Stationary stochastic processes,
Lund University,
Centre for Mathematical Sciences HT 2024,
Mathematical Statistics

Computer exercise 1 (Ladok-Lab 1)

In this computer exercise you will study estimation of expected value, covariance function and spectral density for some process realizations. You will also study and compare the frequency content from recordings of different music instruments.

- Please find all the additional files and data on the course webpage and download these into your computer. **Remember to save your commands and code in scripts for easier access of the results at the presentation. Please also comment well.**
- Please work in groups of two students!

In [ ]:
```python
# Import necessary libraries for the entire exercise.
import numpy as np
import matplotlib.pyplot as plt
import scipy.io
import scipy.signal
import sounddevice as sd

from help_funcs_SSP import signalsim, getCov, melSpectrogram, melFil

plt.close('all')
```

1 Introduction to estimation of expected value, covariance function and spectral density

Expected value, covariance function and spectral density are central concepts for modelling of stationary stochastic processes. In general, the modelling starts from measured data and therefore, we need to learn how to estimate these concepts from data.

1.1 Estimation of expected value

Load the file data.mat using the command load data. The file contains three realizations, each of 100 samples of Gaussian white noise.

In [ ]:
```python
# Load the data (You can use this way the entire course)
path_to_data = "data.mat" # Change to YOUR path to data.
data = scipy.io.loadmat(path_to_data,simplify_cells=True)
print(data.keys())
```

Plot the sequences,

In [ ]:
```
# Plot the sequences
plt.figure(figsize=(8, 4))
plt.plot(data["data1"], label='Realization 1')
plt.plot(data["data2"], label='Realization 2')
plt.plot(data["data3"], label='Realization 3')
plt.xlabel('Sample Index')
plt.ylabel('Value')
plt.title('Realizations of White Gaussian Noise')
plt.legend()
plt.show()
```

Q1. What conclusions can you make from this figure? Is it reasonable to assume that all realizations are from the same stationary stochastic process? Does this process have zero mean?

Estimate the three expected values

In [ ]:
```
m1 = np.mean(data["data1"],axis=0)

# Write your own code below:

print(m1,m2,m3)
```

Now, derive 95 % confidence intervals of all your expected value estimates, using the knowledge that the variance is 0.25 in all cases.

In [ ]:
```
# Write your code here.
```

Q2. Viewing the three confidence intervals, can you say that none, some or all of the underlying processes have zero-mean?

Use the knowledge (or assumption) that all the realizations actually come from the same process. Compute a more reliable estimate of the expected value of the process.

In [ ]:
```
# Write your code here.
```

Q3. What is your final estimated expected value of the white noise process?

1.2 Estimation of covariance function

Load the file data2 which contains a realization of an unknown process as the variable y. Estimate and plot the covariance function using `getCov(x,max_lag,"r")` !

In [ ]:
```
# Input code for loading 'data2.mat'
path_to_data = "data2.mat" # Change to YOUR path to data.
data = scipy.io.loadmat(path_to_data,simplify_cells=True)

max_lag = 10
r, lags = getCov(data["y"],max_lag,"r")

plt.figure(figsize=(8, 4))
plt.plot(lags,r, label='Auto-covariance')
plt.xlabel('lags')
plt.ylabel('Covariance')
plt.title('Covariance function')
```

```
plt.legend()
plt.show()
```

In a new figure, plot the correlation function using `getCov(x,max_lag,"rho")` ,

In [ ]:
```
rho, lags = getCov(data["y"],max_lag,"rho")
plt.figure(figsize=(8, 4))
plt.plot(lags,rho, label='Correlation')
plt.xlabel('lags')
plt.ylabel('Covariance')
plt.title('Covariance function')
plt.legend()
plt.show()
```

Verify the correspondence between the covariance and correlation functions by comparing the values in the two plotted curves.

Q4. What is the connection written as a formula?

In a new figure, plot all possible values of y(t) and y(t-k) against each other in a `plt.scatter` -plot for different values of k, e.g.

In [ ]:
```
k = 1 # sets the lag
plt.figure(figsize=(8, 4))
plt.scatter(data["y"][:-k],data["y"][k:], label='Realization 1')
plt.show()

# Write your own code below:
k = 2 #...
```

Compare with the resulting figures if you change to k=2 and k=3 (in new figures).

Q5. Explain how the view in the scatter plots relate to the values in the correlation function.

1.3 Estimation of Spectral Density

The function `signalsim()` simulates one realization of a Gaussian stationary harmonic process according to

$$x(n) = A_1 \cos(2\pi \frac{f_1}{fs} n + \phi_1) + A_2 \cos(2\pi \frac{f_2}{fs} n + \phi_2), \quad n = 0 \dots N-1,$$

with frequencies $f_k = \{10, 20\}$ Hz, number of samples $N = 500$ and sample frequency $fs = 256$ Hz. The independent phases are $\phi_k \in Rect(0, 2\pi)$ and the independent amplitudes are $A_k \in \text{Rayleigh}(\sigma_k)$ with parameters $\sigma_k = \{2, 2\}$.

Code and plot a simulated realization of the above process corresponding to a timescale in seconds, or run the function `signalsim` .

In [ ]:
```
# Use following to lines...:
x, t = signalsim()
# ... Or create your own method of simulating the process above:

# Visualize the realization
```

```
plt.plot(t, x)
plt.xlabel('Time (s)')
plt.title('Realization')
plt.show()
```

Q6. What is the connection between the maximum time-value in seconds, number of samples N and the sampling frequency fs in Hz?

Estimate and view the covariance function for an appropriate `max_lag` and plot the resulting function. Let the x-scale be lag-values expressed in seconds!

In [ ]:
```
# Write your own code below.
```

Q7. Do the periodicities of the realization and covariance function match each other?

If your answer is no, you have made a mistake in your calculation of the lag-scale (or the timescale)!

Estimate the spectral density with the periodogram, at $NFFT = 2048$ frequency values, and plot the resulting periodogram with f as the x-scale.

In [ ]:
```
fs = 256
nfft = 2048

f,P = scipy.signal.periodogram(x,fs=fs,nfft=nfft)

plt.plot(f,P)
plt.xlabel('frequency [Hz]')
plt.ylabel('Periodogram')
plt.show()
```

Q8. How does the frequency of the peaks relate to the periodicity of the realization (and covariance function)?

Study the peak heights of the two frequencies shown in the periodogram.

Q9. Why are the heights of the periodogram peaks different from each other?

Run your code or the `signalsim` again to view a new simulated realization. Compute and plot the periodogram. Repeat the procedure of simulating a new realization and plotting the periodogram a couple of times. Sometimes the resulting periodogram seems to contain just one frequency. Choose one such case and study the periodogram in a new figure by changing the y-scale to decibel (dB)

In [ ]:
```
plt.plot(f,10*np.log10(P))
plt.xlabel('frequency [Hz]')
plt.ylabel('10*log10(Power)')
plt.ylim([-60,np.max(10*np.log10(P))*1.1])
plt.show()
```

Compare the two different ways of visualizing the periodogram.

Q10. What is the advantage of using the dB-scale?

2 Instruments in a symphony orchestra

The sound from an acoustic instrument consists of a fundamental frequency, often termed a keynote, and some overtones or harmonics. The phases of the overtones typically depend on the instrument and are partly correlated with the oscillation of the keynote. This, together with the relation between the power of the overtones, produces the perceived sound of the instrument. If the keynote has frequency f_0, the frequencies of the overtones will depend on the type of instrument. For string instruments, the overtones can be well represented as $f_k = kf_0$, $k = 1, 2, \ldots$ The examples used in this exercise are recorded by the Philharmonia musicians, and are just a few examples of many, found at their webpage https://philharmonia.co.uk/resources/sound-samples/.

2.1 Keynotes and overtones

Download and load the file `cellodiffA.mat`, where you find three variables, `celloA2`, `celloA3` and `celloA4`, each representing a recording of the note A. The sampling frequency is $fs = 44100$ Hz. Listen to the tones by using the command `sd.play(data, fs)` Note: if the correct sampling frequency is not given as an input parameter, this will result in a strange sound!

Estimate the corresponding spectral densities using the periodogram with dB-scale and compare the keynote and overtones frequencies for the three cases!

Use `%matplotlib qt` and `%matplotlib inline` to switch showing plots in interactive windows and inline respectively (inline is default as you probably have noticed).

In [ ]:
```
%matplotlib qt

# Write your code below.
```

Q11. What are the keynotes and overtones in the three cases and how do the frequencies of the three different A relate to what you hear? Does the frequency structure match the simple overtone-model described above?

Cello and flute

In the previous exercise, the extraction of frequencies from the different sounds could be a winning strategy to differ or classify these sounds. If we instead listen to two different instruments playing exactly the same note A, it is much more difficult to differ them. Download the file `celloandflute.mat`, where you find two examples of the cello-tone A and two examples of the flute-tone A. Listen to the sounds!

Study the periodograms of all four cases!

In [ ]:
```
# Write your code below.
```

Q12. What differences do you see in the periodograms between the two instruments? Suggest some relevant features to be extracted from periodogram structure that could be used to differ the two instruments.

2.3 The spectrogram

Download the file cellomelody.m, where you find two examples of sequences played at the cello, `melody1` and `melody2`. Listen to both sequences. Can you hear the difference between them so you are able to

identify them?

As the sequences are now time-varying, we use the spectrogram to visualize a number of stacked periodograms. Choose a window length (also is the sequence length of a periodogram) as $window = 2048$, which is around 46 ms. Also choose the stacked periodograms to be calculated from somewhat overlapping data sequences, here the number of overlapping samples are chosen as $noverlap = 1024$. **The parameter $noverlap$ can be chosen from zero (no overlap) and as large as $window - 1$.** With no overlap the calculation time is reduced, but the resulting spectrogram time resolution will also be blurred. A large number of overlapping samples is computationally heavy although a nice view of the spectrogram is given. Finally, the number of frequency values must be at least equal to the window length, but is often much larger, e.g. here we choose $NFFT = 16384$. **Compute and plot the spectrograms with**

In [ ]:
```
# 1) Load data
# Write your own code below.

# 2) Set your parameters.
fs = data["fs"]
noverlap = 1024
nfft = 16384
window = 2048

# 3) Calculate Spectrogram
f, t, S = scipy.signal.spectrogram(melody1, fs=fs, noverlap=noverlap, window='hann', nperseg=winc
plt.figure()
plt.pcolormesh(t, f, 10*np.log10(S))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()

f, t, S = scipy.signal.spectrogram(melody2, fs=fs, noverlap=noverlap, window='hann', nperseg=winc
plt.figure()
plt.pcolormesh(t, f, 10*np.log10(S))
plt.ylabel('Frequency [Hz]')
plt.xlabel('Time [sec]')
plt.show()
```

The x-axis represents time in seconds and the y-axis is the frequencies in kHz. The colours of the image show the power according to the colour bar. Read documentation on `scipy.signal.spectrogram` for more information. Visualize both sequences and compare the images. Try to identify which spectrogram that belongs to `melody1` and `melody2` respectively just by listening and viewing an image. Zoom in for frequencies below 2 kHz for a better view of the keynotes! Use `%matplotlib qt` and `%matplotlib inline` to switch showing plots in interactive windows and inline (inline is default as you probably have noticed).

Q13. Can you differ the two sequences by listening and also identify differences of the two melodies in the spectrograms? Describe in words and point out the features in your images you rely on.

Change the input parameter, $window$ and also change the parameter $noverlap$, see description above. View the spectrograms using a much longer window and also a much shorter window.

In [ ]:
```
# Write your own code below.
```

Q14. Describe the differences from the initial setup above when you apply longer or shorter windows.

Another commonly used representation and visualization for audio signals is the mel spectrogram. The mel scale, named after the word melody, is a perceptual scale of pitch, where fundamental frequencies are judged by listeners to be equal in distance from one another. The mel scale was invented almost 100 years ago and is subjective to human perception. And yet, this is the overall most applied algorithm for representation of audio time-frequency images, when such images are used in machine learning classification. The main reason is the size reduction of the resulting so called mel spectrogram image. The x-axis has the same size as the spectrogram, but the y-axis will be represented with few values, the number of chosen mel filters, to be compared to the spectrogram where the size of the y-axis is found as $NFFT/2$. You can visualize the filterbank structure using 32 mel filters with

In [ ]:
```
h = melFil(fs,nfft)
```

The figure shows the shapes of the different mel filters in frequency, for frequencies up to $fs/2$. Each filter will be used to calculate the weighted average of powers in the spectrogram, according to the filter shape, which will be represented at the center frequency value of the filter. For low frequencies, the filters are keeping a better frequency resolution as the bandwidths of these filters are more narrow. For higher frequencies, the bandwidths of the filters are wider and thereby several adjacent frequencies are averaged. The frequency resolution therefore is worse for higher frequencies. The final mel spectrogram powers is calculated for all different time values. For more information, read documentation on $melSpectrogram$. A mel spectrogram, using 32 mel filters, can be calculated as

In [ ]:
```
melSpectrogram(S,fs,noverlap,nfft)
```

Q15. Compare the mel spectrogram images of melody1 and melody2 and judge if you still can decide which image that belongs to which strophe. Are there certain type of audio signals where the reduced frequency resolution of the mel spectrograms is disadvantageous in a classification task? Give examples.

2.4 Decimation and aliasing

Similar to the exercise above, where you reduced the view to zoom in on the keynote for a proper identification, a down-sampling or decimation is often useful in analysis and modelling of oscillating data. Decimate `melody1` by a factor $q = 8$ with

In [ ]:
```
# 1) Decimation
q = 8
melody2dec = scipy.signal.decimate(melody2,q)

# 2) Playback
# Write your own code below.
```

Listen to the decimated sequence. Can you hear any differences from the original data? (The differences should be small and possible differences will depend on the quality of the loudspeaker.) Make sure you keep track of the new input sampling frequency to `sd.play(x,fs)`, so it matches the actual sampling rate of the data!

Use `scipy.signal.spectrogram` to visualize the original data and the decimated data in different figures. A better view of the decimated sequence will be given with use of a shorter window, as there now are fewer samples in each tone. E.g. use a window length of 512 and an overlap of 256 samples. Compare

the spectrogram of the original and decimated data.

In [ ]:
```
# Write your own code below.
```

Q16. Are the keynote frequencies the same in the two images?

If the answer of this question is no, you should check the used sampling frequencies!

The `decimate` -command starts with a low-pass filtering of the data followed by the decimation by picking samples from the filtered data with an interval corresponding to the decimation factor. Use the following command to perform a decimation without the low-pass filter. Listen to this sequence and visualize with the spectrogram.

In [ ]:
```
# 1) Downsampling
melody2alias=melody2[0:-1:q]

#2) Playback
# Write your own code

#2) Spectrogram
# Write your own code
```

Q17. What are the differences between melody1dec and melody1alias when you listen to them? Explain the main differences you see in the two spectrograms.

This comparison aims to show how important it is to be careful in the filtering procedure before decimation of data.