Linear Least Squares Minimization

Linear Least Squares Minimization

The Problem

We propose a linear functional relation between 2 measurable variables, x and y:

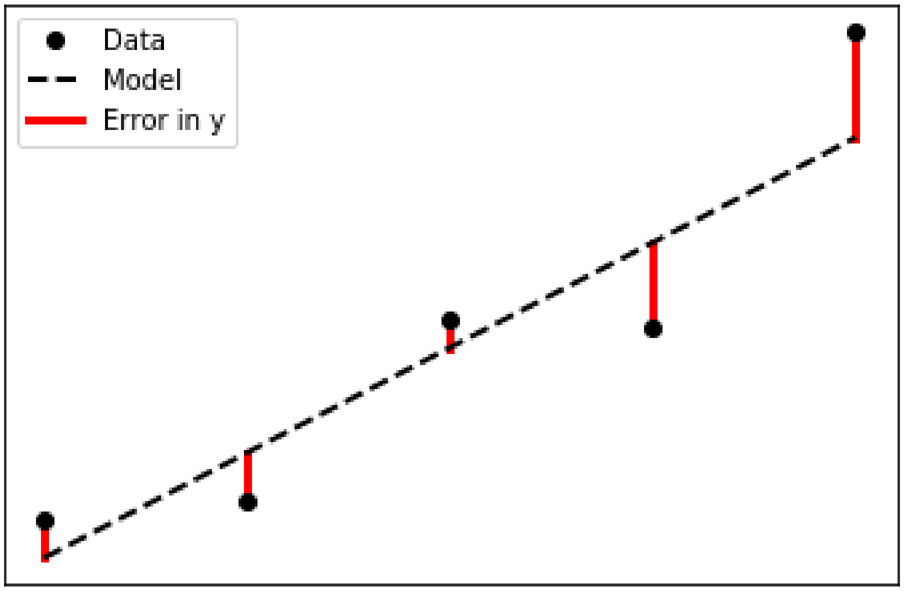
$$y = a_0 + a_1 x$$

where a_0 and a_1 are **unknown** constants. We wish to find these constants.

The Solution

To find these unknown coefficients in practice we measure many x, y pairs (assuming the measurements display some sort of dispersion). We now have a set of measured (x_i, y_i) pairs for $i = 1, 2, 3, \ldots, N$.

If we assume that the x_i are free of error, we can introduce error terms ϵ_i to the y_i data to make up for the dispersion of the data (i.e. that it doesn't follow the linear relation exactly).



With this error term, the relation between our data points can be represented as:

$y_i + \epsilon_i = a_0 + a_1 x_i$

Note that, at this point the error terms we have introduced are unknown to us. They represent the difference between the measured y_i values and the expected values if we plugged x_i into our relation (for which we have yet to determine a_0 and a_1). The error terms can be seen as a means to an end and will soon be done away with.

Now, we need some sort of metric to tell us how much error we have. We can use the sum of the errors squared for this:

$$S = \sum_{i=1}^{N} \epsilon_i^2$$

We use the squares of the error as it is the magnitude of the errors we are concerned about, and with the errors ranging between positive and negative values will end up canceling each other out (these are illustrated as points above and below the lines in the figure above).

We can use the relation between our data points to replace the ϵ_i^2:

$$S = \sum_{i=1}^{N} (a_0 + a_1 x_i - y_i)^2$$

Now, we want our choice of a_0 and a_1 to give us the least amount of error possible, or rather to give us the minimum value of S. To achieve this we minimize S with respect to a_0:

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

$$=$$

$$a_0 + a_1 \langle x \rangle = \langle y \rangle$$

and a_1:

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

$$=$$

$$a_0 \langle x \rangle + a_1 \langle x^2 \rangle = \langle xy \rangle$$

To solve this system of equations we could use a matrix equation and let the computer determine the solution to that numerically, but with only two equations and unknowns, an analytic solution is easy enough to find:

$$
\begin{aligned}
a_1 &= \frac{\langle xy\rangle - \langle x\rangle\langle y\rangle}{\langle x^2\rangle - \langle x\rangle^2} \\
a_0 &= \langle y\rangle - a_1\langle x\rangle
\end{aligned}
$$

Variance of y

If we assume that the y_i data points are distributed around the "true" y values for the given x_i by a Gaussian distribution with constant variance, we can calculate that variance as:

$$
\begin{aligned}
\sigma^2 &= \frac{1}{N}\sum_{i=1}^{N}\epsilon_i^2 \\
&= \frac{1}{N}\sum_{i=1}^{N}(a_0 + a_1 x_i - y_i)^2
\end{aligned}
$$

Worked Example - Cepheid Variables

For this worked example we will use data from Cepheid variables. These are pulsating stars with their luminosity (or magnitude M) related to the period (P) of their pulsations:

$$M = a_0 + a_1 \log P$$

Note that the relation above is can be made more accurate by including the color or temperature of the star, which we shall use later in the chapter.

As this relation is consistent across all specimens, these stars can be used as a standard candle for measuring distances, all that is needed are measurements from stars with known distances from Earth to determine a_0 and a_1.

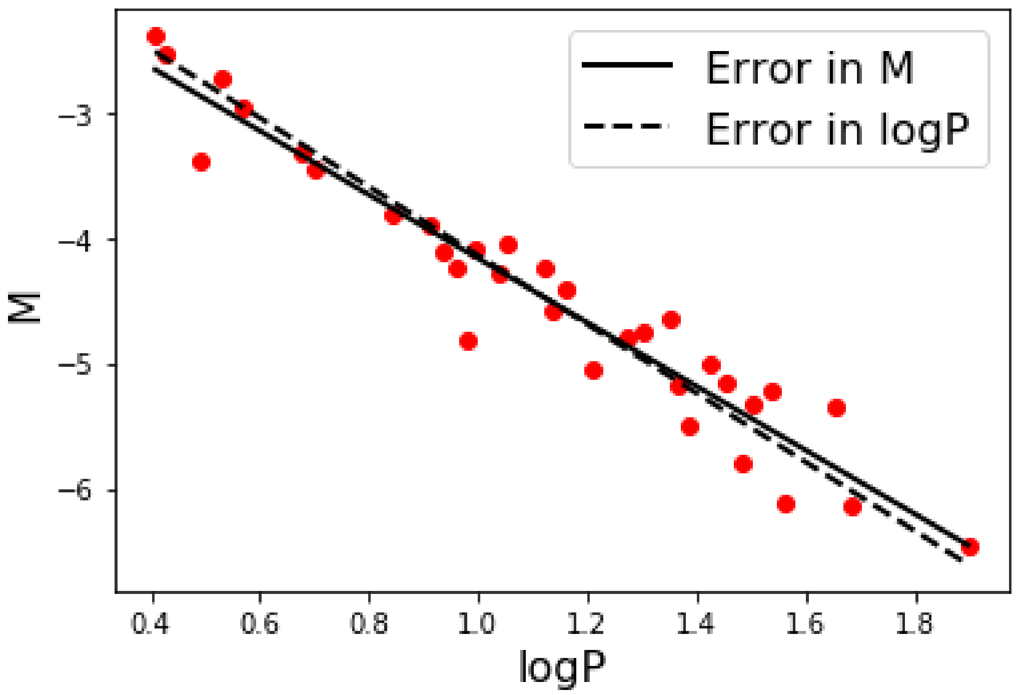
The standard is to measure Cepheids in the Large Magellanic Cloud, whose distance is known. A few of these measurements can be found in the data file 'cepheid_data.csv' provided on Vula (Resources/Exercises/Data/Exercise10/) or on GitHub. The data file contains measurements of:

- $\log P$
- M
- $B - V$ (color, not using yet)

Solution

[13]:
```python
def least_square(x, y):
    mean_x = np.mean(x)
    mean_y = np.mean(y)
    expect_xy = np.mean(x*y)
    expect_xx = np.mean(x*x)
    a1 = (expect_xy - mean_x*mean_y)/(expect_xx - mean_x*mean_x)

    return [mean_y - a1*mean_x, a1]

def get_sigma(a0, a1, x, y):
    return np.sqrt(np.mean((a0 + a1*x - y)**2))
```

[15]:
```
fontsize = 16
linewidth = 2


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

a0 , a1 = least_square(data[:,0], data[:,1]) # error in M
b0, b1 = least_square(data[:,1], data[:,0]) #error in logP

x = np.linspace(data[:,0].min(), data[:,0].max(), 2)

y_M = a0 + a1*x
y_P = -b0/b1 + x/b1

fig_ceph, ax = plt.subplots()

ax.plot(data[:,0], data[:,1], 'ro')
ax.plot(x, y_M, 'k', label = 'Error in M', lw = linewidth)
ax.plot(x, y_P, 'k--', label = 'Error in logP', lw = linewidth)

ax.set_xlabel('logP', fontsize = fontsize)
ax.set_ylabel('M', fontsize = fontsize)

ax.legend(fontsize = fontsize)

plt.show()
```


Error in M
Error in logP
M
logP
−3
−4
−5
−6
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8