University of South-Eastern Norway

Convolution and FIR-filtering
DRAFT - Details will be changed

TSE2280 Signal Processing

Lab 3, spring 2025

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background and Motivation

This lab demonstrates concepts from Chapters 4, 5, and 6 in the course text-book McClellan et al., *DSP First* [1], covering FIR-filters, convolution and frequency response. The lab will give training in the convolution operation, and how this is related to FIR-filters. The lab is based on *Lab 07: Sampling, Convolution, and FIR Filtering* that comes with the course textbook. The original lab has been modified and converted from Matlab to Python.

1.2 Software Tools: Python with Spyder and JupyterLab

Python is used for for programming, with Spyder [2] as the programming environment, and JupyterLab [3] for reporting.

The signals are represented as NumPy [4, 5] arrays and plotted in Matplotlib [6, 7]. This lab also introduces filter and convolution tools from the signal processing modules in SciPy [8, 9], `scipy.signal`, in addition to using the spectral analysis methods from the previous lab. The lab also includes demo programs using interactive JupyterLab widgets, modules for running these must be laoded into the Python environment. The recommended way to import the Python modules is shown in Table 1. You are free to do this in other ways, but the code examples assume modules are imported as described here.

New Modules

Two demo-programs shall be run as interactive JupyterLab files to illustrate central concepts. The *widgets* used to run these files in interactive mode require two extra modules in the Python environment,

`ipywidgets`	*Jupyter Widgets*	Interactive controls for JupyterLab
`ipympl`		Enables interactive features of Matplotlib in JupyterLab.

These modules must be added to the Python environment (Conda or Anaconda) before the demo programs can be run.

2 Theory with Programming Examples

2.1 Overview of Filtering

An FIR filter is a discrete-time system that converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \quad . \tag{1}$$

The equation gives the value of the n-th value of the output sequence $y[n]$ from the M previous values of the input sequence $x[n]$. See chapter 5 in the text-book [1] for details and examples.

Python offers several variants for performing filtering operations. These are found in the SciPy module under the subpackage `scipy.signal`. The two functions to use in this course are

1. `scipy.signal.convolve(x1, x2)`. Convolves the sequences `x1` and `x2`. The FIR-filter operation (1) is a convolution where `x1` is the input sequence $x[n]$ and `x2` the filter coefficients b_k.

2. `scipy.signal.lfilter(b, a, x)`. Filters the sequence `x` with forward coefficients `b` and backward coefficients `a`. A FIR-filter has only forward coefficients, the backward coefficients a are for IIR-filters. FIR-filtering is be done by letting `x` be the input sequence $x[n]$ and `b` the filter coefficients b_k, and setting `a` to 1.

3 Training Exercises

Reporting

Collect answers and code in a JupyterLab notebook. Export this to pdf and upload it to Canvas. You may prefer to do some of the coding in another development tool, such as Spyder. This is actually

Table 1. Recommended format for importing the Python modules. NumPy is used to manipulate signals as arrays, Matplotlib to plot results, and SciPy for signal processing.

```
import numpy as np                      # Handle signals as arrays
import matplotlib.pyplot as plt         # Show results as graphs and images
from math import pi, cos, sin, tan      # Mathematical functions on scalars
from cmath import exp, sqrt             # Complex mathematical functions on scalars

from scipy.fft import fft, fftshift, fftfreq    # FFT and helper functions
from scipy import signal                # Signal processing functions

import sounddevice as sd                # Play NumPy array as sound
```

recommend, as the testing and debugging options are better in Spyder than in JupyterLab. If you use Spyder, you can either copy your code from Spyder and paste it into a block in JupyterLab, or load the Python-files into JupyterLab ans they are. This can then be exported to a pdf-file.

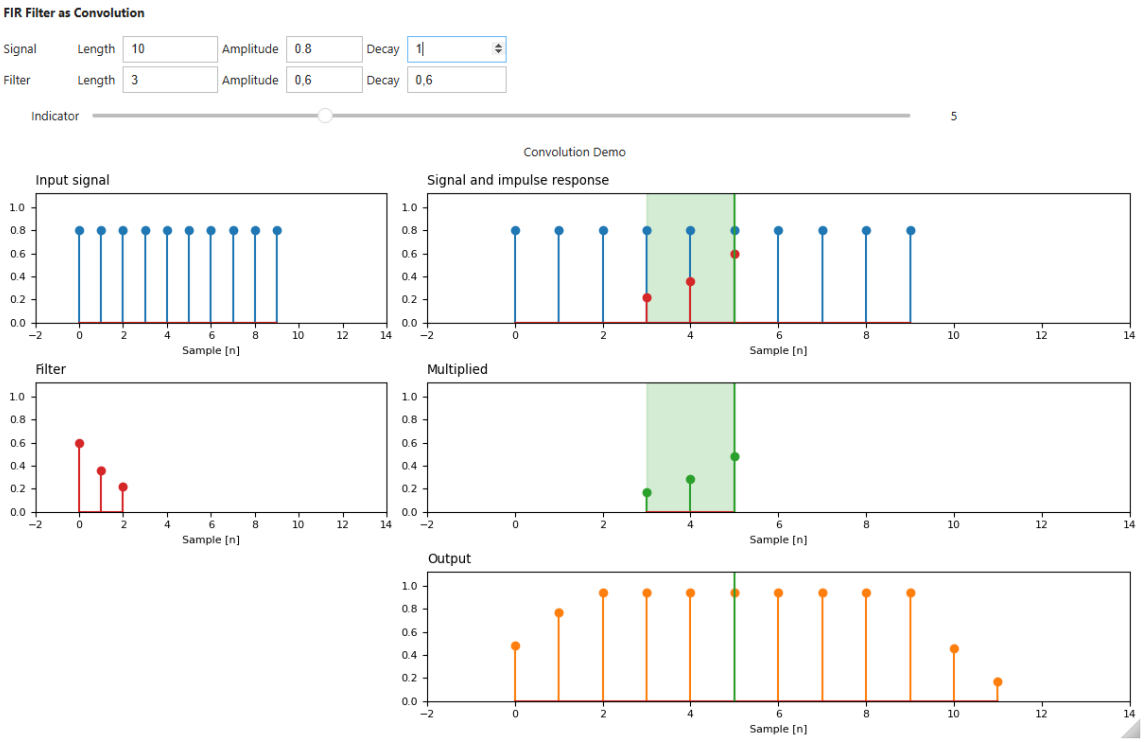

Figure 1. Screenshot of the JupyterLab interactive program `convolution_demo.ipynb`. The left graphs show the input signal $x[n]$ and the filter coefficients $h[k]$. The upper right graph shows $x[n]$ and the flipped filter response $h[n-k]$ for $n = 5$. The middle right graph shows the results of multiplying the signals in the upper graph, $x[n]h[n-k]$, for $n = 5$. The lower right graph shows the result $y[n] = x[n] * h[n]$ of the convolution between the two sequences. The value for $n = 5$ is highlighted, this is the sum of the samples in the multiplied sequence above. The signal parameters and sampling rate can be changed interactively by moving the slider controls.

3.1 Convolution Demo

The interactive JupyterLab-file `convolution_demo.ipynb` was made to illustrate the convolution operation. This is the basis for FIR-filtering, in addition to being an important concept occurring in many other phenomena. The program visualizes the convolution operation for a selected input signal $x[n]$ and timpulse response $h[n]$ by showing the 'flipping and shifting' operation when computing a convolution.

1) Set the input $x[n]$ to a finite-length pulse, $x[n] = (u[n] - u[n-10])$, where $u[n]$ is the unit step signal.

2) Set the filter to a three-point averager. Remember that the impulse response is identical to the filter coefficients b_k for an FIR filter.

3) Use `convolution_demo.ipynb` to produce the output signal.

4) Change the value of the current sample value n and observe the sliding window action of convolution.

3.2 Filtering via Convolution

The convolution operation is implemented in SciPy by the function `scipy.signal.convolve`, and can also be done using the filter function `scipy.signal.filter`.

1) Make a 3-point averager. The filter coefficient vector for this is

$$b = [\tfrac{1}{3}, \tfrac{1}{3}, \tfrac{1}{3}] \tag{2}$$

The this can be implemented as an array in NumPy by

```
b = 1/3 * np.ones(3)
```

Create a signal $x[n]$ as a pulse with length 10 samples,

$$x[n] = \begin{cases} 1 & , \quad n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 \\ 0 & , \quad \text{elsewhere} \quad . \end{cases} \tag{3}$$

The result of a convolution is longer than the two sequences being convolved. For this reason, the input pulse should be extended with extra zeroes at the end. This can be done by the NumPy `append` function,

```
x = np.append(np.ones(10),  np.zeros(5))
```

2) The filtering action can be illustrated with a plot of the input and output signals together. Since $x[n]$ and $y[n]$ are discrete-time signals, a the `stem`-plot should be used. The plots can be put together by using subplots,

```
plt.subplot(2, 1, 1)
plt.stem(n, x[n])
plt.xlabel("n")
plt.ylabel("x[n]")

# Output signal y[n]
y = signal.convolve(x, b)

plt.subplot(2, 1, 2)
plt.stem(n, y[n])
plt.set_xlabel("n")
plt.set_ylabel("y[n] = x[n]*h[n]")
```

Using this code, the plotting range for both signals will be equal to the length of the input signal, which was padded with extra zero samples.

3) The plot above is quite crowded, and it is more informative to show a small part of the signals. Repeat the previous part with first and last chosen to display 30 points from the middle of the signals.

4) Explain the filtering action of the 3-point averager by comparing the plots in the previous part. This may be called a smoothing filter. Note how the transitions in $x[n]$ from zero to one, and from one back to zero, have been smoothed.

3.3 Discrete-Time Convolution

In this section, you will generate filtering results needed in a later section. Use the discrete-time convolution demo, `convolution_demo.ipynb`, to do the following

1) Set the input signal to

$$x[n] = (0.9)^n(u[n] - u[n-10]).$$

Use the *Decay* parameter to create the signal.

2) Set the impulse response to

$$h[n] = \delta[n] - 0.9\delta[n-1]$$

Use again the *Decay* parameter to create the signal.

3) Illustrate the output signal $y[n]$ and explain why it is zero for almost all points. Compute the numerical value of the last point in $y[n]$, i.e., the one that is negative and non-zero.

3.4 Loading Data

The basic filtering function will now be studied with some real data. The data are stored as NumPy arrays in the file `lab_3_data.npz`, using NumPy's `.npz`-format. The data in the file are

`stair`	A stair-step signal. These are found in one sampled scan line from a TV test pattern image.
`scanline`	An actual scan line from a digital image.
`oak`	A speech waveform ("oak is strong") sampled at f_s=8000 samples/s.
`h1`	Coefficients for a FIR discrete-time filter of the form of (1).
`h2`	Coefficients for a second FIR filter.

Data are loaded into the workspace by the NumPy command `load`. This command will first load the contents into an *NpzFile class* variable, called `data` in the example below. The contents of this can then be addressed to extract the individual NumPy arrays into variables. The variables are often, but not necessarily, called the same as the field in the `npz`-file. An example on how to load the contents is shown below

```
data = np.load('lab_3_data.npz')
x1 = data['x1']
xtv = data['xtv']
```

After loading the data, check the *Variable Explorer* in Spyder to verify that all five vectors are in your workspace.

3.5 Filtering a Signal

You will now use the imported signal vector `x1` as the input to an FIR filter.

1) Do the filtering with a 5-point averager, using the 3-point averager (2) as template. Use the convolution function `scipy.signal.convolve` to filter `x1` with the 5-point averager.

   How long are the input and output signals?

2) Make a plot of the input signal and output signals together, as stem plots. Use the example code in Section 3.2 as template.

   Use the same scale on the time-axis for both the input and output signals.

3) Since the previous plot is quite crowded, it is useful to show a small part of the signals. Repeat the previous part, but display only 30 points selected the middle of the signals.

4) Explain the filtering action of the 5-point averager by comparing the plots from parts 2) and 3). This filter might be called smoothing filter. Note how the transitions from one level to another have been smoothed.

5) Make a sketch of what would happen with a 2-point averager.

3.6 Filtering Images: 2-D Convolution

One-dimensional FIR filters, such as running averagers and first-difference filters, can be applied to one-dimensional signals such as speech or music. These same filters can be applied to images if we regard each row or column of the image as a one-dimensional signal. For example, if $x[m, n]$ represents a 2 dimensional image, the 50th row of that image is the sequence $x[50, n]$. This can be filtered with a 1D filter using the `convolution` or `filter` functions in `scipy.signal`.

The next part of this lab will show how simple 2D filtering can be accomplished with 1D row and column filters. It might be tempting to use a for loop to write an M-file that would filter all the rows. This would create a new image made up of the filtered rows

$$y_1[m, n] = x[m, n] - x[m, n - 1] \tag{4}$$

However, this image $y_1[m, n]$ would only be filtered in the horizontal direction. Filtering the columns would require another for loop, and finally you would have the completely filtered image

$$y_2[m, n] = y1[m, n] - y_1[m - 1, n] \tag{5}$$

In this case, the image $y_2[m, n]$ has been filtered in both directions by a first-difference filter. These filtering operations involve a lot of convolution-calculations, so the process can be slow. Fortunately, SciPy's `convolve`-function can handle inputs arrays of higher dimensions, also 2-dimensional. The 2-dimensional filtering operation is more general than row/column filtering, but since it can do these simple 1-D operations it will be very helpful in this lab.

1) Load in the image `echart.mat` with the load command. This will create the variable `echart` with size 257×256.

   To filter the image in the horizontal direction using a first-difference filter, form a row vector of filter coefficients and use the following statements

```
b = [1, -1];
echart_fx = signal.convolve(echart, b);
```

   The filter coefficients `b` for the first-difference filter are stored in a vector and will cause `convolve`to filter all rows in the horizontal direction.

   Display the input image `echart` and the output image `echart_fx` on the screen at the same time. Compare the two images.

2) Filter the image `echart` in the vertical direction with the first-difference filter to produce the image `echart_fy`. Display the image `echart_fy` on the screen and describe how the output image compares to the input.

4 Lab Exercises: FIR Filters

The lab exercise will study how a filter can produce the following special effects

1) Echo: FIR filters can produce echoes and reverberations because the filtering formula (1) contains delay terms. In an image, such phenomena are called ghosts.

2) Deconvolution: One filter can approximately undo the effects of another. We will investigate a cascade of two FIR filters. The first filter distorts the image, the second attempts to restore it back to the original. This process is called deconvolution.

4.1 Deconvolution Experiment for 1D Filters

Use the function `filter` to implement the following FIR filter

$$w[n] = x[n] - 0.9x[n - 1] \tag{6}$$

on an input signal $x[n]$ defined by `= 256*(rem(0:100,50)<10)` The vector of filter coefficients for (6) must be defined in SciPy, the coefficient a ia always 1 for an FIR filter.

a) Plot the input and output waveforms $x[n]$ and $w[n]$ in the same figure using subplots and the stem-function. Restrict the horizontal axis to the range $0 \leq n \leq 75$. Explain why the output appears the way it does by figuring out the effect of the filter coefficients in (6).

b) Note that $w[n]$ and $x[n]$ are not the same length. Determine the length of the filtered signal $w[n]$ and explain how its length is related to the length of $x[n]$ and the length of the FIR filter.

4.2 Restoration Filter

The following FIR filter

$$y[n] = \sum_{l=0}^{M} r^l w[n-l] \tag{7}$$

can be use to undo the effects of the FIR filter in the previous section. It performs restoration, but it only does this approximately. Use the following steps to show how well it works when $r = 0.9$ and $M = 22$.

1) Process the signal $w[n]$ from (6) with the restoration filter (7) to obtain the output signal $y[n]$.

2) Make stem plots of $w[n]$ and $y[n]$ using the same time axis n for both signals.

3) The objective of the restoration filter is to produce a $y[n]$ that is almost identical to $x[n]$. Male a plot of the error, the difference between $x[n]$ and $y[n]$, over the range $0 \leq n < 50$.

4) Evaluate the worst-case error by using Python's `max` function to find the maximum difference between $y[n]$ and $x[n]$ in the range $0 \leq nlt50$.

5) What does the error plot and worst case error tell you about the quality of the restoration of $x[n]$? How small do you think the worst case error has to be to make it invisible on a plot?

4.3 An Echo Filter

The following FIR filter can be interpreted as an echo filter.

$$y_1[n] = x_1[n] + rx_1[n-P] \tag{8}$$

Explain why this is a valid interpretation by working out the following.

1) You have an audio signal sampled at $fs = 8000\,\text{Hz}$ and you would like to add a delayed version of the signal to simulate an echo. The time delay of the echo should be $0.2\,\text{s}$, and the strength of the echo should be $90\,\%$ of the original.

   Determine the values of r and P in (8), make P an integer.

2) Find the filter coefficients of this FIR filter and determine its length.

3) Implement the echo filter (8) with the values of r and P found above. Use the speech signal in the vector `oak`. Listen to the result to verify that you have produced an audible echo.

4.4 Cascading Two Systems

More complicated systems are often made up from simple building blocks. The system in Figure 2 constitutes two FIR filters connected in cascade. Assume that the the filters in Figure 2 are described by

$$w[n] = x[n] - qx[n-1] \qquad \text{FIR-filter 1} \tag{9a}$$

$$y[n] = \sum_{l=0}^{M} r^l w[n-l] \qquad \text{FIR-filter 2} \tag{9b}$$

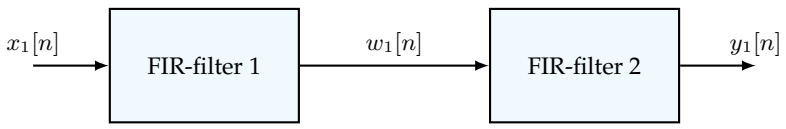

Figure 2. Cascade-coupled FIR-filters. The input signal $x[n]$ is processed in FIR-filter 1 resulting in the intermediate signal $w[n]$. This is then processed in FIR-filter 2 to produce the output signal $y[n]$. FIR-filter 2 can be selected to deconvolve the distortion introduced by FIR-filter 1, but note that a perfect deconvolution is not possible with a FIR-filter.

4.4.1 Overall Impulse Response

a) Implement the system in Figure 2 and get the impulse response of the overall cascaded system for the case where $q = 0.9$, $r = 0.9$ and $M = 22$. Plot the impulse response of the overall cascaded system.

b) Work out the impulse response $h[n]$ of the cascaded system by hand.

c) In a deconvolution application, the second system, FIR-filter 2, tries to undo the effect of the first. Perfect deconvolution would require the cascade combination of the two systems to be equivalent to the identity system, $y[n] = x[n]$.

   If the impulse responses of the two systems are $h_1[n]$ and $h_2[n]$, state the condition on $h_1[n] * h_2[n]$ to achieve perfect deconvolution.

   Note that the cascade of FIR-filter 1 and FIR-filter 2 does not perform perfect deconvolution.

4.4.2 Distorting and Restoring Images

The objective in this section is to show that we can use the second system, FIR filter 2, to undo the distortion from the first filter, FIR filter 1. Since FIR filter 2 will try to undo the convolutional effect of the first, it acts as a *deconvolution* operator.

a) Load in the image `echart`. This creates a matrix called echart.

b) Set $q = 0.9$ in FIR filter 1 and filter the image echart in both directions, i.e., along rows and columns. Apply FILTER-1 along the horizontal direction and then filter the resulting image along the vertical direction also with FILTER-1. call the result `ech90`.

c) Deconvolve `ech90` with FIR FILTER-2, choosing $M = 22$ and $r = 0.9$.

   Describe the visual appearance of the output, and explain its features. Explain why you see "ghosts" in the output image, and use previous calculations to determine how big the ghosts (or echoes) are, and where they are located. Evaluate the worst-case error in order to say how big the ghosts are relative to "black-white" transitions, which are 0 to 255.

d) Now try to deconvolve `ech90` with several different FIR filters for FIR filter 2. Set $r = 0.9$ and try several values for M such as 11, 22 and 33. Pick the best result and explain why it is the best.

   Describe the visual appearance of the output, and explain its features.

   Hint: You can use `convolution_demo.ipynb` to generate impulse responses of the cascaded systems

e) Gray-scale display has 256 levels. How large is the worst-case error from the previous part in terms of number of gray levels?

   Do this calculation for each of the three filters you tested.

   Think about the following question: "Can your eyes perceive a gray scale change of one level, i.e., one part in 256?" Include all images and plots for the previous two parts to support your discussions in the lab report.

References

[1] J. H. McClellan, R. Schafer, and M. Yoder, *DSP First*, 2nd ed. United Kingdom: Pearson Education Limited, 2016.

[2] P. Raybaut. (2024) Spyder. [Online]. Available: https://docs.spyder-ide.org/

[3] Project Jupyter. (2024) Jupyter Lab. [Online]. Available: https://docs.jupyter.org

[4] NumPy. (2024) NumPy ver. 2.2. [Online]. Available: https://numpy.org/doc/2.2

[5] C. R. Harris et al., "Array programming with NumPy," *Nature*, vol. 585, no. 7825, pp. 357–362, Sep. 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-2649-2

[6] Matplotlib. (2024) Matplotlib 3.10.0. [Online]. Available: https://matplotlib.org/stable

[7] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007. [Online]. Available: https://doi.org/10.1109/MCSE.2007.55

[8] SciPy. (2025) SciPy ver. 1.15. [Online]. Available: https://docs.scipy.org/doc/scipy

[9] P. Virtanen et al., "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020. [Online]. Available: https://doi.org/10.1038/s41592-019-0686-2