Multivariate Linear Least Squares Minimization

Multivariate Linear Least Squares Minimization

In Linear Least Squares Minimization, we considered the linear functional relation between two measurable variables, x and y:

$$y = a_0 + a_1 x$$

where a_0 and a_1 are unknown conditions to be determined.

On this page we will look at the more generic case, where we solve the problem for an arbitrary number of variables and constants.

Three Variables

Let's start by solving this problem for three measurable variables: y, x_1 and x_2, in the linear functional relation:

$$y = a_0 + a_1 x_1 + a_2 x_2$$

where a_0, a_1 and a_2 are unknown coefficients.

Consider a data set of measured (x_{1i}, x_{2i}, y_i) pairs for $i = 1, 2, 3, \ldots, N$. If we attribute the dispersion of this data from the functional relation to error in the y_i terms, ϵ_i, then we can relate the data points with:

$$\begin{aligned} y_i + \epsilon_i &= a_0 + a_1 x_{1i} + a_2 x_{2i} \\ \therefore \epsilon_i &= a_0 + a_1 x_{1i} + a_2 x_{2i} - y_i \end{aligned}$$

The sum of errors squared is given by:

$$
\begin{aligned}
S &= \sum_{i=1}^{N} \epsilon_i^2 \\
&= \sum_{i=1}^{N} (a_0 + a_1 x_{1i} + a_2 x_{2i} - y_i)
\end{aligned}
$$

We want to minimize S with respect to each of the constants, a_0, a_1 and a_2:

$$
\frac{\partial S}{\partial a_0} = 2 \sum_{i=0}^{n} (a_0 + a_1 x_{1i} + a_2 x_{21} - y_i) = 0
$$

,

$$
\frac{\partial S}{\partial a_1} = 2 \sum_{i=0}^{n} (a_0 + a_1 x_{1i} + a_2 x_{21} - y_i) x_{1i} = 0
$$

and

$$
\frac{\partial S}{\partial a_2} = 2 \sum_{i=0}^{n} (a_0 + a_1 x_{1i} + a_2 x_{21} - y_i) x_{2i} = 0
$$

Re-arranging the above equations and using our statistical notation yields:

$$
a_0 + a_1 \langle x_1 \rangle + a_2 \langle x_2 \rangle = \langle y \rangle
$$

,

$$
a_0 \langle x_1 \rangle + a_1 \langle x_1^2 \rangle + a_2 \langle x_1 x_2 \rangle = \langle x_1 y \rangle
$$

and

$$
a_0 \langle x_2 \rangle + a_1 \langle x_1 x_2 \rangle + a_2 \langle x_2^2 \rangle = \langle x_2 y \rangle
$$

This time algebraic manipulation is a lot more work, instead we shall use a matrix equation (which will serve us better in the more generic case to come). The matrix equation representation is:

$$
\begin{pmatrix}
1 & \langle x_1 \rangle & \langle x_2 \rangle \\
\langle x_1 \rangle & \langle x_1^2 \rangle & \langle x_1 x_2 \rangle \\
\langle x_2 \rangle & \langle x_1 x_2 \rangle & \langle x_2^2 \rangle
\end{pmatrix}
\begin{pmatrix}
a_0 \\ a_1 \\ a_2
\end{pmatrix}
=
\begin{pmatrix}
\langle y \rangle \\ \langle x_1 y \rangle \\ \langle x_2 y \rangle
\end{pmatrix}
$$

This can easily be solved numerically using:

$$
\begin{aligned}
\boldsymbol{X}\boldsymbol{A} &= \boldsymbol{Y} \\
\therefore \boldsymbol{A} &= \boldsymbol{X}^{-1}\boldsymbol{Y}
\end{aligned}
$$

Example - Cepheid Variables

You now have all you need to find the unknown coefficients for the full functional relation of the magnitude (M), period (P) and color ($B-V$) of the Cepheid variables:

$$M = a_0 + a_1 \log P + a_2(B-V)$$

using the same data file as before. (You should find the values $a_0 = -2.15$ mag, $a_1 = -3.12$ mag and $a_2 = 1.49$)

Arbitrarily Many Variables

Consider a linear functional relation between measurable variables x_1, x_2, x_3, ..., x_m and y:

$$\begin{aligned} y &= a_0 + a_1 x_1 + a_2 x_2 + \cdots + a_m x_m \\ &= a_0 + \sum_{j=1}^{m} a_j x_j \end{aligned}$$

where a_0, a_1, ... and a_m are unknown constants.

Suppose we have a data set of measured $(x_{1i}, x_{2i}, \ldots, x_{mi}, y_i)$ values for $i = 1, 2, 3, \ldots, N$. As before, we assume that the dispersion in our data from the functional relation is due to error in the y_i data points only. Therefore we can write the relation between our data points as:

$$y_i + \epsilon_i = a_0 + \sum_{j=1}^{m} a_j x_{ji}$$

The sum of errors squared can thus be written as:

$$S = \sum_{i=1}^{N} \left(a_0 + \left(\sum_{j=1}^{m} a_j x_{ji} \right) - y_i \right)^2$$

We want to find the values of a_0, a_1, ... and a_m which gives us the minimum value of S. Minimizing S with respect to a_0 gives us:

$$\frac{\partial S}{\partial a_0} = 2 \sum_{i=1}^{N} \left(a_0 + \left(\sum_{j=1}^{m} a_j x_{ji} \right) - y_i \right) = 0$$

Distributing the sum over i amongst the terms:

$$\therefore N a_0 + \left(\sum_{j=1}^{m} a_j \sum_{i=1}^{N} x_{ji} \right) - \sum_{i=1}^{N} y_i = 0$$

Dividing by N:

$$\therefore a_0 + \left(\sum_{j=1}^{m} a_j \frac{1}{N} \sum_{i=1}^{N} x_{ji} \right) - \frac{1}{N} \sum_{i=1}^{N} y_i = 0$$

Using our stats notation:

$$\therefore a_0 + \sum_{j=1}^{m} a_j \langle x_j \rangle = \langle y \rangle$$

Now, let's minimize S with respect to one of the a_k for $k = 1, 2, \ldots, m$, following a similar line of algebraic manipulation as above:

$$\frac{\partial S}{\partial a_k} = \sum_{i=1}^{N} 2x_{ki} \left(a_0 + \left(\sum_{j=1}^{m} a_j x_{ji} \right) - y_i \right) = 0$$

$$\therefore a_0 \sum_{i=1}^{N} x_{ki} + \sum_{j=1}^{m} a_j \left(\sum_{i=1}^{N} x_{ki} x_{ji} \right) - \sum_{i=1}^{N} x_{ki} y_i = 0$$

$$\therefore a_0 \langle x_k \rangle + \sum_{j=1}^{m} a_j \langle x_k x_j \rangle = \langle x_k y \rangle$$

Writing the results for a_0 and a_k ($k = 1, \ldots, m$) into a system of equations, expanding the sum over j:

$$\begin{array}{ccccccc}
a_0 & +a_1\langle x_1 \rangle & +a_2\langle x_2 \rangle & +\ldots & +a_m\langle x_m \rangle & = \langle y \rangle \\
a_0\langle x_1 \rangle & +a_1\langle x_1{}^2 \rangle & +a_2\langle x_1 x_2 \rangle & +\cdots & +a_m\langle x_1 x_m \rangle & = \langle x_1 y \rangle \\
a_0\langle x_2 \rangle & +a_1\langle x_2 x_1 \rangle & +a_2\langle x_2{}^2 \rangle & +\cdots & +a_m\langle x_2 x_m \rangle & = \langle x_2 y \rangle \\
a_0\langle x_3 \rangle & +a_1\langle x_3 x_1 \rangle & +a_2\langle x_3 x_2 \rangle & +\cdots & +a_m\langle x_3 x_m \rangle & = \langle x_3 y \rangle \\
\vdots & + \quad \vdots & + \quad \vdots & +\ddots & + \quad \vdots & = \quad \vdots \\
a_0\langle x_m \rangle & +a_1\langle x_m x_1 \rangle & +a_2\langle x_m x_2 \rangle & +\cdots & +a_m\langle x_m{}^2 \rangle & = \langle x_m y \rangle
\end{array}$$

To solve these equations numerically, we can reformulate these equations into a matrix equation:

$$\begin{pmatrix}
1 & \langle x_1 \rangle & \langle x_2 \rangle & \cdots & \langle x_m \rangle \\
\langle x_1 \rangle & \langle x_1{}^2 \rangle & \langle x_1 x_2 \rangle & \cdots & \langle x_1 x_m \rangle \\
\langle x_2 \rangle & \langle x_2 x_1 \rangle & \langle x_2{}^2 \rangle & \cdots & \langle x_2 x_m \rangle \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
\langle x_m \rangle & \langle x_m x_1 \rangle & \langle x_m x_2 \rangle & \cdots & \langle x_m{}^2 \rangle
\end{pmatrix}
\begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}
=
\begin{pmatrix} \langle y \rangle \\ \langle x_1 y \rangle \\ \langle x_2 y \rangle \\ \vdots \\ \langle x_m y \rangle \end{pmatrix}$$

Notice that the left most matrix is symmetric about the diagonal, this can come in handy when computing the matrix elements. As before, this equation can be solved for the a_i by inverting the left most matrix, i.e.

$$\boldsymbol{X}\boldsymbol{A} = \boldsymbol{Y}$$
$$\therefore \boldsymbol{A} = \boldsymbol{X}^{-1}\boldsymbol{Y}$$

Python Implementation

Let's work on a Python implementation of this solution. You may want to try it yourself before reading further. In order to verify our implementation we will use the Cepheid data we've used so far, though in further exercises you will be given data sets containing more variables.

We start by reading in the file. We will read the data into a 2D array. This can be achieved using the standard library as in the **Data Files** section in the **File I/O** chapter, or using `numpy.loadtxt()` (documentation here). We shall use the latter as it is far more convenient:

[43]:
```
import numpy as np

## Reading in the file

data = np.loadtxt('data/cepheid_data.csv', delimiter = ',', skiprows = 1)
```

The `data` array contains all of the data points for $y_i, x_{1i}, x_{2i}, x_{3i}, \ldots, x_{ji}, \ldots, x_{mi}$, where $i = 1, \ldots, N$ corresponds to each row of `data`. Now, we want the data in the format:

$$\texttt{data} = \begin{bmatrix} [y_1, & x_{11}, & x_{21}, & \cdots, & x_{m1} &], \\ [y_2, & x_{12}, & x_{22}, & \cdots, & x_{m2} &], \\ [y_3, & x_{13}, & x_{23}, & \cdots, & x_{m3} &], \\ [\vdots, & \vdots, & \vdots, & \ddots, & \vdots &], \\ [y_N, & x_{1N}, & x_{2N}, & \cdots, & x_{mN} &] \end{bmatrix}$$

as this will make slicing it more clear. In the case of the Cepheid variable data, however, we have our "y" variable in the central column. Therefore we shall swap column 1 and 0 to better align with our desired data structure:

[44]:
```
# Swapping data[:, 0] and data[:, 1]
# Note that this is particular to the data file we are using
# np.copy is necessaary as arrays are not passed as values by default but as␣
↪reference

data[:, 0], data[:, 1] = np.copy(data[:, 1]), np.copy(data[:, 0])
```

To extract the values of a single variable for each measurement, slice columns out of `data`. For example, the y_i are contained in the slice `data[:, 0]`, the x_{1i} are contained in `data[:, 1]`, the x_{2i} are contained in `data[:, 2]`, etc.

Note that for each of the sums along the data sets ($\sum_{i=1}^{N}$), we will be summing along the columns. For example, for the quantity:

$$\langle x_1 \rangle = \frac{1}{N} \sum_{i=1}^{N} x_{1i}$$

[45]:
```
#Using numpy.mean to calculate the expectation value
#Note that x1 = data[:,1]

x1_mean = np.mean(data[:,1])
```

To calculate an expectation value like

$$\langle x_1 x_2 \rangle = \frac{1}{N} \sum_{i=1}^{N} x_{1i} x_{2i}$$

we can use:

[46]:
```
x1_x2_mean = np.mean( data[:,1] * data[:,2] )
```

where we've made use of NumPy array's vectorized operation to multiply each element together before taking the mean of the results.

Constructing the $\boldsymbol{X}$ Matrix Before we continue, let's break down the structure of the matrix:

$$\boldsymbol{X} = \begin{pmatrix} 1 & \langle x_1 \rangle & \langle x_2 \rangle & \langle x_3 \rangle & \cdots & \langle x_m \rangle \\ \langle x_1 \rangle & \langle {x_1}^2 \rangle & \langle x_1 x_2 \rangle & \langle x_1 x_3 \rangle & \cdots & \langle x_1 x_m \rangle \\ \langle x_2 \rangle & \langle x_2 x_1 \rangle & \langle {x_2}^2 \rangle & \langle x_2 x_3 \rangle & \cdots & \langle x_2 x_m \rangle \\ \langle x_3 \rangle & \langle x_3 x_1 \rangle & \langle x_3 x_2 \rangle & \langle {x_3}^2 \rangle & \cdots & \langle x_3 x_m \rangle \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \langle x_m \rangle & \langle x_m x_1 \rangle & \langle x_m x_2 \rangle & \langle x_m x_3 \rangle & \cdots & \langle {x_m}^2 \rangle \end{pmatrix}$$

Let's construct an empty matrix for which we will fill in the entries as we go:

[47]:
```
var_count = data.shape[1]

X = np.matrix(np.ones((var_count, var_count)))
```

Note that we have created an $(m+1) \times (m+1)$ matrix, where $m+1$ is given by the length of axis-1 of `data`.

Now, as we have noted before, $\boldsymbol{X}$ is a symmetric matrix. That is for for row k and column l, $\boldsymbol{X}_{kl} = \boldsymbol{X}_{lk}$. We only need to construct one of the triangles of the matrix, the other is obtained for free.

Let's work with the upper triangle of the matrix. Here there are 3 regions with distinguishable structures

1. The first row
2. The diagonal

3. The remaining triangle

The first element of the matrix is just one. The remainder of the first row is simply the expectation value of each of the x_j:

$$\boldsymbol{X}_{00} = 1$$

and

$$\boldsymbol{X}_{0l} = \langle x_l \rangle \quad \text{where } l = 1, 2, \ldots, m$$

Note that here we are indexing $\boldsymbol{X}$ from 0 to better translate it to code:

[48]:
```
# First row and column
# We leave the first element as is

for l in range(1, var_count):
    X[0, l] = np.mean(data[:, l])

    # Setting the values for the first column
    # remember that X[k, l] = X[l, k]
    X[l, 0] = X[0, l]
```

Now, consider the triangle off of the diagonal. That is the region:

$$\begin{pmatrix} - & - & - & - & \cdots & - \\ - & - & \langle x_1 x_2 \rangle & \langle x_1 x_3 \rangle & \cdots & \langle x_1 x_m \rangle \\ - & - & - & \langle x_2 x_3 \rangle & \cdots & \langle x_2 x_m \rangle \\ - & - & - & - & \cdots & \langle x_3 x_m \rangle \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ - & - & - & - & \cdots & \langle {x_m}^2 \rangle \end{pmatrix}$$

This region exhibits the pattern:

$$\boldsymbol{X}_{kl} = \langle x_k x_l \rangle \quad \text{where } l > k$$

The diagonal has a fairly simple pattern, starting from (row, column) $(1, 1)$:

$$\boldsymbol{X}_{kk} = \langle {x_k}^2 \rangle$$

Note, however, that this is a special case of the rules for constructing region 3. We can therefore combine regions 2 and 3 with the rule:

$$\boldsymbol{X}_{kl} = \langle x_k x_l \rangle \quad \text{where } l \geq k$$

In the code this becomes:

[49]:
```
# Inner matrix

for k in range(1, var_count):
    for l in range(k, var_count):
        X[k, l] = np.mean( data[:, k] * data[:, l] )

        #Setting the value for the lower triangle
        X[l, k] = X[k, l]
```

That covers the $\boldsymbol{X}$ matrix.

Constructing the $\boldsymbol{Y}$ Matrix Now let's construct the matrix:

$$\boldsymbol{Y} = \begin{pmatrix} \langle y \rangle \\ \langle x_1 y \rangle \\ \langle x_2 y \rangle \\ \vdots \\ \langle x_m y \rangle \end{pmatrix}$$

This is fairly straight forward, with

$$\boldsymbol{Y}_{0,0} = \langle y \rangle$$

and

$$\boldsymbol{Y}_{k,0} = \langle x_k y \rangle \quad \text{where } k = 1, \ldots, m$$

[50]:
```
#Creating the Y column matrix:
Y = np.matrix( np.zeros( (var_count, 1) ) )

#First entry
Y[0, 0] = np.mean(data[:,0])

#The remainder of the entries
for k in range(1, var_count):
    Y[k, 0] = np.mean( data[:, k] * data[:, 0] )
```

Finding Matrix $\boldsymbol{A}$ (Or Solving For the a_j) Lastly, to solve for our a_j values, we consider the matrix:

$$\boldsymbol{A} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_m \end{pmatrix}$$

This fits into the matrix equation

$$\boldsymbol{X}\boldsymbol{A} = \boldsymbol{Y}$$

where we've already constructed $\boldsymbol{X}$ and $\boldsymbol{Y}$. All that's left is to solve the equation be inverting $\boldsymbol{X}$:

$$\boldsymbol{A} = \boldsymbol{X}^{-1}\boldsymbol{Y}$$

To achieve this numerically, we simply take the inverse of `X`, `X.I`:

[51]:
```
#Finding A:

A = X.I*Y

print(A)
```

```
[[-2.14515885]
 [-3.11733284]
 [ 1.48566643]]
```

As you can see are results agree with the specific solution for the the case of 3 variables above.

Putting it all together: Let's gather all of the code cells together into a single script. We will also merge the loops together for efficiency:

[24]:
```
import numpy as np

#Reading the data
data = np.loadtxt('data/cepheid_data.csv', delimiter = ',', skiprows = 1)

# Swapping data[:, 0] and data[:, 1]
# Note that this is particular to the data file we are using
# np.copy is necessaary as arrays are not passed as values by default but as␣
 ↪reference
data[:, 0], data[:, 1] = np.copy(data[:, 1]), np.copy(data[:, 0])

#Creating empty X and Y matrices
var_count = data.shape[1]

X = np.matrix(np.ones( (var_count, var_count) ))
Y = np.matrix( np.zeros( (var_count, 1) ) )

#Filling the X and Y matrices

Y[0, 0] = np.mean(data[:,0])

for k in range(1, var_count):
    #First row and column of X
```

```
    X[0, k] = np.mean(data[:, k])
    X[k, 0] = X[0, k]

    #Y
    Y[k, 0] = np.mean( data[:, k] * data[:, 0] )

    #Inner matrix of X
    for l in range(k, var_count):
        X[k, l] = np.mean( data[:, k] * data[:, l] )
        X[l, k] = X[k, l]

#Calculating A

A = X.I*Y

print(A)
```

```
[[-2.14515885]
 [-3.11733284]
 [ 1.48566643]]
```