Using least squares with a linear system

v1.0, by Iain Cunningham, April 10, 2012

To contact the author, please visit: `http://iainism-blogism.blogspot.co.uk/`

Introduction

Say we have a sequence (the output or dependent variable) that depends upon another variable (the input or independent variable, for example, time) and we want to predict when the output will reach a certain value (e.g. zero). We need to work out what value the input will be when this happens. BUT, we know that the measurement of the output may be corrupted slightly, for example by noise or deficiencies in our sensor. If we assume that the errors in the measurement to the 'true' value of the output are normally distributed then we can use the method of least squares to estimate the equation that defines the system, and thereby solve our problem.

I am not going to describe the maths behind the method of least squares here, I will merely set out how you can get it to a form where it may be easily used. If you want to understand the maths 'under the hood' then Wikipedia links to a few useful resources and if you really, really want to read up then a brilliant book that will tell you everything you ever wanted to know, called 'System Identification', which is now out of print has been made freely available by its authors as a PDF: `http://user.it.uu.se/~ps/ref.html`

Notation and the OLS calculation

Let's establish some notation:

- The output at a given instant, k, is denoted y_k
- The input at the same instant is denoted x_k

If we assume a linear relationship then we have: $y_k = ax_k + b$ where a and b are constants. We can write this in matrix form as follows:

$$y_k = \begin{bmatrix} x_k & 1 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \varphi_k^T \theta$$

where:

$$\varphi_k = \begin{bmatrix} x_k \\ 1 \end{bmatrix} \text{ and } \theta = \begin{bmatrix} a \\ b \end{bmatrix}$$

Using the notation above we can write the OLS operation as follows:

$$\hat{\theta} = (\sum_{k=1}^{N} \varphi_k \varphi_k^T)^{-1} (\sum_{k=1}^{N} \varphi_k y_k)$$

Where the 'hat' on θ indicates that as opposed to 'true' values of a and b this now contains estimated values, $\hat{a}$ and $\hat{b}$.

Note that it is essential that we use more values of x and y than there are parameters that we are trying to find values for: If we are trying to find 2 parameter values (as is the case here, since we have a and b) then we must use 3 or more values of x **and** 3 or more associated values of y. More x and y values will almost always give more accurate parameter estimates than less values will, at a cost of computational burden. If you need more than just a dozen or so x and y values then you should evaluate whether the recursive least squares algorithm ('RLS') is better suited to your needs.

This is all very good, but how do we get it into code?

Rearranging to something we can use

If we look at the calculation we can see it naturally splits into two parts, the first being $(\sum_{k=1}^{N} \varphi_k \varphi_k^T)^{-1}$ and the second being $(\sum_{k=1}^{N} \varphi_k y_k)$.

So to make our life easy we can address these parts separately. Taking the first part and ignoring the inverse for now:

$$\sum_{k=1}^{N} \varphi_k \varphi_k^T = \sum_{k=1}^{N} \begin{bmatrix} x_k \\ 1 \end{bmatrix} \begin{bmatrix} x_k & 1 \end{bmatrix} = \sum_{k=1}^{N} \begin{bmatrix} x_k^2 & x_k \\ x_k & 1 \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{N} x_k^2 & \sum_{k=1}^{N} x_k \\ \sum_{k=1}^{N} x_k & N \end{bmatrix}$$

Where we have 'taken the summations inside' the matrix[1]. Let's simplify the notation a little, let:

$$\begin{bmatrix} \sum_{x^2} & \sum_x \\ \sum_x & N \end{bmatrix} = \begin{bmatrix} \sum_{k=1}^{N} x_k^2 & \sum_{k=1}^{N} x_k \\ \sum_{k=1}^{N} x_k & N \end{bmatrix}$$

So now we need the inverse, and because this matrix is $\mathbb{R}^{2\times 2}$ (a mathematical way of saying "two columns by two rows") we can write out the inverse immediately as:

$$\begin{bmatrix} \sum_{x^2} & \sum_x \\ \sum_x & N \end{bmatrix}^{-1} = \frac{1}{N\sum_{x^2} - (\sum_x)^2} \begin{bmatrix} N & -\sum_x \\ -\sum_x & \sum_{x^2} \end{bmatrix}$$

The second part is also simple to write out, again with a simplification of notation at the end:

$$\sum_{k=1}^{N} \varphi_k y_k = \sum_{k=1}^{N} \begin{bmatrix} x_k \\ 1 \end{bmatrix} y_k = \begin{bmatrix} \sum_{k=1}^{N} x_k y_k \\ \sum_{k=1}^{N} y_k \end{bmatrix} = \begin{bmatrix} \sum_{xy} \\ \sum_y \end{bmatrix}$$

And we can now arrive at the end of our journey by multiplying the two parts together:

$$\hat{\theta} = \frac{1}{N\sum_{x^2} - (\sum_x)^2} \begin{bmatrix} N & -\sum_x \\ -\sum_x & \sum_{x^2} \end{bmatrix} \begin{bmatrix} \sum_{xy} \\ \sum_y \end{bmatrix} = \frac{1}{N\sum_{x^2} - (\sum_x)^2} \begin{bmatrix} N\sum_{xy} - \sum_x \sum_y \\ -\sum_x \sum_{xy} + \sum_{x^2} \sum_y \end{bmatrix}$$

[1] In the case of $\sum_{k=1}^{N} 1$ this is $1_0 + 1_1 + 1_2 + \cdots + 1_N$ and since $1_k = 1$, $\sum_{k=1}^{N} 1 = N \times 1 = N$

Or to put it another way:

$$\hat{a} = \frac{N\sum_{xy} - \sum_x \sum_y}{N\sum_{x^2} - (\sum_x)^2}$$

$$\hat{b} = \frac{\sum_{x^2}\sum_y - \sum_x \sum_{xy}}{N\sum_{x^2} - (\sum_x)^2}$$

The actual coding

From the preceding equations we can see that in the main the least squares method requires two main operations, firstly, a summation of a sequence of numbers, secondly the summation of a sequence of products. The first case covers $\sum_x$, $\sum_y$ and the second case covers $\sum_{xy}$. We also need a 'square' operation. Assuming that we don't overload then this suggests three functions, sumSeq, sumProdSeq, getSquare, the first of which takes a single array as a parameter and returns the sum of its elements, the second of which takes two arrays as parameters and returns the sum of the result of multiplying their corresponding elements together, and the third of which takes a single number as a parameter and returns the square of its value.

There is now a design decision to be made: $\sum_{x^2}$ may be acheived in two ways, firstly as a sumSeq of an array of values already squared with getSquare, secondly as a sumProdSeq of the same two arrays. For the purposes of this document we'll take the second approach.

The first thing we need to do is set N, once more, if this is large then the computational load you create may mean that use of the RLS algorithm is better for you.

The following pseudo-code sets out the algorithm:

```
declare N
define N=<your N>
declare loopCount  // Must be big enough to hold N
define loopCount=0
declare xArray[N]  // Will hold x_1, x_2, ... x_n
declare yArray[N]  // Will hold y_1, y_2, ... y_n
declare aHat
declare bHat
declare den
define aHat=1   // Or your preferred default value
define bHat=1   // Or your preferred default value
define den=1    // Or your preferred default value

loop
// Your application code

// Get the next x and y values into xArray and yArray
```

```
  getNextXY(xArray, yArray)
  // The 'actual' least squares algorithm follows
  if loopCount >= N do
     sumX = sumSeq(xArray)
     sumX2 = sumProdSeq(xArray, xArray)
     sumXY = sumProdSeq(xArray, yArray)
     sumY = sumSeq(yArray)
     den = N * sumX2 - getSquare(sumX)

//INSERT APPLICATION SPECIFIC CODE HERE
//TO HANDLE den BEING zero OR near-zero

     aHat = (N * sumXY - sumX * sumY)/den
     bHat = (sumX2 * sumY - sumX * sumXY)/den
   else
     loopCount = loopCount+1
   endif

// More of your application code
endloop
```

As suggested by the listing, it is essential to check 'den' is not zero (or any other value that could lead to overflow in your target environment) since otherwise the divisions performed in the aHat and bHat calculations could cause extremely undesirable behaviour — depending upon how robust your runtime environment is to a 'division by zero'! Even if your runtime is designed to be robust to 'division by zero' situations, it may well be that 'den' being zero is indicative of some otherwise undetected problem(s) in your system[2]: DO NOT RELY ON YOUR RUNTIME TO SAVE YOU!

Depending on your application you may also want to apply bounds checking to the values of aHat and bHat or even 'force' them to 'safe' (or at least detectable) values in the event that they are out of range or the estimation process has broken down.

[2]For example, if x is time then 'den' being zero could be indicative of a failure in your timekeeping system that has lead to xArray being populated with zeros, while the estimator can be made robust to this eventuality, this kind of failure could have drastic effects elsewhere