DSP Project

Spectrum estimation and texture modeling

This document provides an outline for a Matlab project to be completed by the end of the course. You are expected to investigate, in detail, methods related to solving the problem. There is a design element to the project, and a quantitative evaluation of the performance of the proposed methods must be performed and presented. You are to write up a comprehensive report (of no more than 8 pages) describing your method and results. You should ideally work in groups of two, although you may work alone if you really want to. If you wish to propose a project of your own, then please come and talk to me.

1 The objective

The aim of this project is to investigate a class of parametric spectrum estimation techniques. The method will use an all-pole filter driven by white noise to generate samples from the model, with the parameters being estimated using linear prediction on the inverse model. Because the model extends quite easily to 2D (or more), the spectrum estimates can be used to characterise textures in a generative manner.

2 Basic Theory

The starting point is a time-recursive model of the form

$$x[n] = a_1 x[n-1] + a_2 x[n-2] + \cdots + a_p x[n-p] + w[n].$$

The signal $x[n]$ represents the observed data, and $w[n]$ is assumed to be Gaussian white noise. The model therefore states that the sample $x[n]$ at time n can be generated by a linear combination of previous inputs, plus an additive innovation (random component) $w[n]$. The weights $a_1, \ldots, a_p$ are parameters, and different choices lead to a different model. This is called an autoregressive (AR) process.

We can consider $x[n]$ to be the output of a linear time-invariant filter driven by the input sequence $w[n]$. This is apparent in the transform domain:

$$X(z) = \sum_{k=1}^{p} a_k z^{-k} X(z) + W(z) \Longrightarrow H(z) = \frac{X(z)}{W(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k z^{-k}}.$$

Since the denominator is a polynomial in z^{-1}, the input-output relation is in the form of an all-pole filter.

If $w[n]$ is white noise, then its power spectrum is flat and we can assume $|W(e^{j\omega})|^2 = \sigma^2$. For any set of values $a_1, \ldots, a_p$ appropriately chosen for the data, we have

$$X(e^{j\omega}) = H(e^{j\omega})W(e^{j\omega}) = \sigma H(e^{j\omega}) = \frac{\sigma}{1 - \sum_{k=1}^{p} a_k e^{-j\omega k}}.$$

The parameters a_k and σ^2 therefore provide a parametric description of the spectrum $X(e^{j\omega})$, and estimating their values from data $x[n]$ constitutes parametric spectrum estimation.

It turns out that it's not difficult to estimate the parameter values a_k given a sample of data. Suppose as a simple example that $p = 2$ and the observed data is $x[0], x[1], x[2], x[3], x[4]$. If the model is accurate, then we should have

$$\begin{aligned} x[2] &\approx a_1 x[1] + a_0 x[0] \\ x[3] &\approx a_1 x[2] + a_0 x[1] \\ x[4] &\approx a_1 x[3] + a_0 x[2]. \end{aligned}$$

In general we should have $x[n] \approx a_1 x[n-1] + a_2 x[n-2]$ when the parameters a_1 and a_2 are chosen correctly.

Define $\hat{x}[n] = a_1 x[n-1] + a_2 x[n-2]$ to be the predicted value of $x[n]$ using previous samples according to the model. The argument just presented states that we should have $x[n] \approx \hat{x}[n]$. However, from the model $w[n] = x[n] - \hat{x}[n]$, so the innovation sequence $w[n]$ corresponds directly to the prediction error. The model will be good if the overall error over the entire set of training equations is small. For the example above we can measure how good the model is by finding $E = \sum_{n=2}^{4} (w[n])^2$: a good model will have a small total error, while for a bad model E will be large.

The set of errors can be written as a vector, leading to a system of equations:

$$\mathbf{w} = \begin{pmatrix} w[2] \\ w[3] \\ w[4] \end{pmatrix} = \begin{pmatrix} x[2] \\ x[3] \\ x[4] \end{pmatrix} - \begin{pmatrix} x[1] & x[0] \\ x[2] & x[1] \\ x[3] & x[2] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix},$$

which is in the form $\mathbf{w} = \mathbf{x} - \mathbf{X}\mathbf{a}$. The error defined earlier is $E = \mathbf{w}^T\mathbf{w}$. The objective is to choose $\mathbf{a}$ to minimise E, which ideally would yield $\mathbf{w} = \mathbf{0}$. The solution can be found by least squares[1]:

$$\begin{aligned} \mathbf{w} \approx \mathbf{0} &\Longrightarrow \mathbf{X}\mathbf{a} \approx \mathbf{x} \Longrightarrow \mathbf{X}^T\mathbf{X}\mathbf{a} \approx \mathbf{X}^T\mathbf{x} \\ &\Longrightarrow \mathbf{a} \approx (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{x}. \end{aligned}$$

This is easily calculated using a high-level computer programming language like Matlab — we just need to form the vector $\mathbf{x}$ and the matrix $\mathbf{X}$ from training samples, and solve using the pseudo-inverse solution above.

In general, given observations $x[0], \ldots, x[N]$ and a model of order p, the errors are given by

$$\mathbf{w} = \begin{pmatrix} w[p] \\ w[p+1] \\ \vdots \\ w[N] \end{pmatrix} = \begin{pmatrix} x[p] \\ x[p+1] \\ \vdots \\ x[N] \end{pmatrix} - \begin{pmatrix} x[p-1] & x[p-2] & \cdots & x[0] \\ x[p] & x[p-1] & \cdots & x[1] \\ \vdots & \vdots & \ddots & \vdots \\ x[N-1] & x[N-2] & \cdots & x[N-p] \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_p \end{pmatrix},$$

This can be solved in exactly the same way as shown in the previous case.

Once a_k are specified, the filter $H(z)$ is completely defined. Driving this filter with a white noise input sequence will create an output that has similar spectral properties to that of the signal used to estimate the parameters. It is this characteristic that makes the model generative.

The methods in the previous section extend to 2D with an appropriate definition of the lags to use. Suppose in 2D the model order is (p, q) then we have

$$x(n_1, n_2) = \begin{pmatrix} a(p,q)x(n_1 - p, n_2 - q) & + & \cdots & + & a(0,q)x(n_1, n_2 - q) \\ & + & \cdots & + & \\ a(p,0)x(n_1 - p, n_2) & + & \cdots & & (+0x(n_1, n_2)) \end{pmatrix} + w(n_1, n_2)$$

[1]Formally we need to minimise $E = \mathbf{w}^T\mathbf{w}$ subject to $\mathbf{w} = \mathbf{x} - \mathbf{X}\mathbf{a}$. We can write

$$E = (\mathbf{x} - \mathbf{X}\mathbf{a})^T(\mathbf{x} - \mathbf{X}\mathbf{a}) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T\mathbf{X}\mathbf{a} + \mathbf{a}^T\mathbf{X}^T\mathbf{X}\mathbf{a}.$$

The minimum occurs when the derivative with respect to $\mathbf{a}$ is zero:

$$\frac{dE}{da} = -2\mathbf{X}^T\mathbf{x} + 2\mathbf{X}^T\mathbf{X}\mathbf{a} = 0,$$

or $\mathbf{a} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{x}$.

We can again take multiple training instances to form $\mathbf{w} = \mathbf{y} - \mathbf{X}\mathbf{a}$, where each row of $\mathbf{X}$ represents a set of "past" samples predicting the corresponding value in $\mathbf{y}$.

3 Approach

The aim here is for you to investigate and explore around the concepts described in the previous section. There are no specific objectives — you can take your analysis in any direction you choose, either concentrating on 1D or extending to higher dimensions, or considering how the ideas could be used in an application. To get you started there is demonstration code and some sample images available at

`http://www.dip.ee.uct.ac.za/~nicolls/eee4001f/projects/project08`.

The first script, `project08_runme.m`, contains examples of 1D parametric AR spectrum estimation using a single row of the first frame of a video of moving ocean waves as a sample signal. In practice one wouldn't usually process images in this way, since you would be ignoring all of the interesting image structure that occurs from the samples being ordered in the vertical direction (and in the temporal direction for video). In any event, the pixel values in the row are distinctly correlated, and treating them under a typical white noise assumption is probably inappropriate. Once the model is estimated, the code shows how to generate (or draw) samples from the same model or distribution.

The second script, `project08_runme2.m`, essentially does the same as `project08_runme.m`, but in a 2D setting. Ordering of parameters and corresponding observations is really the only complicated part of this extension. The methods are demonstrated on texture images that are a subset of the Brodatz album[2].

The third script, `project08_runme3.m`, gives some ideas of how the spectrum estimation process can be used to classify textures into different classes. A number of texture images are considered, and a 1D model of the chosen order is fitted to each row. A scatterplot on the filter coefficients that represent the texture indicates that there is some structure that could be useful for the task of classification. Note that the classification system hasn't actually been built — the results just show that the filter parameters could be considered reasonable features.

The examples raise a number of questions, any of which might form the basis for your exploration:

1. In all cases the model order, or the number of lags to use, has to be fixed in advance. A model with more parameters has more "capacity" to represent the actual underlying spectrum, but overfitting may occur if the number of parameters is too high. You could investigate around this factor using different types of data. (Some is provided with this project, but there's plenty on the web.)

2. The generative aspect of the model lets you sample instances from it, to determine how similar the model is to the data it was estimated from. This is particularly interesting for 2D image texture analysis, where the visual appearance of the synthesized texture is meaningful to us. There are clearly examples of things we would call textures that don't conform to the model. There are many 2D texture databases available on the web if you need more samples.

3. The first example uses a row of data from a video image of ocean waves, and arises from a project where we're trying to develop visual trackers for maritime surveillance. The extension to 2D has been presented, but the extension to a full 3D spatio-temporal AR model is useful and interesting — particularly when you draw samples from it. This forms part of an interesting research endeavour that goes by the name "dynamic textures", which involves the

[2]`http://www.ux.uis.no/~tranden/brodatz.html`.

analysis and synthesis of textures through time. If you need more examples of spatiotemporal textures, try the DynTex database[3]. Matlab's `im2col` function unfortunately doesn't extend to more than two dimensions, but the function `im2colp` in the project directory provides this functionality.

4. For the classification problem, we would expect a 2D model of the texture to be much more discriminatory than a 1D row from it. Extending the model to the higher dimension is easy, although plotting feature values on a scatter plot will no longer be easy (or possible) — some kind of distance measure with a nearest neighbour classifier may be required. For testing performance it would be appropriate to split a single input image into blocks, using some for training and some for classifier testing.

4 Requirements

As always, you will have to formulate and design experiments that demonstrate the point you want to make, and generate a comprehensive set of results to assess your findings. This is always difficult: you need to develop performance measures that are effective at quantifying how well the system is working. Just saying "It looks right" does not suffice. Exploring around the different choices of parameter settings is also usually appropriate.

Any interesting findings or observations regarding the structure of the problem or uses in practical applications will be rewarded during the assessment.

The report should not exceed 8 pages. While it can discuss the software structure or design it is not appropriate to include source code: the project is to assess the *method*, not your specific implementation of it (although that might deserve a mention, particularly if the implementation is innovative).

[3] `http://projects.cwi.nl/dyntex.`