Laboratory Project 2:

Spectral Analysis and Optimal Filtering

Random signals analysis (MVE136)

Mats Viberg and Lennart Svensson
Department of Signals and Systems
Chalmers University of Technology
412 96 Göteborg, Sweden

September 2013

1 Introduction

The purpose of this lab is to provide 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, you should be able to

- use nonparametric methods to estimate the power spectral density of a signal and know what the pros and cons are of the different methods.
- estimate the parameters in an AR model, compute the corresponding spectrum and explain for what type of signals the model is appropriate.
- implement causal FIR Wiener filters (optimal FIR filters) for different filter lengths and know what the properties of the filter should be (in particular in terms of the frequence response).

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 on the order of 10 pages. Include relevant Matlab plots and code. Remember to motivate/explain the obtained results clearly (and please do not fail to label the axes in your figures!).

2 Some Background Theory

In this section we provide a compact summary of the theory needed to perform this lab. For a more complete description, see for instance the lecture notes [1].

2.1 Power Spectral Density

The auto-correlation function of a discrete-time signal $x[n]$ is given by $r_x(n, n-k) = E\{x[n]x[n-k]\}$, and the signal is said to be (weakly) stationary if $r_x(n, n-k) = r_x(k)$, i.e., if it does not depend on n. For stationary signals, the power $P_x = r_x(0) = E\{x^2[n]\}$ can be split up into frequency components. This results in the Power Spectral Density (PSD) (often just referred to as the "spectrum"), which is the Fourier transform (DTFT) of the auto-correlation:

$$P_x(e^{j\omega}) = \sum_{n=-\infty}^{\infty} r_x[n]e^{-j\omega n}. \tag{1}$$

The spectrum can also be defined using the DTFT directly on the signal:

$$P_x(e^{j\omega}) = \lim_{N\to\infty} \frac{1}{N} E\{|X_N(e^{j\omega})|^2\} \tag{2}$$

where

$$X_N(e^{j\omega}) = \sum_{n=0}^{N-1} x[n]e^{-j\omega n}. \tag{3}$$

From (2) and (3) it is clear that $P_x(e^{j\omega})$ represents the spectral content of the process $x[n]$.

2.2 Parametric Spectral Estimation

The spectral estimation problem is to determine an approximation of $P_x(e^{j\omega})$ from data, usually in the form of one realization of finite length: $\{x[n]\}_{n=0}^{N-1}$. Given (2), a natural estimate of $P_x(e^{j\omega})$ is the **Periodogram**

$$\hat{P}_{per}(e^{j\omega}) = \frac{1}{N} |X_N(e^{j\omega})|^2. \tag{4}$$

The Periodogram (and other non-parametric spectral estimation techniques) is usually characterized by the following properties:

Resolution This is the ability to separate two peaks in the spectra. For the periodogram, the resolution is $\Delta\omega \approx 2\pi/N$, i.e. inversely proportional to the sample length (and thus the data collection time).

Frequency masking This is due to the sidelobes of the rectangular windowing of data. The peak sidelobe of the periodogram is -13 dB below the mainlobe, which means that a frequency peak that is more than 13 dB weaker than a strong neighboring peak, might not be visible in the spectrum.

Variance This measures the statistical variability, i.e. how similar the spectral estimate would be when applied to different realizations of the same random process. Unfortunately, $\mathrm{Var}\{\hat{P}_{per}(e^{j\omega})\} \approx P_x^2(e^{j\omega})$, which does not improve with increasing N. Thus, the Periodogram always gives a very noisy estimate of the PSD.

The properties of the periodogram can be improved by:

- **Modified periodogram** – leakage through sidelobes (frequency masking) can be reduced by applying a data window that modifies the sidelobes.
- **Bartlett's method** – one may reduce the variance by dividing data into K non-overlapping blocks, each of length $M = N/K$, and compute an average periodogram. As a consequence, the variance is reduced by factor K, but unfortunately frequency resolution is reduced to $2\pi K/N$.
- **Welch's method** – a better trade-off between variance reduction and loss of resolution may be obtained by instead averaging overlapping data combined with windowing.

Blackman-Tukey's Method Another approach to improve non-parametric spectral estimation is to instead window the covariance function. The starting point is the following alternative form of the periodogram

$$\hat{P}_{per}(e^{j\omega}) = \sum_{k=-(N-1)}^{N-1} \hat{r}_x(k)e^{-j\omega k} \tag{5}$$

where

$$\hat{r}_x(k) = \frac{1}{N}\sum_{n=k}^{N-1} x[n]x[n-k] \tag{6}$$

is the sample autocorrelation. Here the periodogram in (4) has been rewritten in a form that resembles (1).
Note that $\hat{r}_x[k]$ is based on averaging $N-k-1$ terms, and it is therefore less reliable for increasing $|k|$. To mitigate this effect, a window can be applied to the autocorrelation in (5). We denote the chosen window function $w_{lag}[k]$, and it is assumed to be symmetric of length $2M+1$, where $M < N$, and a decreasing function of $|k|$. The Blackman-Tukey PSD estimate is defined by

$$\hat{P}_{BT}(e^{j\omega}) = \sum_{k=-M}^{M} w_{lag}[k]\hat{r}_x(k)e^{-jk\omega}. \tag{7}$$

The intuitive idea behind (7) is to reduce the variance by putting small weights on estimates $\hat{r}_x(k)$ with high variance. The drawback is (as usual) that we increase the bias.
Since multiplication in time domain corresponds to convolution in the frequency domain, an alternative form of the B-T spectral estimate is given by

$$\hat{P}_{BT}(e^{j\omega}) = \frac{1}{2\pi}W_{lag}(e^{j\omega}) * \hat{P}_{per}(e^{j\omega}) = \frac{1}{2\pi}\int_{-\pi}^{\pi} \hat{P}_{per}(e^{j(\omega-\xi)})W_{lag}(e^{j\xi})d\xi. \tag{8}$$

The convolution of $\hat{P}_{per}(e^{j\omega})$ with $W_{lag}(e^{j\omega})$ leads to a "smearing" of the spectral estimate. The B-T estimate is therefore often referred to as a **smoothed** periodogram. The frequency resolution of the B-T estimate is determined by mainlobe width of $W_{lag}(e^{j\omega})$, and one can show that the variance is approximately given by

$$\mathrm{Var}\{\hat{P}_{BT}(e^{j\omega})\} \approx \frac{1}{N}\left(\sum_{k=-M}^{M} w_{lag}^2(k)\right) P_x^2(e^{j\omega})$$

Thus, for a fixed frequency resolution (proportional to $1/M$)), the variance decreases like $1/N$, unlike the standard periodogram.

2.3 Auto-Regressive Model Estimation

An alternative to non-parametric, or model-free, methods, is to assume a specific parametric model for $x[n]$. The parameters in that model can then be estimated from data and given the parameters we typically know the spectrum. One popular choice of model is the Auto-Regressive (AR) model. A pth order AR model is defined by

$$x[n] + a_1x[n-1] + \cdots + a_px[n-p] = e[n]\,, \tag{9}$$

where $e[n]$ is white noise (a sequence of i.i.d. random variables), with zero mean and variance $E\{e^2[n]\} = \sigma_e^2$. The spectrum of the AR process is given by

$$P_x(e^{j\omega}) = \frac{\sigma_e^2}{|A(e^{j\omega})|^2}\,. \tag{10}$$

For large enough p, this can model "any" spectrum, but the success of the method depends on if a good approximation can be found for a "moderate" p. AR modeling is usually good for finding peaks in the spectrum, but less accurate with dips (notches).
The AR coefficients and the noise variance can be determined from data by using the **Yule-Walker method**. The Yule-Walker (Y-W) equations are given by

$$\begin{bmatrix} \hat{r}_x(1) & \hat{r}_x(2) & \cdots & \hat{r}_x(p) \\ \hat{r}_x(2) & \hat{r}_x(1) & \cdots & \hat{r}_x(p-1) \\ \vdots & \vdots & \ddots & \vdots \\ \hat{r}_x(p) & \hat{r}_x(p-1) & \cdots & \hat{r}_x(1) \end{bmatrix} \begin{bmatrix} \hat{a}_1 \\ \hat{a}_2 \\ \vdots \\ \hat{a}_p \end{bmatrix} = - \begin{bmatrix} \hat{r}_x(0) \\ \hat{r}_x(1) \\ \vdots \\ \hat{r}_x(p-1) \end{bmatrix} . \tag{11}$$

Once the AR coefficients have been found by solving (11), the noise variance is estimated by

$$\hat{\sigma}_e^2 = \hat{r}_x(0) + \sum_{k=1}^{p} \hat{a}_l \hat{r}_x(k) \tag{12}$$

and the AR spectral estimate is computed as

$$\hat{P}_{AR}(e^{j\omega}) = \frac{\hat{\sigma}_e^2}{|\hat{A}(e^{j\omega})|^2}. \tag{13}$$

In general, the AR estimate provides better resolution and less problems with frequency masking than non-parametric methods. It also typically has smaller variance since the number of parameters to estimate is much smaller. However, it may give a bias if the "true" spectrum cannot be well approximated by the AR model (10).

2.4 Wiener Filtering

Wiener's classical filter problem is to recover a desired, but unobservable, signal $d[n]$ by filtering a measured signal $x[n]$ through a linear filter $H(z) = \sum_n h[n]z^{-n}$. The error is $e[n] = d[n] - y[n]$, where

$$y[n] = h[n] * x[n] = \sum_k h[k]x[n-k] \tag{14}$$

is the filter output, and the filter coefficients, $h[\cdot]$, are selected to minimize the Mean-Square Error (MSE) $E\{e^2[n]\}$. Thus, $y[n]$ is a Linear Minimum MMSE (LMMSE) estimate of $d[n]$.
The optimal filter coefficients are found by solving the *Wiener-Hopf* (W-H) equations

$$\sum_k h[k]r_x(l-k) = r_{dx}(l)\,, \tag{15}$$

which ensure that the error is orthogonal (uncorrelated) to the measurement $x[n-l]$. The range of l for which (15) must hold is the same as the range of k in $H(z) = \sum_k h[k]z^{-n}$, and depends on the type of filter. For causal filters, $k \geq 0$, and for a finite impulse response (FIR) filter $k \leq p-1$, for some finite p. Thus, in case of a causal FIR filter the estimate is given by

$$y[n] = \hat{d}[n] = \sum_{k=0}^{p-1} h[k]x[n-k]. \tag{16}$$

Since the W-H equations (15), must hold for $l = 0, \ldots, p-1$, they can be put on matrix form:

$$\begin{bmatrix} r_x(0) & r_x(1) & \cdots & r_x(p-1) \\ r_x(1) & r_x(0) & \cdots & r_x(p-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_x(p-1) & r_x(p-2) & \cdots & r_x(0) \end{bmatrix} \begin{bmatrix} h[0] \\ h[1] \\ \vdots \\ h[p-1] \end{bmatrix} = \begin{bmatrix} r_{dx}(0) \\ r_{dx}(1) \\ \vdots \\ r_{dx}(p-1) \end{bmatrix} . \tag{17}$$

If we introduce the notation $\mathbf{R}_x\mathbf{h} = \mathbf{r}_{dx}$ for the quantities in (17), the solution is given by $\mathbf{h}_{FIR} = \mathbf{R}_x^{-1}\mathbf{r}_{dx}$. One can also show that the MSE for the LMMSE estimator is

$$E\{e^2[n]\}_{min} = r_d(0) - \sum_{k=0}^{p-1} r_{dx}(k)h_{FIR}[k]. \tag{18}$$

Note that the W-H equations are the same as the Y-W equations if the desired signal is $d[n] = x[n+1]$, and with $h[k] = -a_{k+1}$. Thus, an LMMSE one-step ahead predictor of an AR process is given by

$$\hat{x}[n+1] = -a_1 x[n] - \cdots - a_p x[n-p+1]\,,$$

which could of course be guessed from (9). However, in general the W-H equations are different from Y-W, although they both involve the *auto-correlation matrix* $\mathbf{R}_x$.

3 Assignments

3.1 Nonparametric 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 this part you will apply non-parametric (Fourier-based) methods. The periodogram is first applied, and then its various modifications.

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 white Gaussian noise (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 `f0=0:0.001:0.5;` (unit: "revolutions per sample"). The syntax is `G = freqz(B,A,2*pi*f0);`, where `B=1;` and `A=[1 -1.5 0.64];` represent the numerator and denominator polynomials, respectively. Plot the amplitude in dB, using `20*log10(abs(G));`, versus the linear frequency scale `f0`.

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:end);`. 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 in dB versus the frequency axis `f=0:1/N:(N-1)/(2*N);` . Note that `P` has the unit "power" which means that we obtain dB by evaluating `10*log10(abs(P));`. 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 correct "on average". 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,N);` (it is practical to still use the same number of frequency bins, so that the same frequency axis can be used), 3) `PP=XX.*conj(XX)/M;`, 4) `PB=mean(PP');`, and 5) `PB=PB(1:N/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?

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

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 ($\rightarrow$ best frequency resolution), but very high sidelobes ($\rightarrow$ frequency masking). In this task, you will experiment with Matlab's graphical user interface `wintool` in order to get some familiarity with 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. Go back to the Matlab workspace. Choose a Hamming window of length $M = 64$ and use it with Welch's method for spectral estimation (averaging windowed and overlapping periodograms). Use the command `PW = pwelch(x,window,[],2*pi*f)*2*pi;` (see the note on scaling at the end of Task 4). Choosing the default `[]` for the `NOVERLAP` parameter results in a 50% overlap, which is generally a good choice. 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:

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

where $w_{lag}[k]$ is the covariance window of size $2M+1$, $M \leq N$ (usually $M \ll N$), 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 number of correlation lags `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 of length $2M + 1$, and in step 2) you can use the command `rx = xcorr(x,M,'biased');`. Include the code for the BT function in the lab report.

2. Test the code by estimating the spectrum for the same data as in Task 1. Choose a "suitable" number of lags M, which means that the estimator should smooth as much as possible without losing the details of the true spectrum. How does the amount of smoothing and the frequency resolution depend on M? In a real application the true spectrum is of course unknown, which means that the window size is often selected to give a "sufficient" resolution for the application at hand. Let `NFFT` be 1024 as before. Plot the estimated spectrum and compare to the results using Bartlett's method.

3.2 Parametric Model Estimation

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. See below for a comment on Matlab's scaling of PSDs. As an alternative, you can use the `aryule` command, to retrieve the estimated AR parameters and noise variance. Once these are computed, the spectrum can be formed using `freqz`, similar to computing the spectrum of the true model.

1. Apply the `pyulear` with correct model order (this is possible 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 ($p = 1$)? Too high (try up to $p = 100$)?

Note on Matlab's PSD scaling: Unfortunately, Matlab uses a rather unusual scaling of the PSD estimates. As noted above, the `pwelch` command needs scaling by 2π to give the correct amplitude (probably because the spectral density is defined "per radians" instead of the more common "per Hz" definition). The required scaling in `pyulear` is instead a factor of π, which is harder to understand. Thus, the command is `[PAR,f_ar] = pyulear(x,order,NFFT)*pi;`, where `order` is the order of the AR model. This computes the AR-based spectrum estimate at the `NFFT/2 + 1` frequency points specified in `f_ar`.

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. Again, use `randn` to generate $e[n]$, and `filter` to compute $y[n]$.

1. Compute and plot the true spectrum in dB versus a linear frequency axis. Use the `freqz` command as in Task 1.1.

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=2048`. You may also try different window shapes. Which part of the spectrum is most difficult to estimate (peaks or valleys)?

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? Which part of the spectrum is most difficult to estimate using AR modeling (peaks or valleys)?

3.3 Optimal Filters

In this part you will try FIR Wiener filtering to recover a noisy signal. You shall first generate a desired signal and a noise, with given spectra. The optimal filter is then computed in Matlab by solving the Wiener-Hopf equations as

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

This is the same as $\mathbf{R}_x^{-1}\mathbf{r}_{dx}$, but Matlab implements the "backslash" in a more computationally efficient and numerically stable way. Once the optimal filter has been found, the signal estimate is

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

and the minimum MSE is computed from (18).

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.

$$\begin{aligned} d[n] - 0.13d[n-1] + 0.9d[n-2] &= e_d[n] \qquad &(\mathrm{AR-process}) \\ w[n] &= e_w[n] - 0.8e_w[n-1] + 0.2e_w[n-2] \qquad &(\mathrm{MA-process}) \end{aligned}$$

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

Task 6-A: ACF and PSD of signals and filter

1. Plot the spectra of the desired signal $d[n]$ and the disturbance $w[n]$ in the same plot. Use the `freqz` command, just as in Task 1.1. Verify that the desired signal is of band-pass character, whereas the disturbance is of high-pass type.

2. (Theoretical task) Show that the ACF for $d[n]$ is given by $r_d[0] = 5.2879$, $r_d[1] = 0.3618$, and $r_d[k] = 0.13r_d[k-1] - 0.9r_d[k-2]$ for $k \geq 2$. See [1, Example 3.1]. Then show that the ACF for $w[n]$ is given by $r_w[0] = 1.68$, $r_w[1] = -0.96$, $r_w[2] = 0.2$ and $r_w[k] = 0$ for $k > 2$, see [1, Example 3.2]. Also verify that $r_x[k] = r_d[k] + r_w[k]$.

3. Using the theoretical ACFs given in the previous task, compute the optimal filter coefficients for FIR filters of length $p = \{1, 2, 5\}$ and $p = 10$ respectively ($p = 10$ 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 as indicated above. Compute and plot the frequency responses of the different filters using `freqz`. Do they look as you expected, after seeing the spectra of the desired signal and the interference?

Task 6-B: optimal filters of different lengths

4. Use (18) to compute the theoretical MSEs for the different filters h_1, h_2, h_5 and h_{10} respectively. What is a good choice of filter length in this case?

5. It is now time to generate the simulated data and apply the FIR-Wiener filters. First, generate $N = 200$ samples of the two driving noise processes $e_d[n]$ and $e_w[n]$. Use the `randn` command. Then generate the signals $d[n]$ and $w[n]$ (use `filter`), and $x[n] = d[n] + w[n]$. Plot the last 50 samples of $d[n]$ and $x[n]$.

6. The next task is to filter $x[n]$ in order to recover $d[n]$. Apply the different filters to the simulated noisy measurement $x[n]$ that you generated in the previous task. Compute the error signals $e_p[n] = d[n] - \hat{d}_p[n]$, with $p = \{1, 2, 5, 10\}$. Find the empirical mean-square errors, which are the sample variances of $e_p[n]$. Use `MSE_emp = norm(e_p)^2/N`, where `N` is the number of samples. Compare the theoretical and empirical results – the agreement should be "fairly good" (on the order of $1/\sqrt{N}$).

References

[1] M. Viberg, "Complement on Digital Spectral Analysis and Optimal Filtering: Theory and Exercises", Lecture notes for MVE 136, September 2011.