University of South-Eastern Norway

Convolution and FIR-filtering

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 Chapter 5 in the course text-book McClellan et al., *DSP First* [1], covering convolution and FIR-filters. The lab is based on *Lab 09: Sampling, Convolution, and FIR Filtering* [2] that comes with the course textbook. The original lab has been shortened and modified, and converted from Matlab to Python.

1.2 Software Tools: Python with Spyder and JupyterLab

Python is used for for programming, with Spyder [3] as the programming environment and JupyterLab [4] for reporting. The signals are represented as NumPy [5, 6] arrays and plotted in Matplotlib [7, 8]. This lab introduces filter and convolution tools from the signal processing modules in SciPy [9, 10], `scipy.signal`. The recommended way to import the necessary 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.

Modules for Interactive Operation

A demo-program is included as an interactive JupyterLab files to illustrate the convolution operation. The *widgets* used to run files in interactive mode require two extra modules in the Python environment,

`ipywidgets`	*Jupyter Widgets*, interactive browser controls for Jupyter Notebooks
`ipympl`	Enables interactive features (zooming, cursors etc.) of Matplotlib in Jupyter Notebooks and other tools.

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

Table 1. Recommended format for importing the Python modules nedded in Lab 3.

```
import numpy as np                  # Handle signals as arrays
import matplotlib.pyplot as plt     # Show results as graphs and images
from scipy import signal            # Signal processing functions
import sounddevice as sd            # Play NumPy array as sound
```

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 the weighted summation

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

The equation calculates the value of the n-th sample in 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 FIR filtering operations. These are found in the SciPy module under the subpackage `scipy.signal`. The two functions used 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`. An FIR-filter has only forward coefficients b_k, the backward coefficients a_k are used by IIR-filters. FIR-filtering can be done by this function by letting `x` be the input sequence $x[n]$, `b` is an array of the filter coefficients b_k, and `a`=1.

2.2 Convolution Demo

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

Use the JupyterLab convolution demo program `convolution_demo.ipynb` to test the following filter operations.

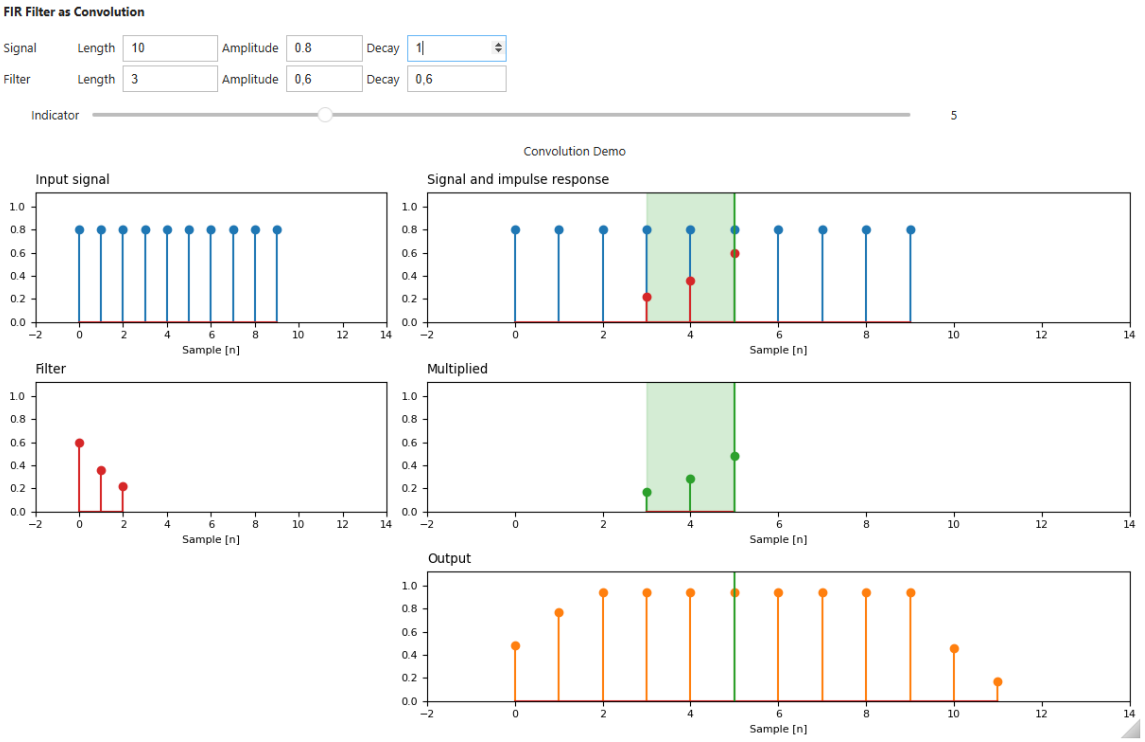

Figure 1. Screenshot of the JupyterLab interactive program `convolution_demo.ipynb`. The left graphs show the input signal $x[n]$ and the filter coefficients $h[k]$. The upper right graph shows $x[n]$ and the flipped filter response $h[n-k]$ for $n = 5$. The middle right graph shows the results of multiplying 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.

3-Point Averager

a) 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.

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

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

d) Use the slider to change the value of the current sample value n, and observe the sliding window action of convolution.

Cancellation

a) Set the input signal to

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

The *Decay* parameter r creates the sequence r^n. Use this to create the signal.

b) Set the impulse response to

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

Use again the *Decay* parameter r to create the signal, this can also be negative.

c) Explain why the output signal $y[n]$ 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.

2.3 Stored Data Sets

The lab exercises will apply filters on recorded data from a file. The data are stored as NumPy arrays in the file `lab3_data.npz`, using NumPy's `.npz`-format. The data in the file are

`x1`	A stair-step signal. These are found in one sampled scan line from a TV test pattern image.
`x2`	A scan line from a digital image.
`x3`	A speech waveform ("Oak is strong") sampled at f_s=8000 samples/s.
`h1`	Coefficients for a FIR discrete-time filter.
`h2`	Coefficients for a second FIR filter.
`echart`	A 2D black and white image displaying letters

Data are loaded into the workspace by the NumPy command `load`, which loads the contents into an *NpzFile class* variable, called `data` in the example below. The contents of this variable can then be assigned to individual NumPy arrays, often with the same names as the fields in the `npz`-file. An example on how to load the contents of an `npz`-file is shown below.

```
data = np.load('lab3_data.npz')
x1 = data['x1']   # Assign data from field 'x1' to variable x1
x2 = data['x2']   # Assign data from field 'x2' to variable x2
```

3 Lab 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, as this has better testing and debugging options. From Spyder, you can either copy your code and paste it into a block in JupyterLab, or load the Python-files into JupyterLab as they are. This can then be exported to a pdf-file.

3.1 FIR Filtering via Convolution. *Running Average*

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

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

Create a signal $x[n]$ as a pulse with length 8 samples,

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

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.

The two vectors above can be created in NumPy as shown below. The `append` function is used to add extra zeros at the end of $x[n]$. The vector n with integers is created for use as x-axis, starting at $n = 0$ and with the same length as x.

```
b = 1/3 * np.ones(3)
x = np.append(np.ones(8),  np.zeros(5))
n = np.arange(len(x))
```

Generate the signal and filter from the the code above. Use stem-plots to verify that the two sequences are as specified.

b) The FIR filtering operation can be done by a convolution, which is done by one command in SciPy, shown below.

```
y = signal.convolve(x, b)
```

This line creates the vector $y[n]$ as the the convolution of the input $x[n]$ with the filter $b[n]$.

Run this code and show the result by plotting the input $x[n]$ and output $y[n]$ as stem-plots in two subplots.

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

d) Import the signal vector `x1` from the file `lab3_data.npz` and show its value as a graph. A conventional `plot` is probably better than `stem` in this case.

e) Filter `x1` with a 5-point averager and plot the filtered result in the same graph as the original signal.

How long are the input and output signals?

f) To get a better view of the details, repeat the previous part, but display only 30 points selected from the middle of the signals.

g) Explain the filtering action of the 5-point averager by comparing the input signal $x[n]$ and the filtered signal $y[n]$.

This filter might be called smoothing filter. Note how the transitions from one level to another have been smoothed.

h) Repeat the previous taskes with 2-point averager. Explain the result.

3.2 Filtering Images. 2D Convolution

One-dimensional FIR filters can be applied to one-dimensional signals such as speech or music. Filtering can also be done in higher dimensions, such as 2D images. 2D filtering could be done by running through the rows in the image one by one, and then repeat the process along the columns. However, this would involve a lot of convolution-calculations, making the the process slow. A better way to do filter a 2D image is to make use of that SciPy's `convolve`-function can handle input arrays of higher dimensions.

The tasks below requires zooming into details in the figures, remember to add the command `%matplotlib ipympl` to your code.

Tasks

a) Import the 2D image `echart` from `lab_3_data.npz`. Display the image using `imshow` with the colormap `grey`. You are free to use a different colormap, but interpretation is easiest if shown in greyscale.

b) Filter the 2D image `echart.mat` with coefficients b_x defined as a row vector,

$$b_x = \begin{bmatrix} 1 & -1 \end{bmatrix} \tag{4}$$

The code to make a two-dimensional row vector `bx` is shown below. Note the double brackets to make it 2D.

```
bx = [[1, -1]]
```

These coefficients will cause `convolve` to filter along all rows in a 2D image.

Use this to filter `echart` along the horizontal direction. Display the filtered image on the screen and describe how the output image compares to the input.

The action of the filter is most easily understood if you zoom in the letter 'E'.

c) The filter coefficients `b` can also be defined as a column vector,

$$b_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix} \tag{5}$$

The code to create this set of coefficients is

```
by = [[1], [-1]]
```

These coefficients will cause `convolve` to filter along all columns in a 2D image.

Filter `echart` with these coefficients, display the result on the screen, and describe how the output image compares to the input.

d) A 2D matrix of filter coefficients will filter the image in both dimensions in one operation. i.e., along both the rows and columns. An example of this is

$$b_{xy} = \begin{bmatrix} 1 & -1/2 \\ -1/2 & 0 \end{bmatrix} \tag{6}$$

The code to implement b_{xy} for use by SciPy is

```
bxy = [[1, -1/2],
       [-1/2, 0]]
```

The line shift is not necessary, but can make the code easier to read.

Filter `echart` with these coefficients, display the result on the screen, and describe how the output image compares to the input.

e) The three versions of filter coefficients above can be viewed as *edge detection filters*. Compare the filtered images and the values of the filter coefficients, and explain how this can be used to find edges on an image.

3.3 Deconvolution

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

Filtering

a) Create a signal $x[n]$ by the following code

```
n = np.arange(100)
x = (n % 50) < 10
```

In Python, `%` is the modulus operator, also called the remainder operator. It gives the remainder after integer division. Plot `x` as a `stem`-plot and explain why the resulting curve looks as it does.

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

$$w[n] = x[n] - 0.9x[n-1] \quad . \tag{7}$$

Find the filter coefficients b_k for the filter defined by (7) and apply this filter on the signal $x[n]$.

c) 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 (7).

d) 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.

Restoration by devonvolution

The following FIR filter

$$y[n] = \sum_{l=0}^{M} r^l w[n-l] \tag{8}$$

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 $M = 22$.

a) Find the filter coefficients b_k for the filter in (8).

b) Process the signal $w[n]$ from (7) with restoration filter (8) to obtain the output signal $y[n]$.

c) Make stem plots of the original signal $x[n]$ and the restored signal $y[n]$ using the same time axis n for both signals.

d) 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 $y[n]$ and $x[n]$.

e) Evaluate the worst-case error by using Python's `max` function to find the maximum absolute difference between $y[n]$ and $x[n]$.

f) Increase the filter length to $M = 44$ and repeat the tasks above. How does this change the error plot and worst case error?

3.4 Echo Filter

FIR filters can produce echoes and reverberations because the filtering formula (1) contains delay terms. In an image, such phenomena are called ghosts. The following FIR filter can be interpreted as an echo filter.

$$y_1[n] = x_1[n] + rx_1[n-P] \tag{9}$$

a) You have an audio signal sampled at $fs = 8000\,\text{Hz}$ and would like to add a delayed version of the signal to simulate an echo. The delay of the echo should be $0.2\,\text{s}$, and the strength of the echo should be $70\,\%$ of the original.

Determine the values of r and P in (9), make P an integer.

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

c) Implement the echo filter (9) with the values of r and P found above. Use the speech signal in the vector `x3` (*Oak is strong ...*) loaded from the file `lab3_data.npz`.

Plot the original signal and the signal with the echo in two subplots.

Listen to the result to verify that you have produced an audible echo.

References

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

[2] ——, "Lab P-9: Sampling, Convolution, and FIR Filtering," 2016. [Online]. Available: https://dspfirst.gatech.edu/database/?d=labs

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

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

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

[6] 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

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

[8] 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

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

[10] 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