Applying the Least-Squares Method

28th Jan 2004

Summary of the lecture

1. We proved that the inverse of a linear system, when it exists, is also linear

2. Overview of system identification

3. We saw how to apply the least-squares method to system identification when we're modelling the system as MA.

Inverse of a linear system

Any linear system can be represented by a matrix (or by infinite matrix-shaped things). That's because a linear system is completely specified by its impulse responses at each time; and the columns of the matrix are these impulse responses.

So, a linear system P is the matrix multiplication

$$y = Px$$

where x is the input and y the output.

When does the function P^{-1} exist ? For any function, the inverse exists when it is one-one and onto. In the language of matrices and spaces and stuff, what does this mean ?

One-one-ness :

A necessary condition is that the output should have atleast as many dimensions as the input. That's because it is impossible to **linearly** map (i.e. "project") a higher dimensional space into a smaller dimensional one without becoming a many-to-one function.

It turns out that this is also a sufficient condition. There's a theorem that if a linear system is one-one then it will have same output dimension as input dimension. You can't become many-to-one and stay linear without losing a dimension.

Onto-ness :

A function is onto when its range equals its codomain.

The dimension of the output space is the rank of the matrix P^T.

To see why, recall how we find the rank of a matrix: by doing row transformations and trying to make some row zero. The number of non-zero rows remaining is the rank.

This means that we are trying to express some row as the linear combination of other rows. When this matrix is P^T, the rows are the bases of P. So we're trying to express *one basis vector as a linear combination of others.* Such a vector doesn't need to be lying around in the basis, because the basis would span the same space without it. So we remove this extra vector. Removing all such extra vectors is exactly what we do while finding the rank. That's why the "true" dimension of the output vector space is the rank of P^T.

For the matrix to be an onto function, it must be a *full rank* matrix, meaning that none of the colums should be extra vectors. This corresponds to the range=codomain condition.

Now we'll prove that when the inverse does exist, it is linear. That is, we'll show that P^{-1} is a linear transformation.

Applying the function P^{-1} to a vector gets back the linear combinants[1]

First we show that $P^{-1}(x+y) = P^{-1}x + P^{-1}y$:

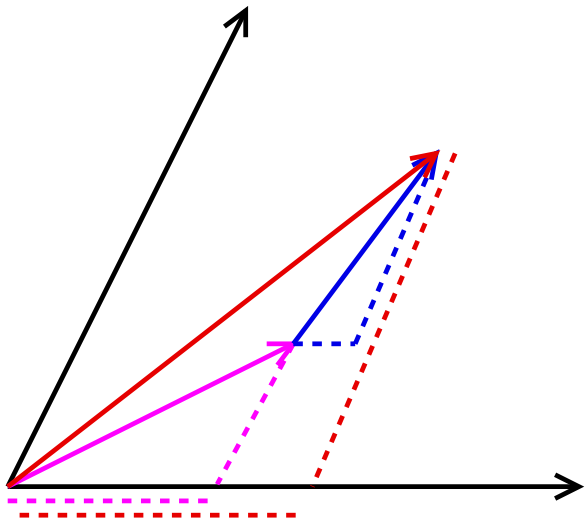

Figure 1: Linearity : Superposition

The blue solid line is x; the pink solid line is y; The red solid line is $(x+y)$. The dotted lines are what P^{-1} gives us. See how the blue and pink dotted lines add up to the red dotted line.

Then, we prove $P^{-1}(ax) = aP^{-1}x$:

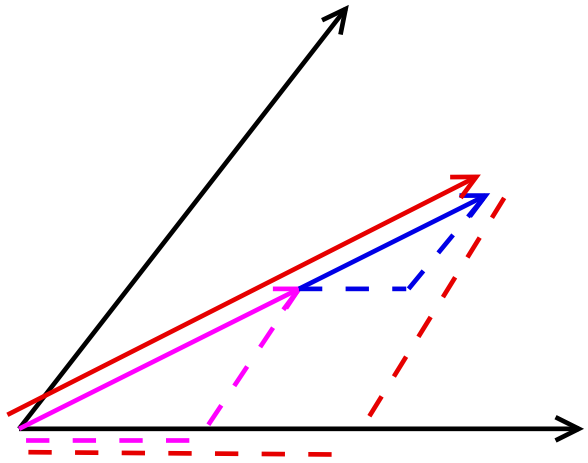

Figure 2: Linearity : Scaling

The pink solid line is x; the blue solid line is ax for some a; The red solid line is $(a+1)x$. Again, the blue and pink dotted lines add up to the red dotted line, i.e. the components of $(a+1)x$ are $(a+1)$ times the components of x.

[1] In 12th we called these the *scalar components* of the vector. The scalar components multiplied by the corresponding basis form the *vector components*. We'll call both these things "linear combinants".

Algebraically, if $Px = y$:

Superposition:

$$P(x_1 + x_2) = Px_1 + Px_2 = y_1 + y_2$$

Therefore,

$$P^{-1}(y_1 + y_2) = P^{-1}(P(x_1 + x_2)) = x_1 + x_2 = P^{-1}y_1 + P^{-1}y_2$$

Scaling:

$$P(ax) = aPx = ay$$

Therefore

$$P^{-1}ay = P(P^{-1}(ax)) = ax = aP^{-1}y$$

Overview of System Identification

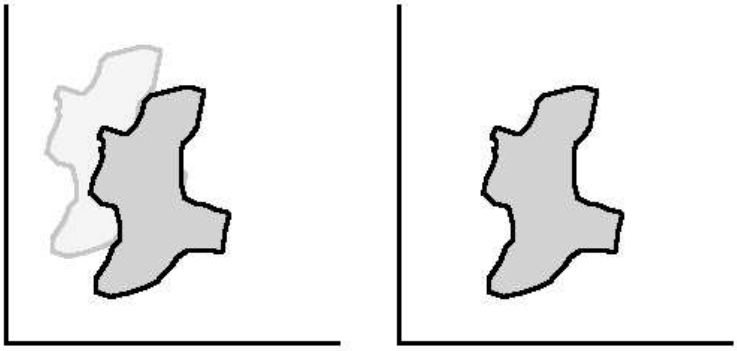

Figure 3: Output and Input

If you have a telescope which gives you an image like the one the left, and you know that the object it's looking at looks like the one on the right.

Then you have to find the convolution kernel which turns the image on the right into the one on the left. This kernel will be something like:

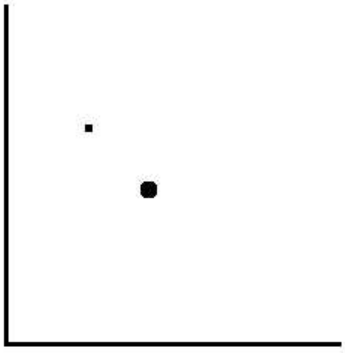

Figure 4: The filter kernel

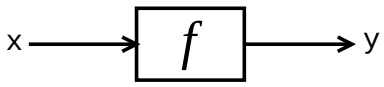

Figure 5: A system

The system identification problem is: to find the impulse response of the LSI system f given an input vector x and the resulting output vector y.

We classify this problem into two kinds:

1. Systems where we cannot control x, only measure what is already there.

2. Systems where we can give any x we want.

In the second type, what's a good x to give ? Well, the best x is an impulse because that immediately gives you the impulse response. But you can't always give an impulse, because it may be impossible in that system. For example, if x is a sound wave, you can't physically make an exact sound impulse.

Basically, any input x which has a wide range of frequencies in it (i.e. a broadband signal) is good. A square wave, a ramp, a frequency sweep, are all good inputs to give. A sine wave is not, because it has just one frequency. If you give a sine wave to the system, you'll get the response to just one frequency. It's better if you give many sine waves with different frquencies. This is called a frequency sweep.

When you cannot control x, you can measure the input being given and try to work with that. But if it is narrow band, then there's nothing you can do about it.

Any causal non-zero signal is broadband.

Applying the Least Squares Method to MA system identification

Suppose we model the system as a moving average system with q coefficients. Then, this is how we can apply the least squares method to find those coefficients given x and y:

The least squares method gives us f when we give it the equation

$$Af = y$$

In this equation, we'll make A out of the input vector x; f is the vector of dimension q that we want to find; and y is the output vector y. For example, if $q = 4$:

$$\begin{pmatrix} x_1 & 0 & 0 & 0 \\ x_2 & x_1 & 0 & 0 \\ x_3 & x_2 & x_1 & 0 \\ x_4 & x_3 & x_2 & x_1 \\ x_5 & x_4 & x_3 & x_2 \\ x_6 & x_5 & x_4 & x_3 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ \vdots \end{pmatrix}$$

We've shown x causal in this equation, but it need not be. If it is non-causal then those zeroes in A will be replaced by x_{-1},x_{-2} etc.

When x is causal, no column in A_x can be expressed as a linear combination of the others. This means that the basis has no extra vectors.

What should the order (q) of the filter be ?

Sometimes you may be able to decide the order from other characteristics of the system. For example if you're modelling a system which disperses its input a little, and you know the time for which the dispersion can occur, then you can decide the order.

Another way to get the order is to iterate over it, each time using the least-squares method to find f, and plot the error $\|y - A_x f\|$ against q. The graph will look something like the blue line in :

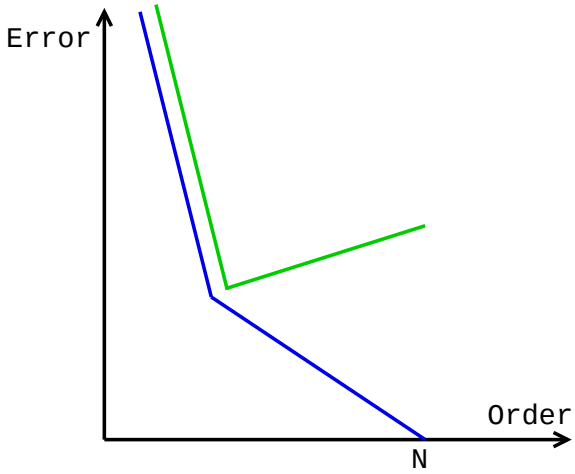

Figure 6: Deciding q

The error becomes zero when there are as many coefficients as the dimension of x, in which case least-squares is actually Gauss Elimination.

The corner in the middle of the graph is the order you should use. Increasing the order beyond this point is just putting in extra coefficients to make the filter better for this particular input.

Instead of plotting the error for the same input and output as was used in the least-squares, suppose we use a different pair of x and y to find $\|y - A_x f\|$. Then if you increase the order too much, the error increases because you're tailoring the filter to the input that was given to least-squares, not the input which was given to $\|y - A_x f\|$. So in this method the order we should use is the point where the error is minimum.

Using MA modelling for modelling an AR system:

You can use MA modelling to model an AR system, using a neat little trick:

The inverse of an AR system is an MA system. So, to model a system as AR, interchange the roles of y and x, give them to least-squares. An MA system will come out. Invert it. That's the required AR system.