}_0 = 2.258$$

$$\hat{\theta}_1 = 0.741$$

Comments on LS estimators

We can derive the method of Least Squares from a more general principle called the method of Maximum Likelihood applied to the special case where the y_i are independent and Gaussian distributed:

$$y_i \sim \text{Gauss}(\mu_i, \sigma_i) \; , \qquad \mu_i = f(x_i; \boldsymbol{\theta})$$

It is equally valid to take the minimum of $\chi^2(\boldsymbol{\theta})$ as the definition of the least-squares estimators, and in fact there is no general rule for finding estimators for parameters that are optimal in every relevant sense.

Next steps

1. How do we find the estimators, i.e., how do we minimize $\chi^2(\boldsymbol{\theta})$?
2. How do we quantify the statistical uncertainty in the estimated parameters that stems from the random fluctuations in the measurements, and how is this information used in an analysis problem, e.g., using error propagation?
3. How do we assess whether the hypothesized functional form $f(x; \boldsymbol{\theta})$ adequately describes the data?

Finding estimators in closed form

For a limited class of problem it is possible to find the estimators in closed form. An important example is when the function $f(x; \boldsymbol{\theta})$ is linear *in the parameters* $\boldsymbol{\theta}$, e.g., a polynomial of order M (note the function does not have to be linear in x):

$$f(x; \boldsymbol{\theta}) = \sum_{n=0}^{M} \theta_n x^n$$

As an example consider a straight line (two parameters):

$$f(x; \boldsymbol{\theta}) = \theta_0 + \theta_1 x$$

We need to minimize: $$\chi^2(\theta_0, \theta_1) = \sum_{i=1}^{N} \frac{(y_i - \theta_0 - \theta_1 x_i)^2}{\sigma_i^2}$$

Finding estimators in closed form (2)

Set the derivatives of $\chi^2(\boldsymbol{\theta})$ with respect to the parameters equal to zero:

$$\frac{\partial \chi^2}{\partial \theta_0} = \sum_{i=1}^{N} \frac{-2(y_i - \theta_0 - \theta_1 x_i)}{\sigma_i^2} = 0 \,,$$

$$\frac{\partial \chi^2}{\partial \theta_1} = \sum_{i=1}^{N} \frac{-2x_i(y_i - \theta_0 - \theta_1 x_i)}{\sigma_i^2} = 0 \,.$$

Finding estimators in closed form (3)

The equations can be rewritten in matrix form as

$$\begin{pmatrix} \sum_{i=1}^N \frac{1}{\sigma_i^2} & \sum_{i=1}^N \frac{x_i}{\sigma_i^2} \\ \sum_{i=1}^N \frac{x_i}{\sigma_i^2} & \sum_{i=1}^N \frac{x_i^2}{\sigma_i^2} \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^N \frac{y_i}{\sigma_i^2} \\ \sum_{i=1}^N \frac{x_i y_i}{\sigma_i^2} \end{pmatrix}$$

which has the general form

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} \theta_0 \\ \theta_1 \end{pmatrix} = \begin{pmatrix} e \\ f \end{pmatrix}$$

Read off a, b, c, d, e, f, by comparing with eq. above.

Finding estimators in closed form (4)

Recall how to invert a 2×2 matrix:

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} , \qquad A^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Apply A^{-1} to both sides of previous eq. to find solution (written with hats, because these are the estimators):

$$\hat{\theta}_0 = \frac{de - bf}{ad - bc} \,,$$

$$\hat{\theta}_1 = \frac{af - ec}{ad - bc} \,.$$

Comments on solution when $f(x;\boldsymbol{\theta})$ is linear in the parameters

Finding solution requires solving a system of linear equations; can be done with standard matrix methods.

Estimators are linear functions of the y_i. This is true in general for problems of this type with an arbitrary number of parameters.

Even though we could find the solution in closed form, the formulas get a bit complicated.

If the fit function $f(x; \boldsymbol{\theta})$ is not linear in the parameters, it is not always possible to solve for the estimators in closed form.

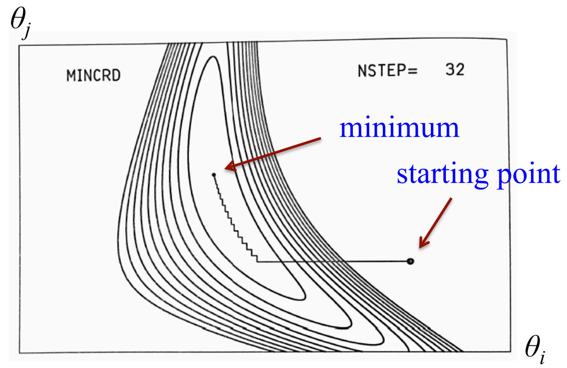
So for many problems we need to find the solution numerically.

Finding LS estimators numerically

Start at a given point in the parameter space and move around according to some strategy to find the point where $\chi^2(\boldsymbol{\theta})$ is a minimum.

For example, alternate minimizing with respect to each component of $\boldsymbol{\theta}$:

Many strategies possible, e.g., steepest descent, conjugate gradients, ... (see Brandt Ch. 10).



Siegmund Brandt, Data Analysis: Statistical and Computational Methods for Scientists and Engineers 4th ed., Springer 2014

Fitting the parameters with Python

The routine routine **`curve_fit`** from **`scipy.optimize`** can find LS estimators numerically. To use it you need:

```
import numpy as np
from scipy.optimize import curve_fit
```

We need to define the fit function $f(x; \boldsymbol{\theta})$, e.g., a straight line:

```
def func(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x
```

Fitting the parameters with Python (2)

The data values (x_i, y_i, σ_i) need to be in the form of NumPy arrays, e.g,

```
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

Start values of the parameters can be specified:

```
p0 = np.array([1.0, 1.0])
```

To find the parameter values that minimize $\chi^2(\boldsymbol{\theta})$, call **`curve_fit`**:

```
thetaHat, cov = curve_fit(func, x, y, p0, sig, absolute_sigma=True)
```

Returns estimators and covariance matrix as NumPy arrays.

Need **`absolute_sigma=True`** for the fit errors (cov. matrix) to have desired interpretation.

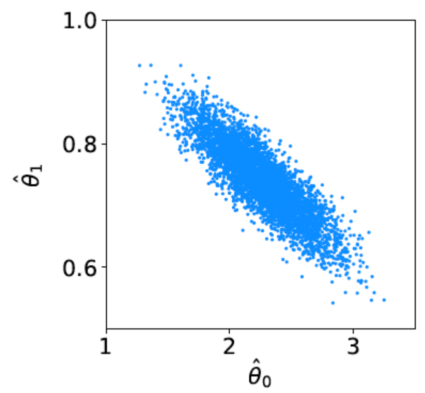
Statistical errors of fitted parameters

The estimators have statistical errors that are due to the random nature of the measured data (the y_i).

If we were to obtain a new independent set of measured values, $y_1,..., y_N$, then these would in general give different values for the estimated parameters.

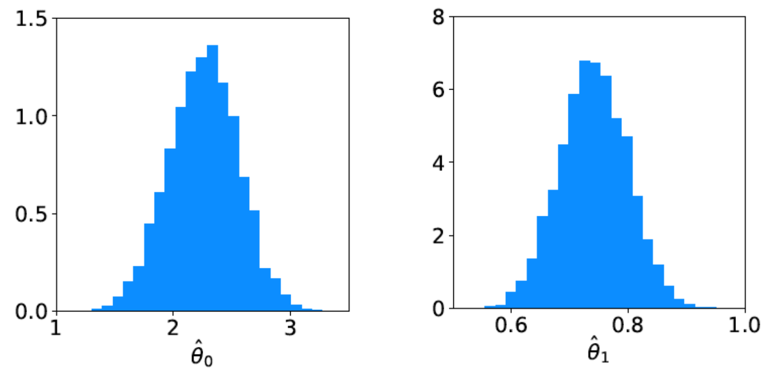
We can simulate the data set $y_1,..., y_N$ many times with the Monte Carlo method.

For each set evaluate the estimators for θ_0 and θ_1 from the straight-line fit and enter into a scatter plot:



Statistical errors of fitted parameters (2)

Project points onto the $\hat{\theta}_0$ and $\hat{\theta}_1$ axes:



Each distribution's standard deviation (~width) is used as a measure of the corresponding estimator's statistical error.

(Co)variance, correlation

The scatter plot of $\hat{\theta}_0$ versus $\hat{\theta}_1$ showed that if one estimate came out high, the other tended to be low and vice versa. This indicates that the estimators are (negatively) *correlated*.

To quantify the degree of correlation in any two random variables *u* and *v* we define the *covariance*,

$$\mathrm{cov}[u,v] = \langle uv \rangle - \langle u \rangle \langle v \rangle = \langle (u - \langle u \rangle)(v - \langle v \rangle) \rangle$$

The covariance of a variable *u* with itself is its variance $V[u] = \sigma_u^2$

$$\mathrm{cov}[u,u] = \langle u^2 \rangle - \langle u \rangle^2 = V[u] = \sigma_u^2$$

The square root of the variance = standard deviation σ_u.

Also define dimensionless correlation coefficient (can show $-1 \le \rho \le 1$):

$$\rho = \frac{\mathrm{cov}[u,v]}{\sigma_u \sigma_v}$$

Covariance etc. from straight-line fit

From the simulated values shown in the scatter plot, use standard formulae (see RHUL Physics formula book) to obtain the standard deviations and covariance:

$$\begin{aligned} \sigma_{\hat{\theta}_0} &= 0.29\,, \\ \sigma_{\hat{\theta}_1} &= 0.057\,, \\ \mathrm{cov}[\hat{\theta}_0, \hat{\theta}_1] &= -0.0142\,, \\ \rho &= -0.86\,. \end{aligned}$$

Covariance matrix

If we have a set of estimators

$$\hat{\boldsymbol{\theta}} = (\hat{\theta}_1, \dots, \hat{\theta}_m)$$

we can find the covariance for each pair and put into a matrix

$$U_{ij} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_j]$$

Covariance matrix is square and symmetric.

Diagonal elements are the variances: $U_{ii} = \mathrm{cov}[\hat{\theta}_i, \hat{\theta}_i] = \sigma^2_{\hat{\theta}_i}$

The vector of estimators and their covariance matrix are the two objects returned by the routine **`curve_fit`**:

```
thetaHat, cov = curve_fit(func, x, y, p0, sig, absolute_sigma=True)
```

Covariance from derivatives of $\chi^2(\boldsymbol{\theta})$

It is also possible to obtain the covariance matrix from second derivatives of $\chi^2(\boldsymbol{\theta})$ with respect to the parameters at its minimum.

First find U^{-1},

$$U_{ij}^{-1} = \frac{1}{2}\left(\frac{\partial^2 \chi^2}{\partial\theta_i \partial\theta_j}\right)_{\boldsymbol{\theta}=\hat{\boldsymbol{\theta}}}$$

and then invert to find the covariance matrix U.

This is what `curve_fit` does (derivatives computed numerically). Example with straight-line fit gives:

$$U = \begin{pmatrix} 0.08537 & -0.01438 \\ -0.01438 & 0.003275 \end{pmatrix}$$

Using the covariance matrix: error propagation

Suppose we've done a fit for parameters (θ_1,..., θ_m) and obtained estimators and their covariance matrix.

We may then be interested in a given function of the fitted parameters, e.g.,

$$u(\hat{\boldsymbol{\theta}}) = (\hat{\theta}_1 \hat{\theta}_2 - 2\hat{\theta}_3)^2$$

What is the standard deviation of the quantity u? That is, how do we *propagate* the statistical errors in the estimated parameters through to u?

Or suppose we have two functions u and v. What is are their standard deviations and what is their covariance cov[u,v]?

The error propagation formulae

By expanding the functions to first order about the parameter estimates, one can show that the covariance is approximately

$$\mathrm{cov}[u,v] \approx \sum_{i,j=1}^{m} \frac{\partial u}{\partial\hat{\theta}_i}\frac{\partial v}{\partial\hat{\theta}_j} U_{ij}$$

and thus the variance for a single function is

$$\sigma_u^2 \approx \sum_{i,j=1}^{m} \frac{\partial u}{\partial\hat{\theta}_i}\frac{\partial u}{\partial\hat{\theta}_j} U_{ij}$$

In the special case where the covariance matrix is diagonal, $U_{ij} = \sigma_i \sigma_j \delta_{ij}$, we can carry out one of the sums to find

$$\sigma_u^2 \approx \sum_{i=1}^{m} \left(\frac{\partial u}{\partial\hat{\theta}_i}\right)^2 \sigma_{\hat{\theta}_i}^2$$

Comments on error propagation

In general the estimators from a fit *are correlated*, so their full covariance matrix must be used for error propagation.

The approximation of error propagation is that the functions are linear in a region of plus-or-minus one standard deviation about the estimators.

Simple example:

$$u = a\hat{\theta}_1 + b\hat{\theta}_2 \qquad \frac{\partial u}{\partial\hat{\theta}_1} = a \qquad \frac{\partial u}{\partial\hat{\theta}_2} = b$$

$$\begin{aligned} \sigma_u^2 &= a^2 U_{11} + b^2 U_{22} + abU_{12} + baU_{21} \\ &= a^2\sigma_{\hat{\theta}_1}^2 + b^2\sigma_{\hat{\theta}_2}^2 + 2ab\,\mathrm{cov}[\hat{\theta}_1, \hat{\theta}_2] \end{aligned}$$

Exercise 1: polynomial fit

Exercise 1: Polynomial fit and error analysis

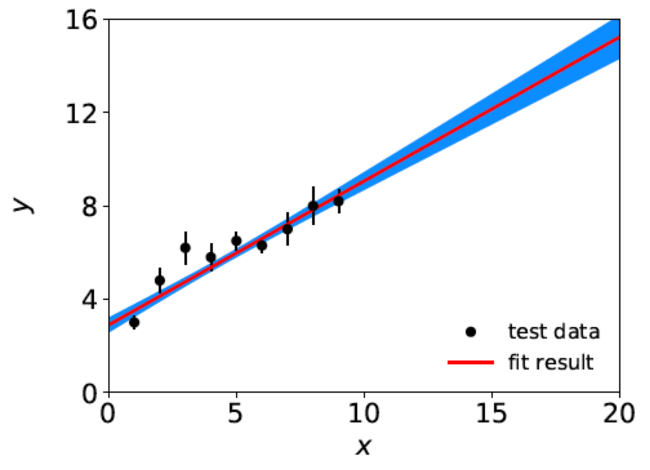
Consider the following set of (x, y, σ) data points:

```
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])
```

1(a): Using these data carry out the least-squares fit of an Mth order polynomial, i.e., with $M+1$ adjustable parameters, for $M = 1, 2, 3$.

1(b): For each fit, use the error propagation formula Eq. (26) to find the standard deviation of the fitted function σ_f as a function of x. Note that to do this you will need to compute the derivatives of $f(x;\hat{\boldsymbol{\theta}})$ with respect to the components of $\hat{\boldsymbol{\theta}}$. Display the fitted curve plus-or-minus one standard deviation as a shaded band, and extend the x axis to at least 20. (The shaded band can be made with the function `matplotlib.fill_between`.) The result for $M = 1$ is shown in Fig. 9. Note how the size of the error band increases when one goes to x values outside the region where data are available; investigate how this behaviour changes as the order of the polynomial is increased.

Polynomial fit: error band



Polynomial fit: error propagation

Consider the difference between the fitted curve values at $x = a$ and $x = b$:

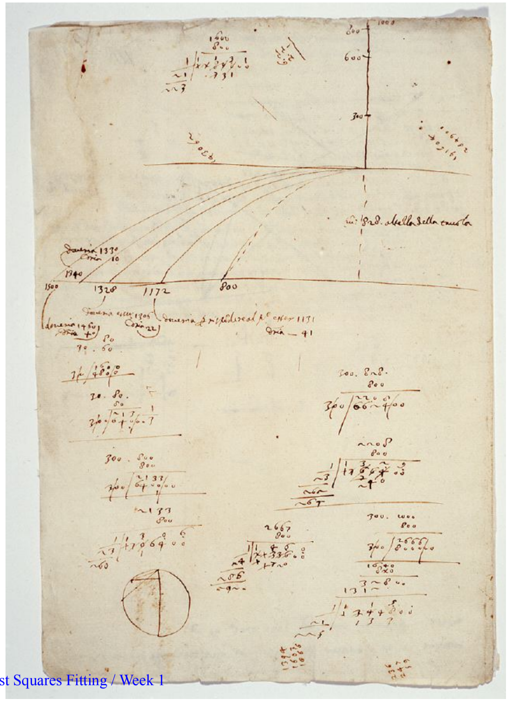
$$\Delta_{ab}(\hat{\boldsymbol{\theta}}) = f(a;\hat{\boldsymbol{\theta}}) - f(b;\hat{\boldsymbol{\theta}})$$

Use error propagation to find the standard deviation of Δ_{ab} (see script).

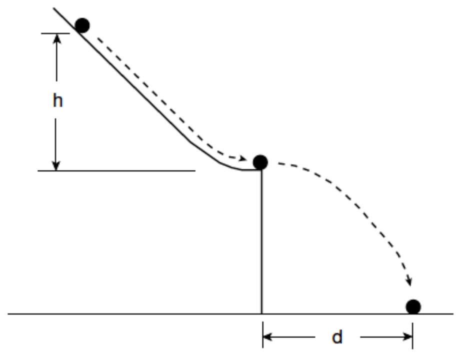
Ball and ramp data from Galileo

Galileo Galilei, Manuscript *f.*116, Biblioteca Nazionale Centrale di Firenze, bncf.firenze.sbn.it

In 1608 Galileo carried out experiments rolling a ball down an inclined ramp to investigate the trajectory of falling objects.



Ball and ramp data from Galileo



Units in punti
(approx. 1 mm)

h	d
1000	1500
828	1340
800	1328
600	1172
300	800

Suppose h is set with negligible uncertainty, and
d is measured with an uncertainty $\sigma = 15$ punti.

Analysis of ball and ramp data

What is the correct law that relates d and h?

Try different hypotheses:

$$d = \alpha h$$

$$d = \alpha h + \beta h^2$$

$$d = \alpha h^\beta$$

For now, fit the parameters α and β, find their standard deviations and covariance.

Next week we will discuss how to test whether a given hypothesized function is in good or bad agreement with the data.

Extra slides

History

Least Squares fitting also called “regression”

F. Galton, *Regression towards mediocrity in hereditary stature*, The Journal of the Anthropological Institute of Great Britain and Ireland. 15: 246–263 (1886).

Developed earlier by Laplace and Gauss:

C.F. Gauss, *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambentium*, Hamburgi Sumtibus Frid. Perthes et H. Besser Liber II, Sectio II (1809);

C.F. Gauss, *Theoria Combinationis Observationum Erroribus Minimis Obnoxiae*, pars prior (15.2.1821) et pars posterior (2.2.1823), Commentationes Societatis Regiae Scientiarium Gottingensis Recectiores Vol. V (MDCCCXXIII).

PH3010 / MSci Skills Miniproject

Statistical Data Analysis

Week 2: Goodness-of-fit,
LS fitting with correlated data,

Autumn term 2017

Glen D. Cowan
RHUL Physics

Outline – lecture 2

Today:

Goodness-of-fit

Fitting correlated data

More exercises

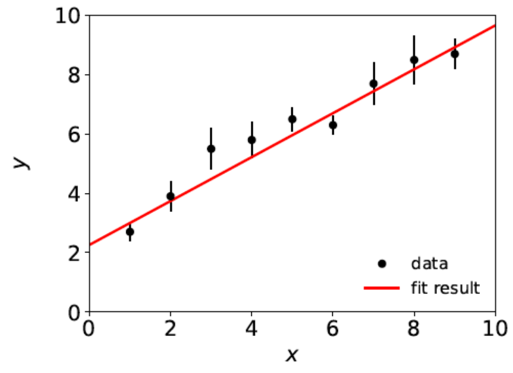
Discussion of project report

Next week:

Introduction to machine learning

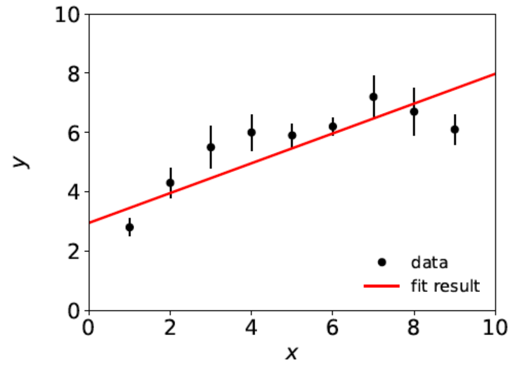
A “good” fit

Last week we fitted data that were reasonably well described by a straight line:



A “bad” fit

But what if a straight-line fit looks like this:



Maybe here we should fit a higher-order polynomial?

Goodness-of-fit: the questions

How do we quantify the level of agreement between the data and the hypothesized form of the fit function?

How do we decide whether to try a different fit function?

⚠ Note first the following common misunderstanding:

If the fit is "bad", you may expect large statistical errors for the fitted parameters. This is not the case.

The statistical errors say how much the parameter estimates will fluctuate under repetition of the experiment, under assumption of the hypothesized fit function. This is not the same as the degree to which the function is able to describe the data.

If the hypothesized $f(x; \boldsymbol{\theta})$ is not correct, the fitted parameters will have some systematic uncertainty – a more complex question that we will not take up here.

Quantifying goodness-of-fit

We can quantify the goodness-of-fit directly from the value of $\chi^2(\boldsymbol{\theta})$ evaluated at its minimum, i.e., at $\boldsymbol{\theta} = \hat{\boldsymbol{\theta}}$:

$$\chi^2_{\min} = \sum_{i=1}^{N} \frac{(y_i - f(x_i; \hat{\boldsymbol{\theta}}))^2}{\sigma_i^2}$$

If the fitted function is in good agreement with the data, then the numerator of each term in the sum should be small.

If $f(x_i; \hat{\boldsymbol{\theta}})$ were equal to the true mean of y_i, then we would expect the residual $y_i - f(x_i; \hat{\boldsymbol{\theta}})$ to have an rms value of σ_i. Each term in the sum would contribute 1, and we'd have $\chi^2_{\min} = N$.

This is not quite true: if we have fitted m parameters and the hypothesized function is correct, the *expected* value of $\chi^2_{\min}$ is $N - m$ (called the *number of degrees of freedom* of the fit.)

Distribution of $\chi^2_{\min}$

$\chi^2_{\min}$ is a function of the data so is itself a random variable.

If the hypothesized fit function is correct and the data are Gaussian distributed, one can show $\chi^2_{\min}$ follows a chi-square distribution with $n = N - m$ degrees of freedom (here let $\chi^2_{\min} = z$):

$$f_{\chi^2}(z; n) = \frac{1}{2^{n/2}\Gamma(n/2)} z^{n/2-1} e^{-z/2}$$

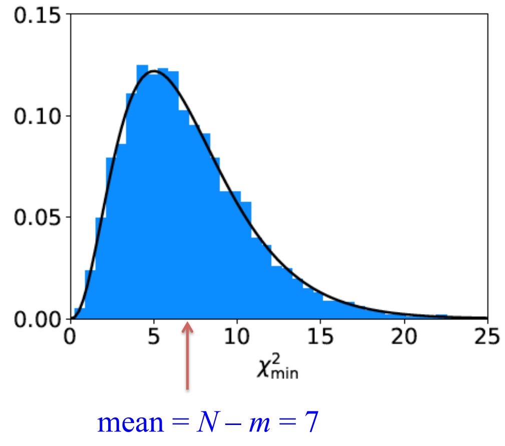
Mean and standard deviation of the chi-square distribution:

$$\langle z \rangle = n$$
$$\sigma_z = \sqrt{2n}$$

If the hypothesized fit function is not correct, then one would obtain a distribution of $\chi^2_{\min}$ shifted to higher values.

Distribution of $\chi^2_{\min}$ from straight-line fit

$\chi^2_{\min}$ from straight-line fit with $N = 9$ data points, $m = 2$ fitted parameters.



mean = $N - m = 7$

Curve: chi-square pdf for $n = 7$ degrees of freedom.

Histogram: values of $\chi^2_{\min}$ from straight-line fit from repeated Monte Carlo simulation of the experiment.

How to interpret χ^2_{min}

A simple way to assess the goodness-of-fit is simply to compare χ^2_{min} to the number of degrees of freedom, $n_{\text{dof}} = N - m$.

$\chi^2_{\text{min}} \sim n_{\text{dof}}$ $\rightarrow$ fit is “good”

$\chi^2_{\text{min}} \gg n_{\text{dof}}$ $\rightarrow$ fit is “bad”

$\chi^2_{\text{min}} \ll n_{\text{dof}}$ $\rightarrow$ fit is better than what one would expect given fluctuations that should be present in the data.

Often this is done using the ratio $\chi^2_{\text{min}}/n_{\text{dof}}$, i.e. fit is good if the “chi-square per degree of freedom” comes out not much greater than 1.

Often report as, e.g., $\chi^2_{\text{min}}/n_{\text{dof}} = 8.2/7$. It is best to communicate both χ^2_{min} and n_{dof}, not just their ratio.

p-value from χ^2_{min}

Another way to assess the goodness-of-fit is to give the probability, assuming the fit function is correct, to obtain a χ^2_{min} value as high as the one we got or higher:

$$p = \int_{\chi^2_{\text{min}}}^{\infty} f_{\chi^2}(z;n)\,dz$$

This is an example of what is called a p-value of the hypothesis (here the hypothesized form of the fit function).

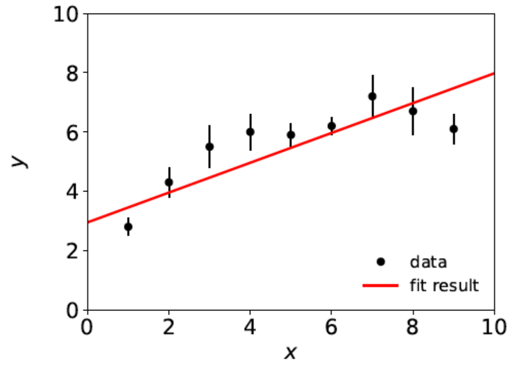
p-value is not the same as the probability that the hypothesis is true!

Nevertheless, a small p-value indicates that the hypothesis is disfavoured.

Compute using: `scipy.stats.chi2.sf`

χ^2_{min} from the “bad” fit

Straight-line fit with $N = 9$ data points, $m = 2$ fitted parameters.



$\chi^2_{\text{min}} = 20.9$ for $n_{\text{dof}} = 7$

$\chi^2_{\text{min}}/n_{\text{dof}} = 3.7$

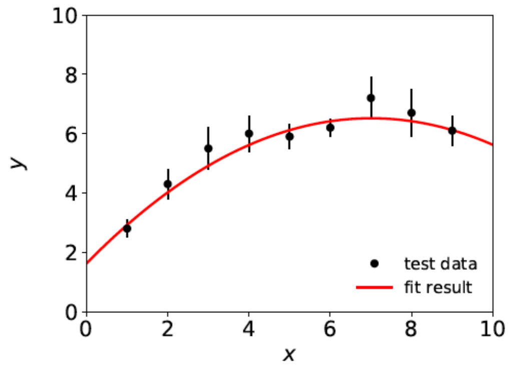
p-value = 0.0039

So is the straight-line hypothesis correct? It could be, but if so we would expect a χ^2_{min} as high as observed or higher only 4 times out of a thousand.

A better fit

If we decide the agreement between data and hypothesis is not good enough (exact threshold is a subjective choice), we can try a different model, e.g., a 2nd order polynomial:

$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2$$



$\chi^2_{\text{min}} = 3.5$ for $n_{\text{dof}} = 6$

$\chi^2_{\text{min}}/n_{\text{dof}} = 0.58$

p-value = 0.75

Least squares with correlated measurements

Up to now we have assumed that the measurements $y_1,\ldots,y_N$ are all independent.

This means that if one value fluctuates, say, high, then this has no influence on whether one of the others will fluctuate high or low.

But there could be cases where the y_i are correlated, i.e., they have nonzero covariances

$$\mathrm{cov}[y_i, y_j] = V_{ij} = \rho_{ij}\sigma_i\sigma_j$$

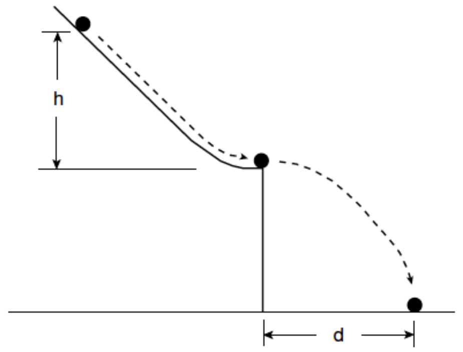
In this case, the formula for $\chi^2(\boldsymbol{\theta})$ becomes

$$\chi^2(\boldsymbol{\theta}) = \sum_{i,j=1}^{N} (y_i - f(x_i;\boldsymbol{\theta}))V_{ij}^{-1}(y_j - f(x_j;\boldsymbol{\theta}))$$

where V^{-1} is the inverse of the covariance matrix of the data V.

Goodness-of-fit with Galileo's data

Last week we used data from Galileo...



h	d
1000	1500
828	1340
800	1328
600	1172
300	800

Goodness-of-fit with Galileo's data

...to fit several hypotheses for the functional relation between the initial height h and flight distance d:

$$d = \alpha h$$

$$d = \alpha h + \beta h^2$$

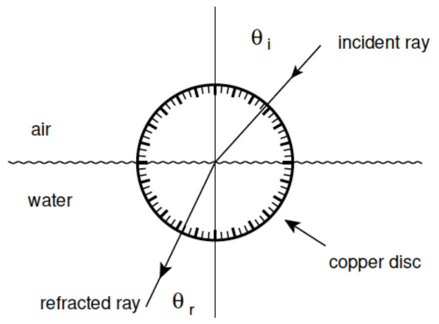
$$d = \alpha h^\beta$$

So now we can put ourselves in Galileo's position and, without knowledge of Newton's laws, find which hypotheses are compatible with or disfavoured by the data (find $\chi^2_{\mathrm{min}}/n_{\mathrm{dof}}$ and p-value).

You can also use your knowledge of Newtonian mechanics to work out the predicted law and compare to what you find empirically.

Exercise 3: refraction data from Ptolemy

Astronomer Claudius Ptolemy obtained data on refraction of light by water in around 140 A.D.:



Angles of incidence and refraction (degrees)

θ_{i}	θ_{r}
10	8
20	$15\frac{1}{2}$
30	$22\frac{1}{2}$
40	29
50	35
60	$40\frac{1}{2}$
70	$45\frac{1}{2}$
80	50

Suppose the angle of incidence is set with negligible error, and the measured angle of refraction has a standard deviation of ½°.

Laws of refraction

A commonly used law of refraction was

$$\theta_{\rm r} = \alpha \theta_{\rm i} \ ,$$

although it is reported that Ptolemy preferred

$$\theta_{\rm r} = \alpha \theta_{\rm i} - \beta \theta_{\rm i}^2 \ .$$

The law of refraction discovered by Ibn Sahl in 984 (and rediscovered by Snell in 1621) is

$$\theta_{\rm r} = \sin^{-1} \left(\frac{\sin \theta_{\rm i}}{r} \right) \ .$$

where $r = n_{\rm r}/n_{\rm i}$ is the ratio of indices of refraction of the two media.

Analysis of refraction data

Fit the parameters and find their statistical errors and (where relevant) covariance matrix.

Assess goodness-of-fit of each hypothesis ($\chi^2_{\rm min}/n_{\rm dof}$ and p-value).

What can you conclude?

(See The Feynman Lectures on Physics, Vol I., Addison-Wesley, 1963, Section 26-2, `www.feynmanlectures.caltech.edu` .)

Discussion of project report

Exercise 4 in the Least Squares script is optional.

There will be some exercises next week on Machine Learning.

Guidelines for report writing are on the Moodle page (section Reports 2017-2018), with LaTeX template, etc.

You should submit the electronic version of your report to the Turnitin Repository on the Moodle page Topic 6: Report Repository for Statistical Analysis Miniproject.

Deadline is 17 Nov 2017 at 10:00 a.m. (extended one week).

You should also submit 2 bound copies of the report to the departmental office (same deadline).

Do not put off writing to the last minute (start ~now).

Discussion of project report (2)

Your report should have

A short introduction

A section for each exercise

Brief conclusions

Bibliography

Appendices (including all code you’ve written)

Use the relevant tools in LaTeX for the components of the report (sections, figures, bibliography, etc.).

Maximum word count is (including captions but not including appendices) is 3000.

PH3010 / MSci Skills
Statistical Data Analysis

Week 3: Introduction to Machine Learning

Autumn term 2017

Glen D. Cowan
RHUL Physics

Outline – week 3

What Machine Learning is and how it can be applied

Classification of two types of “events”

Linear classifiers: Fisher Discriminant

Nonlinear classifiers: Neural Networks

Software for Machine Learning: scikit-learn

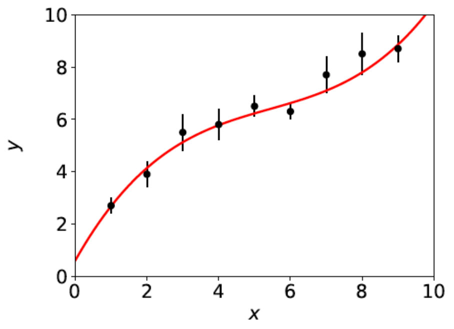
Exercises

What Machine Learning is

The term Machine Learning (ML) refers to algorithms that “learn from data” and make predictions based on what has been learnt.

In its simplest sense, “learning” means the algorithm contains adjustable parameters whose values are estimated using data.

Formally, curve fitting can be seen as Machine Learning.



Hypothesized curve:

$$f(x;\boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$$

Values of parameters are “learnt” from the data.

Fitted curve can make predictions at x values of future measurements.

More general Machine Learning

But generally ML refers to situations in which:

the hypothesized model is very general (e.g., not just a polynomial) and contains many adjustable parameters;

the quantity we want to predict could depend on many variables, e.g., not just x but a set $\boldsymbol{x} = (x_1,\ldots,x_n)$.

ML can be seen as a part of or related to:

Artificial Intelligence

Pattern Recognition

Statistical Learning

Multivariate Analysis

Development from (mainly) Computer Science, (also) Statistics; sometimes “Data Science” used to refer to all of above.

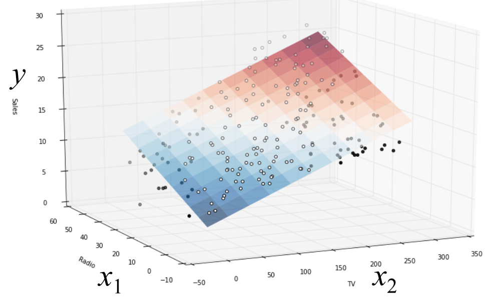
http://nbviewer.ipython.org/urls/s3.amazonaws.com/datarobotblog/notebooks/multiple_regression_in_python.ipynb

Curve fitting → regression

The generalisation of curve fitting with a multidimensional control variable $x \rightarrow \boldsymbol{x} = (x_1,..., x_n)$ is called multivariate regression.

The data consist of sets of points $(\boldsymbol{x}_i, y_i)$, where now $\boldsymbol{x}_i$ is a multidimensional vector, and usually the y_i does not come with an error bar.

Goal is to predict the expected y value for a new $\boldsymbol{x}$.



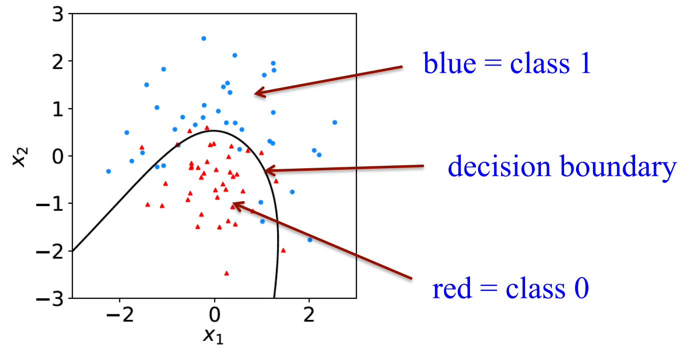
E.g. in two dimensions, multivariate regression means fitting a surface $f(x_1, x_2)$ to data points $(x_{1,i}, x_{2,i}, y_i)$:

Classification

A related type of ML algorithm is called classification: the data consist of points $(\boldsymbol{x}_i, y_i)$, where now y_i is discrete class label, (e.g., 0 or 1, red or blue, etc.).

Learning based on events with known y_i = supervised learning.

Goal is to create a decision boundary in $\boldsymbol{x}$-space so as to predict the class y of a new instance of $\boldsymbol{x}$.



Example of classification: Industrial Fishing

You scoop up fish which are of two types:

You examine the fish with automatic sensors and for each one you measure a set of features:

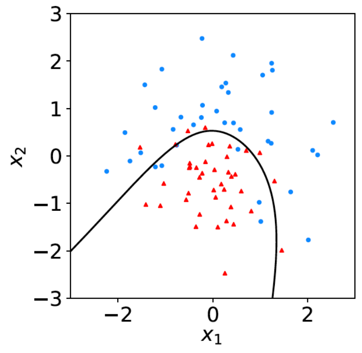
x_1 = length	x_4 = area of fins
x_2 = width	x_5 = mean spectral reflectance
x_3 = weight	x_6 = ...

These constitute the “feature vector” $\boldsymbol{x} = (x_1,..., x_n)$.

In addition you hire a fish expert to identify the “true class label” y = 0 or 1 (i.e., 0 =sea bass, 1 = cod) for each fish.

Distributions of the features

If we consider only two features $\boldsymbol{x} = (x_1, x_2)$, we can display the results in a scatter plot (red: $y = 0$, blue: $y = 1$).



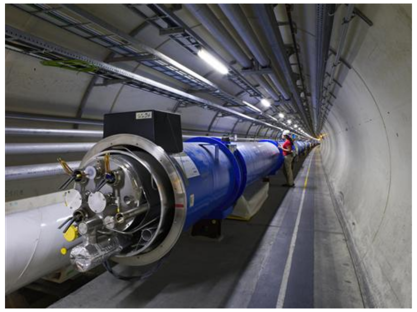
Goal is to determine a decision boundary, so that, without the help of the fish expert, we can classify new fish by seeing where their measured features lie relative to the boundary.

Same idea in multi-dimensional feature space, but cannot represent as 2-D plot. Decision boundary is n-dim. hypersurface.

Example use of Machine Learning: Particle Physics at the LHC

Counter-rotating proton beams in 27 km circumference ring

pp centre-of-mass energy 14 TeV

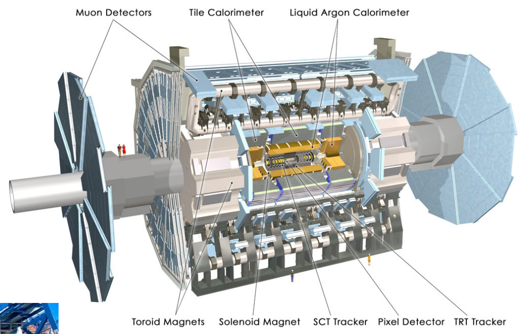


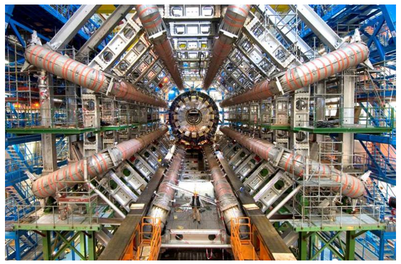
Detectors at 4 pp collision points:

ATLAS
CMS — general purpose
LHCb (b physics)
ALICE (heavy ion physics)

The ATLAS Detector at the LHC

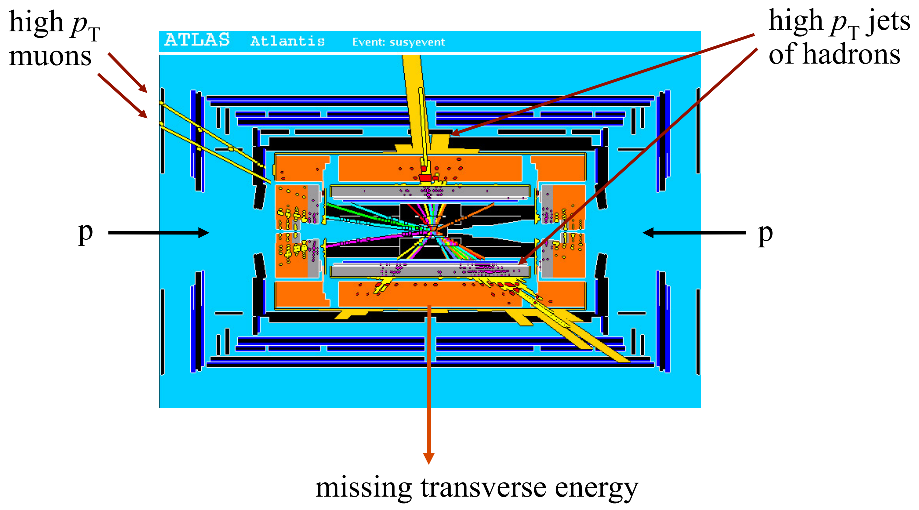
3000 physicists
37 countries
167 universities/labs



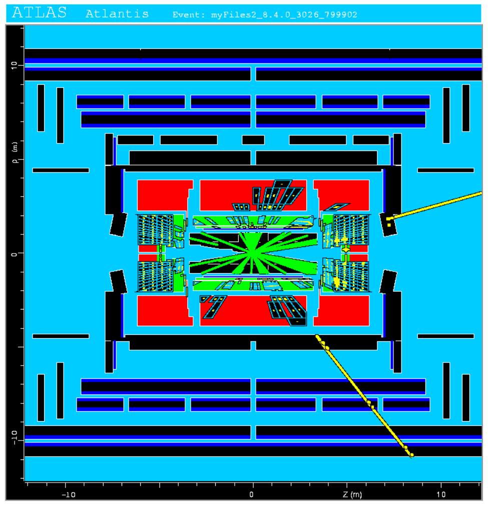


25 m diameter
46 m length
7000 tonnes
~10^8 electronic channels

A simulated proton-proton collision from supersymmetry (“signal”)


high p_T muons
ATLAS Atlantis Event: susyevent
high p_T jets of hadrons
p
p
missing transverse energy

A simulated proton-proton collision from production of top quarks (“background”)



This event has features similar to what we hope to see in supersymmetric events, and thus constitutes a “background” that can mimic the “signal”.

Classification of proton-proton collisions

Proton-proton collisions can be considered to come in two classes:

signal (the kind of event we're looking for, $y = 1$)
background (the kind that mimics signal, $y = 0$)

For each collision (event), we measure a collection of features:

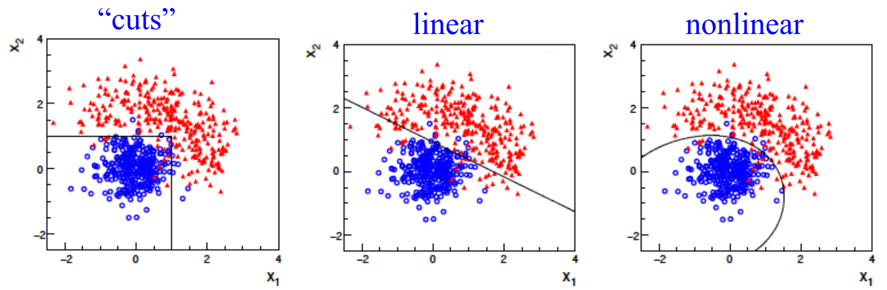
x_1 = energy of muon	x_4 = missing transverse energy
x_2 = angle between jets	x_5 = invariant mass of muon pair
x_3 = total jet energy	x_6 = ...

The real events don't come with true class labels, but computer-simulated events do. So we can have a set of simulated events that consist of a feature vector $\boldsymbol{x}$ and true class label y (0 for background, 1 for signal):

$$(\boldsymbol{x}, y)_1, (\boldsymbol{x}, y)_2, \ldots, (\boldsymbol{x}, y)_N$$

The simulated events are called "training data".

What is the best decision boundary?



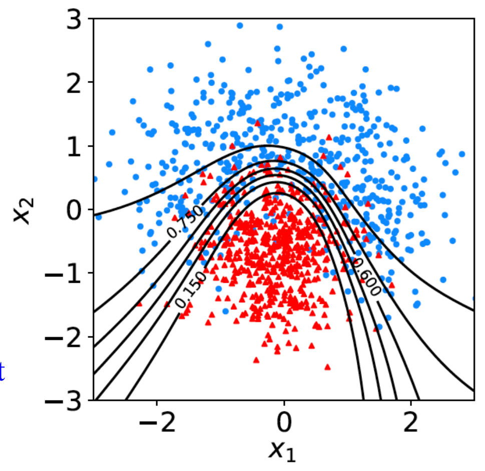
What is the best decision function?

A surface in an n-dimensional space can be described by

$$t(x_1, \ldots, x_n) = t_c$$

scalar function

constant



Different values of the constant t_c result in a family of surfaces.

Problem is reduced to finding the best decision function $t(\boldsymbol{x})$.

Linear decision boundary

A simple *Ansatz* is to try a decision function of the form

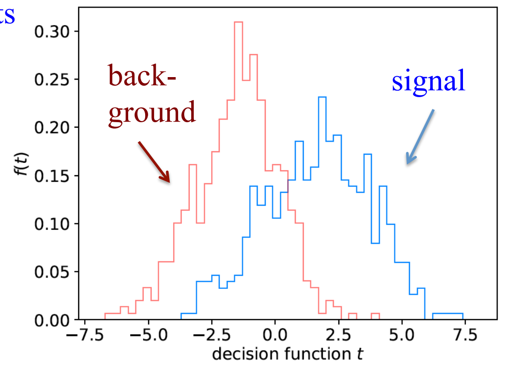
$$t(\mathbf{x}) = \sum_{i=1}^{n} w_i x_i$$

where the coefficients $w_1, \ldots, w_n$ are constants (or "weights") we need to determine using the training data.

A given choice of the weights fixes the function $t(\boldsymbol{x})$.

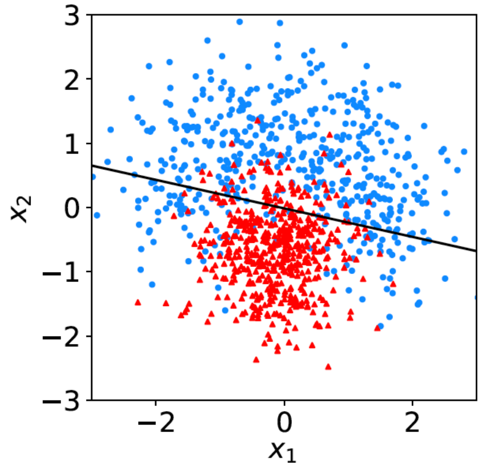
Look at the training events from the two classes, for each evaluate $t(\boldsymbol{x})$ and enter into a histogram.

Goal is to maximize the "separation" between the two distributions.



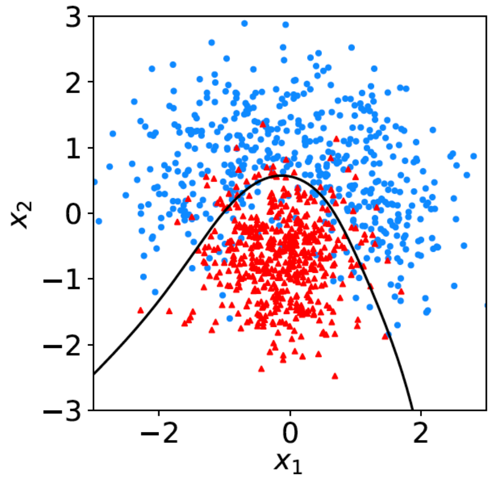
Fisher discriminant

Using a particular definition of what constitutes "best separation" due to R. Fisher one obtains a *Fisher discriminant*:



Nonlinear decision boundaries

From the scatter plot below it's clear that some nonlinear boundary would be better than a linear one:



And to have a nonlinear boundary, the decision function $t(\boldsymbol{x})$ must be nonlinear in $\boldsymbol{x}$.

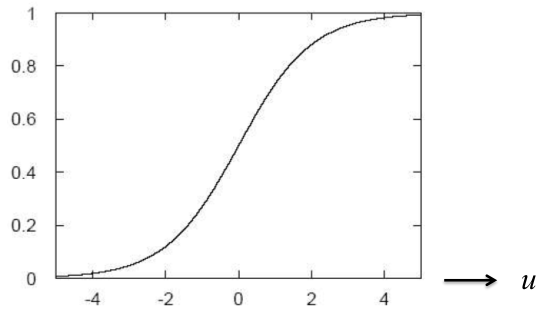
Neural Networks

A simple nonlinear decision function can be constructed as

$$t(\mathbf{x}) = h\left(w_0 + \sum_{i=1}^{n} w_i x_i\right)$$

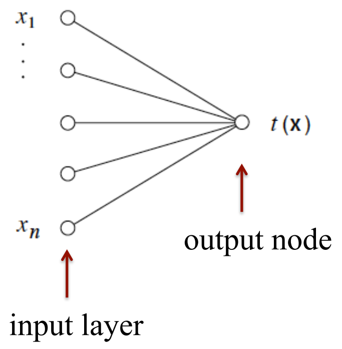
where h is called the "activation function". For this one can use, e.g., a logistic sigmoid function,

$$h(u) = \frac{1}{1+e^{-u}}$$



Single Layer Perceptron

In this form, the decision function is called a Single Layer Perceptron – the simplest example of a Neural Network.



But the surface described by $t(\boldsymbol{x}) = t_c$ is the same as by

$$h^{-1}(t(\mathbf{x})) = w_0 + \sum_{i=1}^{n} w_i x_i = h^{-1}(t_c)$$

So here we still have a linear decision boundary.

Multilayer Perceptron

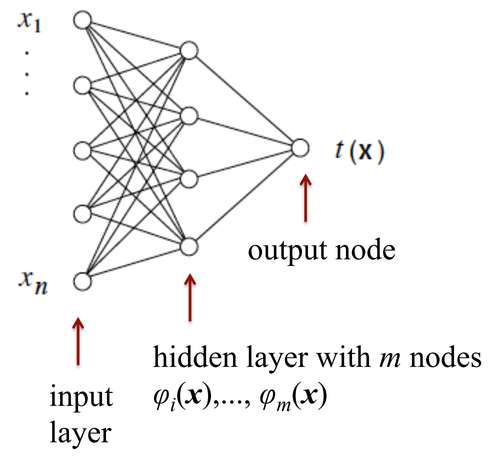
The Single Layer Perceptron can be generalized by defining first a set of functions $\varphi_i(\mathbf{x})$, with $i = 1,\dots, m$:

$$\varphi_i(\mathbf{x}) = h\left(w_{i0}^{(1)} + \sum_{j=1}^{n} w_{ij}^{(1)} x_j\right) \qquad i = 1,\dots,m$$

The $\varphi_i(\mathbf{x})$ are then treated as if they were the input variables, in a perceptron, i.e., the decision function (output node) is

$$t(\mathbf{x}) = h\left(w_{10}^{(2)} + \sum_{j=1}^{n} w_{1j}^{(2)} \varphi_j(\mathbf{x})\right)$$

Multilayer Perceptron (2)



x_1
x_n
$t(\mathbf{x})$
output node
input layer
hidden layer with m nodes
$\varphi_i(\mathbf{x}),\dots, \varphi_m(\mathbf{x})$

Each line in the graph represents one of the weights $w_{ij}^{(k)}$, which must be adjusted using the training data.

Training a Neural Network

To train the network (i.e., determine the best values for the weights), define a loss function, e.g.,

$$E(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N} |t(\mathbf{x}_i) - y_i|^2$$

where $\boldsymbol{w}$ represents the set of all weights, the sum is over the set of training events, and y_i is the (numeric) true class label of each event (0 or 1).

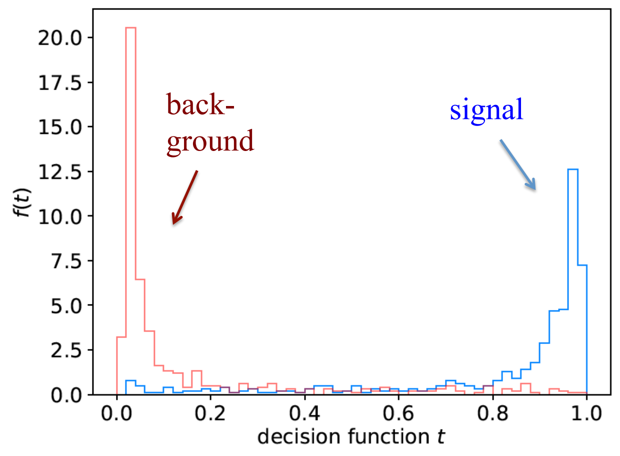
The optimal values of the weights are found by minimizing $E(\boldsymbol{w})$ with respect to the weights (non-trivial algorithms: backpropagation, stochastic gradient descent,...).

The desired result for an event with feature vector $\boldsymbol{x}$ is:

if the event is of type 0, want $t(\boldsymbol{x}) \sim 0$,
if the event is of type 1, want $t(\boldsymbol{x}) \sim 1$.

Distribution of neural net output

Degree of separation between classes now much better than with linear decision function:

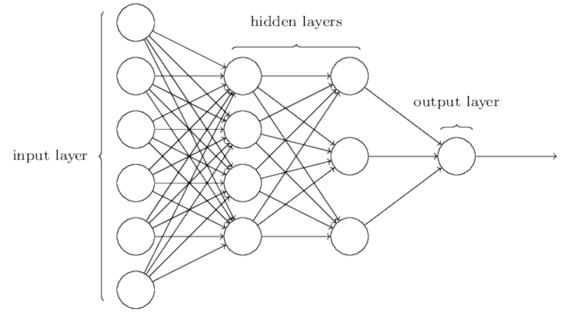


http://neuralnetworksanddeeplearning.com/chap1.html

Deep Neural Networks

The multilayer perceptron can have be generalized to have an arbitrary number of hidden layers, with an arbitrary number of nodes in each (= "network architecture").

A "deep" network has several (or many) hidden layers:

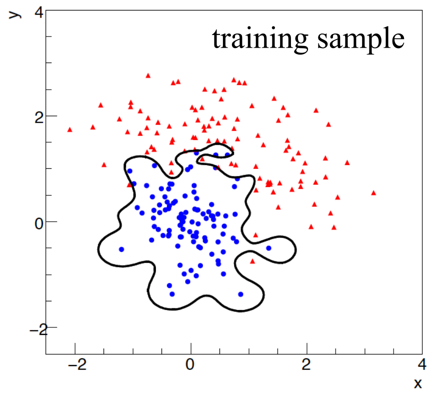


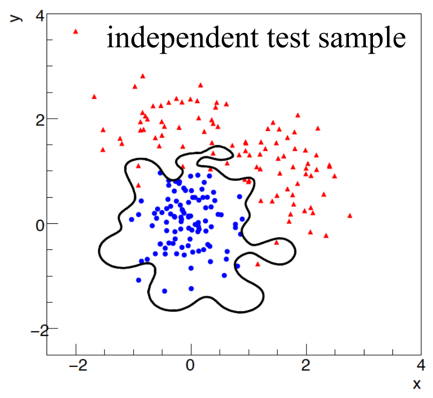
"Deep Learning" is a very recent and active field of research.

Overtraining

Including more parameters in a classifier makes its decision boundary increasingly flexible, e.g., more nodes/layers for a neural network.

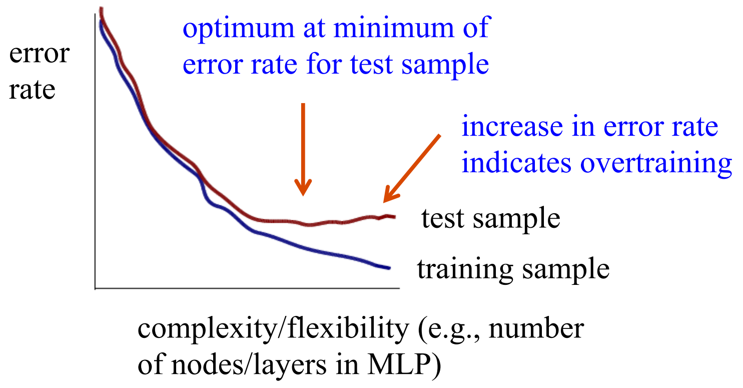
A "flexible" classifier may conform too closely to the training points; the same boundary will not perform well on an independent test data sample (→ "overtraining").





Monitoring overtraining

If we monitor the fraction of misclassified events (or similar, e.g., error function $E(\mathbf{w})$) for test and training samples, it will usually decrease for both as the boundary is made more flexible:



Other types of classifiers

We have seen only two types of classifiers:
- Linear (Fisher discriminant)
- Neural Network

There are many others:
- Support Vector Machine
- Boosted Decision Tree
- *K*-Nearest Neighbour
- ...

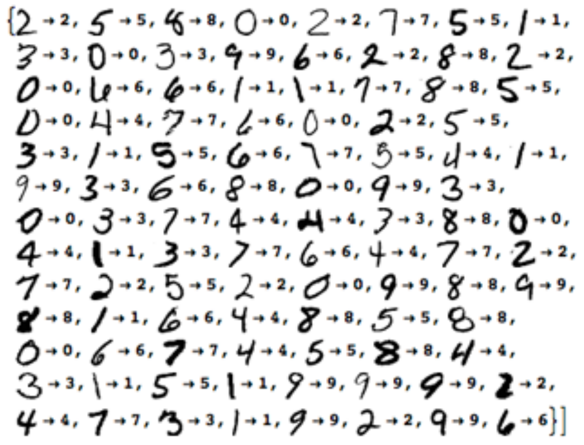
The field is rapidly developing with advances, e.g., that allow one to use feature vectors of very high dimension, such as the pixels of an image.

→ face/handwriting recognition, driverless cars...

https://www.wolfram.com/mathematica/new-in-10/highly-automated-machine-learning/

Machine Learning for handwriting recognition

Initial feature vector = set of pixels of an image



Y. LeCun

Scene parsing/labeling with Convolutional Neural Nets

[Farabet et al. ICML 2012, PAMI 2013]

See *Deep Learning and the Future of AI*, seminar at CERN by Yann LeCun: **`https://indico.cern.ch/event/510372/`**

Software for Machine Learning

We will practice ML with the Python package scikit-learn

`scikit-learn.org` ← software, docs, example code

scikit-learn built on NumPy, SciPy and matplotlib, so you need

```
import scipy as sp
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

and then you import the needed classifier(s), e.g.,

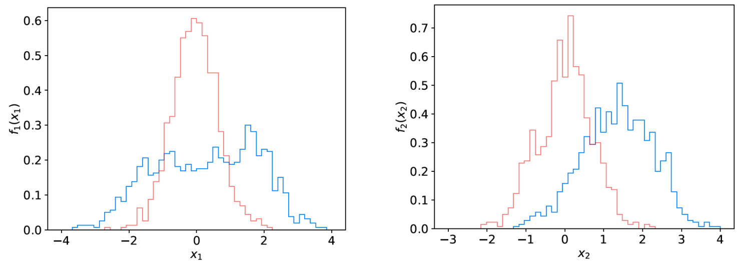
```
from sklearn.neural_network import MLPClassifier
```

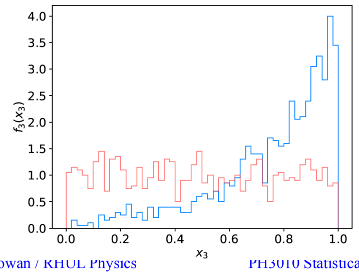
For a list of the various classifiers in scikit-learn see the docs on scikit-learn.org, also a very useful sample program:

`http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html`

Example: the data

We will do an example with data corresponding to events of two types: signal ($y = 1$, blue) and background ($y = 0$, red).





Each event is characterised by 3 quantities: $\boldsymbol{x} = (x_1, x_2, x_3)$.

Components are correlated.

Suppose we have 1000 events each of signal and background.

Reading in the data

scikit-learn wants the data in the form of numpy arrays:

```
#  read the data in from files,
# assign target values 1 for signal, 0 for background
sigData = np.loadtxt('signal.txt')
nSig = sigData.shape[0]
sigTargets = np.ones(nSig)
bkgData = np.loadtxt('background.txt')
nBkg = bkgData.shape[0]
bkgTargets = np.zeros(nBkg)

# concatenate arrays into data X and targets y
# split into two parts:  use one for training, the other for testing
X = np.concatenate((sigData,bkgData),0)[:,0:2]        # for now, only use x1, x2
y = np.concatenate((sigTargets, bkgTargets))

# split data into training and testing samples
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5,
                                                    random_state=1)
```

Create, train, evaluate the classifier

Create an instance of the MLP (multilayer perceptron) class and “train”, i.e., adjust the values of the weights to minimise the loss function.

Here we request 3 hidden layers with 10 nodes each:

```
# create classifier object and train
clf = MLPClassifier(hidden_layer_sizes=(10,10,10), activation='tanh',
                    max_iter=2000, random_state=6)
clf.fit(X_train, y_train)
```

Use test data to see what fraction of events are correctly classified (default takes threshold of 0.5 for decision function)

```
# evaluate its accuracy (= 1 – error rate) using the test data
y_pred = clf.predict(X_test)
print(metrics.accuracy_score(y_test, y_pred))
```

Evaluating the decision function

So now for any point (x_1, x_2) in the feature space, we can evaluate the decision:

```
xt = np.array([0.37, 2.46]).reshape((1L, 2L))      # input is numpy array
t = clf.predict_proba(xt)[0, 1]
```

Or we may have an array of points in $\boldsymbol{x}$-space, so we can get an array of probabilities:

```
t = clf.predict_proba(X_test)[:, 1]         # returns prob to be of type y=1
```

Can get this separately for the signal and background events and make histograms (see sample code).

Note for most other classifiers, the decision function is called decision_function – use this instead of predict_proba.

Resources on Machine Learning

Mini-intro to ML included in PH4515 Statistical Data Analysis

Course in our CS Dept., CS3920/CS4920 (also KCL Maths).

Online courses, e.g., K. Markham, Introduction to machine learning with scikit-learn (also on youtube):

`www.dataschool.io/machine-learning-with-scikit-learn/`

Many online courses, tutorials – worth a look but often very qualitative and/or approach from computer science angle.

More python software (very rapid development):
pandas, seaborn, tensorflow, keras, theano,..

Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, An Introduction to Statistical Learning with Applications in R

`www-bcf.usc.edu/~gareth/ISL/`

`https://www.r-bloggers.com/in-depth-introduction-to-machine-learning-in-15-hours-of-expert-videos/`

Exercises

Run the program simpleClassifier.py and describe the output.
It is set up to use at first only the first two components, x_1 and x_2, so that the results can be displayed as a scatter plot.

Change program to use all three input variables by removing the line X = X[:,0:2] ; also you will have to side-step the code that makes the scatter plot.

Change numbers of hidden layers and nodes – try to find the maximum possible classification accuracy.

For 1 hidden layer e.g., with 10 nodes: hidden_layer_sizes=(10,) .

Note if the number of requested layers/nodes gets too large, it will not be possible to train (find the minimum of the loss function).

For the best architecture that you find, use the test sample to produce histogram of the network output (see sample code).

Exercises (continued)

By looking at the scikit-learn documentation and the sample program plot_classifier_comparison mentioned above, implement a linear classifier (class LinearDiscriminantAnalysis) using all three input variables.

Make a histogram of the classifier output; compare its performance to your best neural network.

Optional: by consulting the documentation and sample program, implement any/all of:

K-Nearest Neighbor Classifier (KNeighborsClassifier).

Support Vector Machine (SVC)

Boosted Decision Tree (AdaBoostClassifier)

Write up the ML exercises as a single section of your project report.

```python
#  simpleFit.py
#  G. Cowan / RHUL Physics / October 2017
#  Simple program to illustrate least-squares fitting with curve_fit

import matplotlib
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import curve_fit

# define fit function
def func(x, *theta):
    theta0, theta1 = theta
    return theta0 + theta1*x

# set data values
x   = np.array([1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0])
y   = np.array([2.7, 3.9, 5.5, 5.8, 6.5, 6.3, 7.7, 8.5, 8.7])
sig = np.array([0.3, 0.5, 0.7, 0.6, 0.4, 0.3, 0.7, 0.8, 0.5])

# set default parameter values and do the fit
p0 = np.array([1.0, 1.0])
thetaHat, cov = curve_fit(func, x, y, p0, sig, absolute_sigma=True)

# Retrieve minimized chi-squared, etc.
numPoints = len(x)
numPar = len(p0)
ndof = numPoints - numPar
chisq = sum(((y - func(x, *thetaHat))/sig)**2)
print "chisq = ", chisq, ",     ndof = ", ndof

# Print fit parameters and covariance matrix
print "\n", "Fitted parameters and standard deviations:"
sigThetaHat = np.sqrt(np.diag(cov))
for i in range(len(thetaHat)):
    print "thetaHat[", i, "] = ", thetaHat[i], "  +-  ", sigThetaHat[i]

print "\n", "i, j, cov[i,j], rho[i,j]:"
for i in range(len(thetaHat)):
    for j in range(len(thetaHat)):
        rho = cov[i][j] / (sigThetaHat[i]*sigThetaHat[j])
        print i, "  ", j, "  ", cov[i][j], "  ", rho

# Set up plot
matplotlib.rcParams.update({'font.size':18})     # set all font sizes
plt.clf()
fig, ax = plt.subplots(1,1)
plt.gcf().subplots_adjust(bottom=0.15)
plt.gcf().subplots_adjust(left=0.15)
plt.errorbar(x, y, yerr=sig, xerr=0, color='black', fmt='o', label='data')
plt.xlabel(r'$x$')
plt.ylabel(r'$y$', labelpad=10)
xMin = 0
xMax = 10
yMin = 0
yMax = 10
plt.xlim(xMin, xMax)
plt.ylim(yMin, yMax)
xPlot = np.linspace(xMin, xMax, 100)        # enough points for a smooth curve
fit = func(xPlot, *thetaHat)
plt.plot(xPlot, fit, 'red', linewidth=2, label='fit result')

# Tweak legend
```

```
handles, labels = ax.get_legend_handles_labels()
handles = [handles[1], handles[0]]
labels = [labels[1], labels[0]]
handles = [handles[0][0], handles[1]]      # turn off error bar for data in legend
plt.legend(handles, labels, loc='lower right', fontsize=14, frameon=False)

# Make and store plot
plt.show()
plt.savefig("simpleFit.pdf", format='pdf')
```

```
#  simpleClassifier.py
#  G. Cowan / RHUL Physics / October 2017
#  Simple program to illustrate classification with scikit-learn

import scipy as sp
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics

#  read the data in from files,
#  assign target values 1 for signal, 0 for background
sigData = np.loadtxt('signal.txt')
nSig = sigData.shape[0]
sigTargets = np.ones(nSig)

bkgData = np.loadtxt('background.txt')
nBkg = bkgData.shape[0]
bkgTargets = np.zeros(nBkg)

# concatenate arrays into data X and targets y
X = np.concatenate((sigData,bkgData),0)
X = X[:,0:2]                    # at first, only use x1 and x2
y = np.concatenate((sigTargets, bkgTargets))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_st
ate=1)

# create classifier object and train
clf = MLPClassifier(hidden_layer_sizes=(5,), activation='tanh',
                    max_iter=2000, random_state=0)
clf.fit(X_train, y_train)

# evaluate its accuracy using the test data
y_pred = clf.predict(X_test)
print 'classification accuracy = ', metrics.accuracy_score(y_test, y_pred)

# make a scatter plot
fig, ax = plt.subplots(1,1)
plt.gcf().subplots_adjust(bottom=0.15)
plt.gcf().subplots_adjust(left=0.15)
ax.set_xlim((-2.5,3.5))
ax.set_ylim((-2,4))
x0,x1 = ax.get_xlim()
y0,y1 = ax.get_ylim()
ax.set_aspect(abs(x1-x0)/abs(y1-y0))       # make square plot
xtick_spacing = 0.5
ytick_spacing = 2.0
ax.yaxis.set_major_locator(ticker.MultipleLocator(xtick_spacing))
ax.yaxis.set_major_locator(ticker.MultipleLocator(ytick_spacing))
plt.scatter(sigData[:,0], sigData[:,1], s=3, color='dodgerblue', marker='o')
plt.scatter(bkgData[:,0], bkgData[:,1], s=3, color='red', marker='o')

# add decision boundary to scatter plot
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
h = .01  # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
# depending on classifier call predict_proba or decision_function
```

```
Z = clf.predict_proba(np.c_[xx.ravel(), yy.ravel()])[:, 1]
Z = Z.reshape(xx.shape)
plt.contour(xx, yy, Z, 1, colors='k')
plt.xlabel(r'$x_{1}$', labelpad=0)
plt.ylabel(r'$x_{2}$', labelpad=15)
plt.savefig("scatterplot.pdf", format='pdf')

# make histogram of decision function
plt.figure()                                      # new window
matplotlib.rcParams.update({'font.size':14})      # set all font sizes
tTest = clf.predict_proba(X_test)[:,1]            # for some classifiers use decisi
on_function
tBkg = tTest[y_test==0]
tSig = tTest[y_test==1]
nBins = 50
tMin = np.floor(np.min(tTest))
tMax = np.ceil(np.max(tTest))
bins = np.linspace(tMin, tMax, nBins+1)
plt.xlabel('decision function $t$', labelpad=3)
plt.ylabel('$f(t)$', labelpad=3)
n, bins, patches = plt.hist(tSig, bins=bins, normed=1, histtype='step', fill=False
, color='dodgerblue')
n, bins, patches = plt.hist(tBkg, bins=bins, normed=1, histtype='step', fill=False
, color='red', alpha=0.5)
plt.savefig("decision_function_hist.pdf", format='pdf')

plt.show()
```