Laboratory Project 2:

Spectral Analysis and Optimal Filtering

Random Processes With Applications (MVE 135)

Mats Viberg
Department of Signals and Systems
Chalmers University of Technology
412 96 Gteborg, Sweden
Email: viberg@chalmers.se

Oct. 2007. Last update Aug. 2008.

1 Introduction

The purpose of this lab is to provide some hands-on experience and to give a better understanding of spectral analysis and Wiener filtering. The focus in this lab is on the discrete-time case, i.e., digital signals. After completing the lab, the student should be able to analyze a given signal with different non-parametric and parametric (AR) methods, and select appropriate windows etc. based on prior knowledge and the analysis results. The student should also be able to select a suitable order and to compute FIR Wiener filters for given signal scenarios. This involves theoretical calculation of the autocorrelation and cross-correlation functions for simple cases. He/she should also have a good understanding of the benefits and the limitations of such filters. A theoretical background to the topics under study is given in [1].
The tasks are performed in Matlab, using mostly commands from the Signal Processing Toolbox. The lab project is time consuming (expect to spend at least 10 hours) and there are many tasks. It is a good idea to start with the first tasks as early as possible, although the theoretical background for the latter tasks is still lacking. You should work in groups of two people. The results should be documented in a report with some 10 pages. Include relevant Matlab plots and code. The deadline for handing in the report will be posted on the course homepage. A preliminary date is Oct. 20. Submit the report as a pdf-file by email to `viberg@chalmers.se`.

2 Spectral Estimation

The first part of the lab deals with spectral estimation. You will investigate various methods using simulated data. All experiments assume discrete-time data. A sequence of "observed" data $x[n]$, $n = 0, 1, \ldots, N-1$ will be generated from a true model, which is a stationary AR or ARMA model. In the first part you will apply non-parametric (Fourier-based) methods. The periodogram is first applied, and then its various modifications. After that you will get some experience from parametric AR modeling.

Task 1: The Periodogram and Averaging

The first test system is the AR(2) model:

$$x[n] - 1.5x[n-1] + 0.64x[n-2] = e[n]\,,$$

where $e[n]$ is WGN with zero mean and variance $\sigma_e^2 = 1$.

1. Plot the true spectrum, which is given by $P(e^{j\omega}) = \sigma_e^2 |G(e^{j\omega})|^2$, with $G(z) = 1/A(z)$. Use the command `freqz` in Matlab[1]. Plot the spectrum versus the frequency axis `f=0:0.01:0.5;` (unit: "revolutions per sample"). Use linear amplitude and frequency scales.

2. Generate $N+L$ samples of data according to the above model, where $N = 1024$ and $L = 50$. Use the `randn` and `filter` commands in Matlab. Then, discard the first L samples of the data to ensure that the simulated data is stationary. In Matlab, this can easily be done by `x=x(L+1:N);`. Next, estimate the spectrum using the Periodogram. Follow the steps 1) `X=fft(x,N);`, 2) `P=X.*conj(X)/N;`, and 3) `P=P(1:N/2);` (we only look at the first half, since the spectrum of a real-valued signal is symmetric). Plot the Periodogram versus the frequency axis `f=0:1/N:(N-1)/(2*N);` Compare the periodogram and the true spectrum. Comment on the result!

3. Clearly, the original Periodogram is a very noisy estimate of the true spectrum, although it appears to be "on average" correct. The first attempt to improve the situation is to divide the data into K non-overlapping segments, each of length $M = N/K$. Then the periodograms from each segment are averaged. This is called Barlett's method, or just periodogram averaging. In Matlab, this is most conveniently done by 1) `xx=reshape(x,M,K);`, 2) `XX=fft(xx);`, 3) `PP=XX.*conj(XX)/M;`, 4) `PB=mean(PP');`, and 5) `PB=PB(1:M/2)';`. Apply Bartlett's method using first $K = 4$ and then $K = 16$. Plot the resulting spectral estimates along with the true spectrum in the same plot. What is the benefit of periodogram averaging? What is the drawback?

Task 2: Windows in the Time and Frequency Domains

Windowing plays a key role in all non-parametric spectral estimates. So far you have used a rectangular window. This gives the most narrow mainlobe (=best frequency resolution), but very high sidelobes (=frequency masking). In this task, you will experiment with Matlab's graphical user interface `wintool` in order to get some familiarity with the some common window functions. Start the GUI by typing `wintool`.

1. Make a table that compares the window shapes "rectangular", "bartlett" (triangular), "hann" (or Hanning), and "hamming". Compare the performance parameters "mainlobe width" (3 dB, twosided) and "peak sidelobe", in dB relative the mainlobe. Use the lengths $N = \{16, 32, 64, 128\}$ respectively. How do the two performance parameters depend on N for the different windows?

2. For $N = 128$, apply a Chebyshev window with the same peak sidelobe as the Hamming window. Which of the two has the better frequency resolution?

3. Finally, look at a Flat Top window of length 128 in the frequency domain. What characterizes this window? Try to imagine what it can be used for!

4. Go back to the Matlab workspace. Choose your favorite window from the GUI and use it with Welch's method for spectral estimation (averaging windowed and overlapping periodograms). In Matlab, this is called `pwelch`. Compare the result with what you got from "pure" periodogram averaging.

Task 3: Blackman-Tukey's Method

In this task you will have to do some Matlab-programming yourself. The task is to implement Blackman-Tukey's method, also known as Periodogram smoothing. Although a frequency-domain implementation may be more efficient from a computational point of view, you will here implement the method in the time domain. Recall the definition:

$$\hat{P}_{BT}(e^{j\omega}) = \sum_{k=-M}^{M} w_{lag}[k]\hat{r}_x[k]e^{-j\omega} = \mathrm{DTFT}\{w_{lag}[k]\hat{r}_x[k]\},$$

[1] Generally, the `typewriter` font is used for Matlab commands in this document.

where $w_{lag}[k]$ is the covariance window of size $2M+1$, $M \leq N$, and $\hat{r}_x[k]$ is the sample autocorrelation function,

$$\hat{r}_x[k] = \frac{1}{N}\sum_{n=k}^{N-1} x[n]x[n-k]\,.$$

Thus, the steps are: 1) generate the window function, 2) compute the sample autocorrelation function, and 3) compute the FFT of the product between the two.

1. Write a Matlab function:

   $$[\texttt{PBT}, \texttt{fgrid}] = \texttt{btmethod}(\texttt{x}, \texttt{M}, \texttt{NFFT});$$

   that takes the data `x`, the window size `M` and the FFT length `NFFT` as inputs, and delivers the BT spectral estimate `PBT` as output, computed at the frequency grid given by `fgrid`. For step 1), use a `hamming` window, and in step 2) you could use the `xcorr` command (which generates correlations for lags $k = -N+1$ to $k = N-1$, note the indices of the output vector!). Include the code for the BT function in the report.

2. Test the code by estimating the spectrum for the same data as in Task 1. Choose a "suitable" window size (what dictates this choice?). Let `NFFT` be 2048 to get a smooth graph. Plot the estimated spectrum and compare to the results using Bartlett's method.

Task 4: Parametric AR Modeling

In this task you will test how to use Matlab's built in command `pyulear` to estimate the spectrum. This function computes estimates of the AR parameters by solving the Yule-Walker equations, using the sample autocorrelations. The spectrum is then formed from the estimated AR coefficients and noise variance. Note that the level of the estimated spectrum may be wrong in some Matlab versions. You can then use `aryule` instead, to get the estimated AR parameters and noise variance. Once these are computed, the spectrum can be formed using `freqz`, just like when computing the spectrum of the true model.

1. Apply the `pyulear` with correct model order (since the true data is an AR process in this case). Compare the results with that using non-parametric methods.

2. Try AR modeling with incorrect model orders. What happens when the order is too low? Too high?

Task 5: ARMA Model

Now it is time to try a more challenging data set. Generate a set of $N = 8192$ samples from the ARMA process:

$$x[n] - 0.24x[n-1] + 0.08x[n-2] - 0.37x[n-3] + 0.52x[n-4] = e[n] + 0.56e[n-1] + 0.81e[n-2]\,,$$

where $e[n]$ is $N(0,1)$ white noise.

1. Plot the true spectrum, again using linear scales.

2. Estimate the spectrum using the BT method. Try to find the "best" window size. Plot the estimate and the true spectrum in the same plot. Use `NFFT=4096`. You may also try different window shapes. Which part of the spectrum is most difficult to estimate?

3. Now, try AR modeling using the `pyulear` command. Try to find the "best" choice of model order. Compare the results with that using the BT method. Is it possible to get a good spectrum estimate using AR modeling in this case? If not, then why?

3 Optimal Filters

A common situation in signal processing is that one has a certain desired signal $d[n]$ which is not observable. Instead, another measured signal $x[n]$ is used to recover $d[n]$. In the case of an FIR Wiener filter, the signal estimate is:

$$\hat{d}[n] = h[n] * x[n] = \sum_{k=0}^{p-1} h[k]x[n-k]\,,$$

where p is the number of filter coefficients $h[k]$. In the Wiener filter, the $h[k]$ are selected to minimize the MSE:

$$E\{(d[n] - \hat{d}[n])^2\}\,.$$

Since $\hat{d}[n]$ is linear in the data, we call it the LMMSE (Linear Minimum Mean Square Error) estimate of $d[n]$. The optimal filter coefficients are found by solving the Wiener-Hopf (W-H) equations. For the case of an FIR filter, the W-H equations are given by

$$\begin{bmatrix} r_x[0] & r_x[1] & \cdots & r_x[p-1] \\ r_x[1] & r_x[0] & \cdots & r_x[p-2] \\ \vdots & \vdots & \ddots & \vdots \\ r_x[p-1] & r_x[p-2] & \cdots & r_x[0] \end{bmatrix} \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[p-1] \end{bmatrix} = \begin{bmatrix} r_{dx}[0] \\ r_{dx}[1] \\ \vdots \\ r_{dx}[p-1] \end{bmatrix},$$

where $r_x[k]$ is the autocorrelation function of $x[n]$ and $r_{dx}[k]$ is the cross-correlation. We can express the W-H equations compactly as $\mathbf{R}_x\mathbf{h} = \mathbf{r}_{dx}$, and the solution is given by

$$\mathtt{h} = \mathtt{R_x} \backslash \mathtt{r_{dx}}$$

(in Matlab notation). This is the same as $\mathbf{R}_x^{-1}\mathbf{r}_{dx}$, but Matlab implements the "backslash" in a more computationally and numerically efficient way. The minimum MSE is given by

$$E\{(d[n] - \hat{d}[n])^2\} = r_d[0] - \sum_{k=0}^{p-1} r_{dx}[k]h[k] = r_d[0] - \mathbf{r}_{dx}^T\mathbf{h}\,.$$

Once the optimal filter has been found, the signal estimate is

$$\hat{d}[n] = \sum_{k=0}^{p-1} h[k]x[n-k] = h[n] * x[n] = \mathbf{x}^T[n]\mathbf{h}\,.$$

Task 6: The FIR Wiener Filter

In this task, we will consider the "filtering" problem, in which the desired signal is corrupted by additive noise:

$$x[n] = d[n] + w[n]\,.$$

The desired signal $d[n]$ and the disturbance $w[n]$ are uncorrelated, and they are both modeled as filtered white noises:

$$\begin{aligned} d[n] &= e_d[n] + 0.8e_d[n-1] + 0.2e_d[n-2] \quad (\mathrm{MA-process}) \\ w[n] &= -0.94w[n-1] - 0.64w[n-2] + e_w[n] \quad (\mathrm{AR-process}) \end{aligned}$$

Here, $e_d[n]$ and $e_w[n]$ are uncorrelated white noises, with zero mean and variances $\sigma_{ed}^2 = 1$ and $\sigma_{ew}^2 = 0.5$ respectively. In the simulation, both will be generated as Gaussian processes.

1. (Theoretical task) Determine the autocorrelation functions for $d[n]$, $w[n]$ and $x[n]$ respectively. See Problem 4 in Chapter 5 of [1].
2. Plot the spectra of the desired signal $d[n]$ and the disturbance $w[n]$ in the same plot.

3. Compute the optimal filter coefficients for FIR filters of length $p = \{1, 2, 5\}$ and $p = 20$ respectively ($p = 20$ is assumed to be close enough to ∞ in this case, so this should be a good approximation of the causal IIR filter). When forming the autocorrelation matrix $\mathbf{R}_x$, you may find the `toeplitz` command useful. Solve the W-H equations using the backslash command in Matlab. Plot the frequency responses of the different filters. Do they look as you expected, after seeing the spectra of the desired signal and the interference?

4. Generate $N = 200$ samples of the two driving noise processes $e_d[n]$ and $e_w[n]$. Use the `randn` command, and scale with the standard deviations. Then generate the signals $d[n]$, $w[n]$ and $x[n]$ according to their respective model. Plot the last 50 samples of $d[n]$ and $x[n]$. The task is now to filter $x[n]$ in order to recover $d[n]$.

5. Compute the theoretical MSEs for the different filters. What is a good choice of filter length in this case? Then apply the filters to the signal $x[n]$ you generated in the previous task. Then compute the error signals $e_p[n] = d[n] - \hat{d}_p[n]$, with $p = \{1, 2, 5, 20\}$. Find the empirical mean-square errors, which are the sample variances of $e_p[n]$ (discarding the first 20 samples), and compare with the theoretical results. Also plot the last 50 samples of the various error signals in the same plot.

6. (Voluntary task) Suppose in reality we do not know of any models for the various signals. Instead we might have access to a "training data set", $\{\tilde{x}[n], \tilde{d}[n]\}_{n=0}^{K-1}$, which is different from the true signals, but has the same statistical properties. How could we use the training data to construct an approximate Wiener filter? Feel free to try the idea on your simulated data, and compare the result to the optimal filter!

References

[1] M. Viberg, "Complement on Digital Spectral Analysis and Optimal Filtering: Theory and Exercises", Course notes for MVE 135, Aug. 2008.