ELEN90052 Advanced Signal Processing

Additional reading for SP2-41

1 Introduction

These notes are intended to cover the basics of recursive algorithms for parameter estimation. The intention is that these notes will provide most of the background needed to complete the lab SP2-42.

Be warned that these notes don't contain a lot of detail. For plenty of detail see the lecture notes [1].

We are going to start by introducing a fairly general viewpoint of parameter estimation called a 'prediction error' approach. First we'll look at how it fits into the bigger picture.

Next we'll give a general framework for parameter estimation from this viewpoint Finally we'll look at a number of specific situations and commonly used algorithms for parameter estimation in those situations.

2 Parameter estimation

In a typical situation, we have a real physical system that we want to modify/control/improve by using mathematical techniques. To do this we need a system model — a mathematical description of the system that does a good job of predicting what the system output will be for a given input signal. We then apply our mathematical techniques to the system model and hope[1] that when we convert back to the real world things work as we expect.

There is a general tradeoff between choosing a system model that closely approximates the true system behaviour and choosing a system model that is simple and so is easy to 'engineer' (i.e. analyse/control/improve).

The usual approach is to start by choosing a *model class* for the system model. By this we mean a collection of system models that have a very similar mathematical structure but have different *parameters*.

For example the class of discrete-time, linear, time-invariant state-space models are those of the form

$$\begin{aligned} x(n+1) &= Ax(n) + Bu(n) \\ y(n) &= Cx(n) + Du(n) \end{aligned}$$

where A, B, C, D are fixed matrices of the appropriate dimensions.

Given a model class, we then want to choose a particular model in the model class that best describes the real system we are interested in. This involves choosing values for each of the parameters in the model class.

In the example above, choosing matrices A, B, C, D specifies a particular system model.

From this viewpoint there are two stages to modelling a system:

1. Choose a model class.

2. Choose a model from the model class (i.e. choose parameters for the model)

[1] Actually, if we do careful enough analysis we may be able to do something a bit more precise than hoping.

The choice of model class generally involves balancing choosing a simple model class that is easier to 'engineer' (for example LTI system models) and choosing a model class that has enough complexity to model the system well. Another consideration is choosing a model class for which parameter estimation is not too difficult. An example we will see a number of times in these notes is where the model 'output' depends linearly on the model parameters.

There are a range of different techniques for choosing an appropriate model from the model class (parameter estimation). Essentially they all involve looking at real input and output data and trying to choose parameters so that the resulting model (approximately) reproduces the real data.

3 Model classes and predictors

3.1 Model Classes

One standard model class for a SISO (single-input single-output) system is:

$$y(k) = -a_1 y(k-1) - \cdots - a_{n_a} y(k-n_a) + b_1 u(k-1) + \cdots + b_{n_b} u(k-n_b) + e(k)$$

where $e(k)$ is a sequence of i.i.d. random variables with mean 0 and variance σ^2. Here the parameters are $a_1, \ldots, a_{n_a}, b_1, \ldots, b_{n_b}$. For brevity we typically lump all the parameters together into a vector

$$\theta = \begin{bmatrix} -a_1 & \cdots & -a_{n_a} & b_1 & \cdots & b_{n_b} \end{bmatrix}^T.$$

For many situations this model class is far too large (i.e. there are too many parameters to estimate). So we then should choose a smaller model class contained inside this one (a *subclass*).

For example, consider an echo system that describes the signal a microphone in a room detects when an audio signal is played through the loudspeakers in the room. Imagine we have access to the signal going in to the loudspeaker $u(k)$, and the signal measured by the microphone is $y(k)$. Suppose the sound can take two main paths from loudspeaker (source) to the microphone — a direct path, or an indirect path. It takes the sound 1 second to travel along the direct path and 5 seconds to travel along the indirect path. If we have a sampling interval of 0.5 seconds we might choose our model class to be

$$y(k) = b_1 u(k-2) + b_2 u(k-10) + e(k).$$

Here we have used our understanding of the real system we are trying to model to set superfluous parameters to zero.

3.2 Predictors

Remember that the idea of building a system model is to closely match the output of the real system for given inputs. Given a model class and input and output data from the real system up to time $k-1$ a *predictor* essentially tells us what the output at time k of each of the models in the model class would have been given the data up to time $k-1$.

More precisely, suppose we know the inputs $u(n)$ and outputs $y(n)$ for $n = 0, 1, \ldots, k-1$. Suppose the vector of parameters in the system model is θ.

In the model class given above, our models are not deterministic, so we can't know for sure what the system output will be given these parameters and this past data. Specifically,

if we only have data up to time $k-1$ we don't know much about the noise process $e(k)$. Since it is zero mean, the best we can do is to use its mean value.

So the predictor at time k of the model with parameters θ given data up to time $k-1$ is

$$\hat{y}(k|k-1;\theta) \quad = \quad b_1 u(k-1) + \cdots + b_{n_b} u(k-n_b) - a_1 y(k-1) - \cdots - a_{n_a} y(k-n_a)$$

Note that this predictor has a particularly nice form because it is a linear function of the parameters. If we define, for each k,

$$\phi(k)^T = \begin{bmatrix} u(k-1) & \cdots & u(k-n_b) & -y(k-1) & \cdots & -y(k-n_a) \end{bmatrix}$$

then we can write

$$\hat{y}(k|k-1;\theta) = \phi(k)^T \theta.$$

You might be wondering what we do when $k - n_a < 0$ or $k - n_b < 0$.

One way to deal with this 'initial' situation is by actually defining a different model class for each $k = 0, 1, \ldots, \max\{n_a, n_b\}$. For example, if $0 \leq k \leq \min\{n_a, n_b\}$ we might choose our model class to be models of the form

$$y(k) = -a_1 y(k-1) - \cdots - a_k y(0) + b_1 u(k-1) + \cdots + b_k u(0) + e(k).$$

For the lab, this is probably the best way to deal with the problem.

Another approach would be to just start estimating parameters after time $k = \max\{n_a, n_b\}$. That is, after we have enough data that our full model class makes sense.

From this point on, in these notes, we will ignore this annoying technicality so that the ideas are as clear as possible. You will need to deal with it in any implementation, though.

4 Parameter estimation by prediction error methods

The strategy is this. Suppose we are given input and output data $\{u(0), u(1), \ldots, u(N)\}$ and $\{y(0), y(1), \ldots, y(N)\}$ from our real system up to time N. Let us denote all this data by $\mathcal{Z}_N$ for short.

For a given parameter vector θ we want to know how well the predictors $\hat{y}(k|k-1;\theta)$ have predicted the true outputs $y(k)$ for all the data we have seen so far (i.e. $k = 0, 1, \ldots, N$). To do so we define a *cost function* $C(\mathcal{Z}_N, \theta)$ that depends on the data we have seen so far and the parameter vector. If $C(\mathcal{Z}_N, \theta)$ is small then the system model with parameters θ describes the data $\mathcal{Z}_N$ well. If $C(\mathcal{Z}_N, \theta)$ is large then the system model with parameters θ describes the data $\mathcal{Z}_N$ badly.

Then at time N the 'best' set of parameters (and hence the 'best' system model) is the choice of parameters that minimizes the cost function. That is let

$$\hat{\theta}(N) = \arg\min_{\theta} C(\mathcal{Z}_N, \theta). \tag{1}$$

When it comes to implementing this there are two major choices to make:

1. What cost function C will we use?
2. What method will we use to solve (either approximately or exactly) the optimization problem (1)?

Below we will look at a couple of common choices of cost functions and optimization methods and the types of system models and situations for which they are well suited.

5 A least squares approach

Suppose we are in a situation like we described earlier where the predictor $\hat{y}(k|k-1;\theta)$ is a *linear* function of θ. That is, for each k there is some column vector $\phi(k)$ such that $\hat{y}(k|k-1;\theta) = \phi(k)^T\theta$.

Consider the cost function

$$C(\mathcal{Z}_N, \theta) = \sum_{k=0}^{N} (y(k) - \hat{y}(k|k-1;\theta))^2 = \sum_{k=0}^{N} (y(k) - \phi(k)^T\theta)^2.$$

This is a sensible choice because it means that if the predictors $\hat{y}(k|k-1;\theta)$ all do a good job for some set of parameter values θ then the cost will be low, just as we want.

In this case the optimization problem becomes

$$\hat{\theta}(N) = \arg\min_{\theta} \sum_{k=0}^{N} (y(k) - \hat{y}(k|k-1;\theta))^2 = \arg\min_{\theta} \sum_{k=0}^{N} (y(k) - \phi(k)^T\theta)^2 \tag{2}$$

which can be solved exactly by least squares.

5.1 Recursive least squares

In our general framework, and in the above situation, the cost function depends on N. Thus whenever we get new data (i.e. every sampling interval in a real-time system) we have a new cost function and a new optimization problem to solve. In the case of least squares, solving the problem 'from scratch' each time would be very costly. Fortunately there is a clever way to be able to solve (2) efficiently by updating the solution from the previous time-step. This is the recursive least squares algorithm[2].

$$\hat{\theta}(N) = \hat{\theta}(N-1) + P(N)\phi(N)(y(N) - \phi(N)^T\hat{\theta}(N-1)) \tag{3}$$

$$P(N) = P(N-1) - \frac{P(N-1)\phi(N)\phi(N)^T P(N-1)}{1 + \phi(N)^T P(N-1)\phi(N)} \tag{4}$$

We typically initialize $\hat{\theta}(0)$ to our best estimate of the parameter values given no data (this is often just zero). We typically take $P(0)$ to be some multiple of the identity matrix.

6 Steepest descent methods

The least squares situation, where the predictor is a linear function of the parameters and the cost function is of a very simple type, is a fairly special (albeit very widely used) case. What can we do in more general situations?

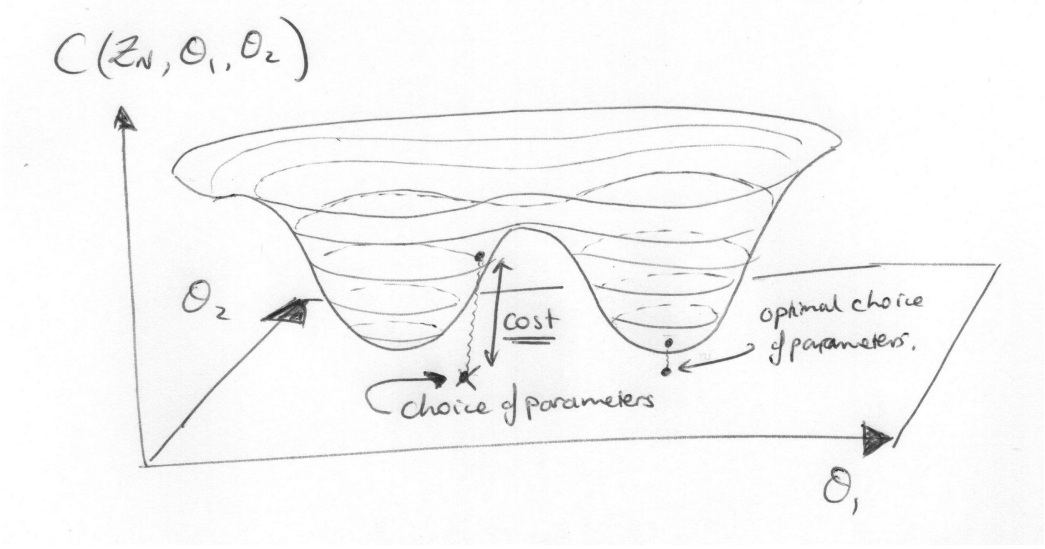
If the cost function C is differentiable, we can approximately solve the optimization problem

$$\hat{\theta}(N) = \arg\min_{\theta} C(\mathcal{Z}_N, \theta)$$

by a steepest descent method.

The picture to have in mind here is the situation where there are two parameters. Then we can think of $C(\mathcal{Z}_N, \theta_1, \theta_2)$ as the height of a surface above the θ_1, θ_2 plane.

[2]These equations look a lot like the equations for a Kalman filter. This is no accident, but it will take us a bit far afield to explore the connection here.



Our aim, then, is to find the lowest point on the surface. Generally this is very hard to do. A reasonable approximate method is to iteratively search for a local minimum by stepping in the direction of 'steepest descent' at each iteration. You might recall from first year calculus that if we start at the parameter value $\hat{\theta}$ then the direction of steepest descent of the cost function C from this starting point is exactly

$$-\nabla C|_{\theta=\hat{\theta}} = -\left[\frac{\partial C}{\partial \theta_1} \quad \cdots \quad \frac{\partial C}{\partial \theta_m}\right]^T\Bigg|_{\theta=\hat{\theta}}.$$

So for a given N, a fairly general, but only approximate procedure for finding $\hat{\theta}(N)$ would be:

1. Start with $\theta_0 = \hat{\theta}(N-1)$ (the previous parameter estimate).
2. Repeatedly take a step of length μ in the direction of steepest descent of C. That is
$$\theta_i = \theta_{i-1} - \mu \nabla C|_{\theta_{i-1}} \tag{5}$$
3. Take $\hat{\theta}(N)$ to be the last θ_i value obtained.

As we have mentioned before, in an online application we need to compute $\hat{\theta}(N)$ in a single sampling interval because when new data arrives, the cost function changes and we have a new optimization problem to solve. Thus in practice we may perform as few as one single iteration of step 2 (equation (5)) above.

So, in general, the critical design choices to make here are:

- How big should the step size (μ) be?
- How many times should we iterate (5)?

6.1 Least Mean Squares

A widely used special case of the steepest descent method is least mean squares. In this case we assume, once again, that the predictor is a linear function of the parameters. That is, for each k there is some column vector $\phi(k)$ such that $\hat{y}(k|k-1;\theta) = \phi(k)^T\theta$.

We choose the cost function simply to be

$$C(\mathcal{Z}_N, \theta) = (y(N) - \hat{y}(N|N-1;\theta))^2 = (y(N) - \phi(N)^T\theta)^2.$$

Notice that this cost function *only* looks at how well the model with parameters θ predicts the most recent output of the true system. It doesn't try to match up *all* the previous inputs and outputs (as was the case with our cost function for least squares).

One reason for choosing a quadratic cost function such as this one is that it only has one local minimum. Thus even though a steepest-descent type algorithm is only a 'local' optimization algorithm, it can't get lured into highly suboptimal local minima because there aren't any!

With this choice of cost function

$$-\nabla C|_{\theta=\hat{\theta}} = 2\phi(k)(y(k) - \phi(k)^T\hat{\theta}).$$

Finally, so that this algorithm is very fast, we typically only perform one steepest descent step for each new data point that arrives. Thus the least mean squares algorithm is:

$$\hat{\theta}(N) = \hat{\theta}(N-1) - \mu\nabla C|_{\hat{\theta}(N-1)} = \hat{\theta}(N-1) + 2\mu\phi(N)(y(N) - \phi(N)^T\hat{\theta}(N-1))$$

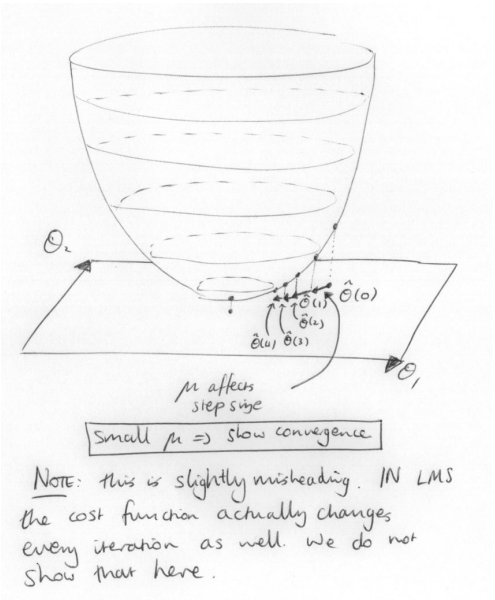
To initialize this we need to choose $\hat{\theta}(0)$. As before this should just be our best estimate of the parameters before any data arrives. Typically we just set $\hat{\theta}(0) = 0$.

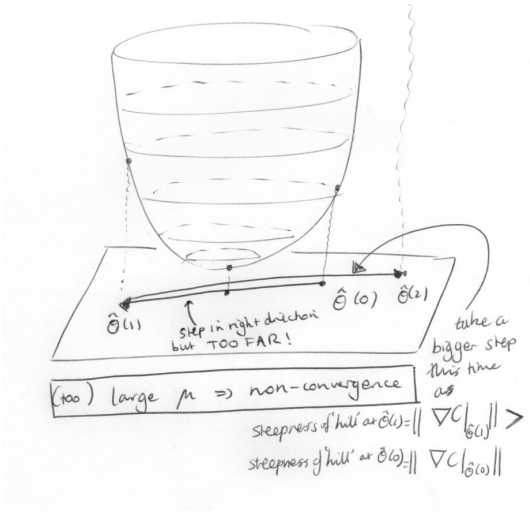
6.2 Choosing the step size

The choice of LMS step size has a (sometimes dramatic) effect on the convergence of the algorithm. If the step size is too small, the algorithm takes a long time to converge. If the step size is to large the algorithm can fail to converge at all.

Notice that the distance that we step in each iteration depends on μ *and* on $\|\nabla C\|$ (that is the 'steepness' of the 'hill' at our starting point).

The step-size trade-off is illustrated below.





The main issue is that if we take small steps it takes a long time to get to the minimum. Yet if we take really large steps we might step right past the minimum and a long way up the other side of the hill. Then because the hill is very steep when we are very far up, the next we take will be even larger, so we will step even further up the other side of the hill, causing things to diverge.

In addition, the data we are feeding into the LMS algorithm is typically noisy. This also has the potential to cause trouble with algorithm like this. Generally speaking, the smaller the step size is, the less the parameter estimates will be affected by noise in the data. This is because a large step size corresponds to having great trust in the accuracy of the most recent data (on which the step size and direction are based). A small step size, on the other hand, suggests that the algorithm trusts the existing estimate that it is stepping from more than it trusts the new data that tells it where to step at each iteration.

More quantitatively, (see [1] for details) as a rough guide it is a good idea to choose

$$0 < \mu \ll \frac{1}{\mathbb{E}\left[\|\phi(k)\|^2\right]}.$$

In practice we don't know what $\mathbb{E}\left[\|\phi(k)\|^2\right]$ is, so it needs to be estimated.

7 Time varying parameters

In practical applications, such as the echo-cancellation application in the lab, we may need to allow time-varying parameters in our system model. If we assume that the system parameters vary slowly compared to the size of the sampling interval, we can deal with this situation by just making some modifications to the sort of cost functions we have already seen.

Before, in the framework of section 4, we said that the cost function $C(\mathcal{Z}_N, \theta)$ should be small if *all* of the past data is well predicted by the system model with parameters θ, and large otherwise.

This intuition is no good if the system parameters change with time. In this new situation we want the cost function to be small if all of the *recent* data is well predicted by the system model with parameters θ, and large otherwise. This is because only recent system data will be relevant to our current estimation of the system parameters.

An example of such a cost function is the function we used for LMS in the previous section.

$$C(\mathcal{Z}_N, \theta) = (y(N) - \hat{y}(N|N-1;\theta))^2.$$

Recall that this function *only* cares whether the most recent data is well predicted by the system model with parameters θ. Thus we can see that the LMS algorithm, as stated in the last section, is already suitable for the situation where the system parameters change with time. In this case the choice of μ affects how quickly the parameters estimates will converge to the time-varying parameters.

Another example of a suitable cost function is the following modification of the cost function we used for least squares based techniques of section 5.

$$C(\mathcal{Z}_N, \theta) = \sum_{k=0}^{N} \lambda^{N-k} (y(k) - \hat{y}(k|k-1;\theta))^2.$$

Here we have introduced a parameter $0 < \lambda \leq 1$ called the 'forgetting factor'. This factor weights the terms in the cost function so that when optimizing it is much more important for us to get 'recent' predictions correct than very old ones.

Just like the choice of μ for LMS, the choosing λ involves a trade-off between robustness to the effects of noisy data (large λ) and fast convergence to changes in the parameter values (small λ).

7.1 RLS with 'forgetting factor'

Conveniently the recursive least squares equations can be easily modified to incorporate the forgetting factor λ. The appropriate equations are

$$\hat{\theta}(N) = \hat{\theta}(N-1) + P(N)\phi(N)(y(N) - \phi(N)^T \hat{\theta}(N-1)) \quad (6)$$

$$P(N) = \frac{1}{\lambda}\left[P(N-1) - \frac{P(N-1)\phi(N)\phi(N)^T P(N-1)}{\lambda + \phi(N)^T P(N-1)\phi(N)}\right] \quad (7)$$

References

[1] *ELEN90052 Advanced Signal Processing: lecture notes.*