University of South-Eastern Norway

Sampling, FIR-filtering, and a touch-tone telephone

TSE2280 Signal Processing

Lab 3, spring 2025

1 Introduction

1.1 Background and Motivation

This lab will give training in the convolution operation, and how this is related to FIR-filters. The last part of the lab will apply this to crate a system to encode and decode the signals used in touch-tone telephones.

This lab is based on two labs that that accompany the course text-book *DSP First* by McClellan et al. [1], *Lab 07: Sampling, Convolution, and FIR Filtering* and *Lab 09: Encoding and Decoding Touch-Tone (DTMF) Signals*[2]. The lab demonstrates concepts from Chapters 5 and 6 in the text-book, covering FIR-filters, convolution and frequency response covering spectra, spectrograms, and aliasing. The original labs have been modified and converted from Matlab to Python.

Listening to the sounds of a signals is necessary in this lab, and can be done using the sound card in the computer. The generated sounds can be used to automatically dial numbers using a touch-tone telephone.

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] arrays and plotted in Matplotlib [6]. This lab also introduces the filter and convolution tools from the signal processing modules in SciPy [7], `scipy.signal`, in addition to using the spectral analysis methods from the previous lab, most important are the spectrogram representation.

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 in this text assumes modules are imported as described here.

2 Theory with Programming Examples

2.1 Overview of Filtering

For this lab, we will define an FIR filter as a discrete-time system that converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation,

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

This equation givesd 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 course text-book for details and examples.

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

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
```

1. `scipy.signal.convolve(x1, x2)`. Convolves the two sequences `x1` and `x2`. Convolution is the same as the FIR-filter operation in (1).

   FIR-filtering is done by letting `x1` be 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 a-coefficients are for IIR-filters.

   FIR-filtering is be done by letting `x` be the input sequence $x[n]$, `b` the filter coefficients b_k, and setting `a` to 1.

3 Training Exercises

Reporting

Two examples have been supplied as interactive JupyterLab files to illustrate some important concepts.

3.1 Sampling and Aliasing Demo

The interactive JupyterLab-file `sampling-aliasing-demo.ipynb` shows the result of sampling and reconstruction of a single frequency sinusoidal signal, where the frequency and the sampling rate can both be changed. The graphs show the sampled signal, $x[n]$ and its spectrum and also the reconstructed output signal, $y(t)$ with its spectrum.

1) Set the input signal to $x(t) = cos(40\pi t)$.

2) Set the sample rate to $f_s = 24\,\text{samples/s}$.

3) Determine the locations of the spectrum lines for the discrete-time signal, $x[n]$, found in the middle panels. Click the Radian button to change the axis to from linear frequency f to angular frequency ω.

4) Determine the formula for the output signal, $y(t)$ shown in the right panels. What is the output frequency in Hz?

3.2 Convolution Demo

The interactive JupyterLab-file `convolution-demo.ipynb}` allows you to select an input signal $x[n]$ and the impulse response of the filter $h[n]$. The demo then shows the "flipping and shifting" used when a convolution is computed. This is the same as the sliding window of the FIR filter, a FIR-filter is a convolution between the input signal and the filter coefficients.

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

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. Also, the interactive JupyterLab-file allows you to modify the length and values of the pulse.

3) Use the JupyterLab-file to produce the output signal.

4) Change the value of n, which represents the current sample value. This allows you to observe the sliding window action of convolution.

3.3 Filtering via Convolution

You can perform the same convolution as done by the `convolution-demo.ipynb}` by using the SciPy function `scipy.signal.convolve` or the filter function `scipy.signal.filter`.

1) Do the filtering with a 3-point averager. The filter coefficient vector for the 3-point averager 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 length-10 pulse,

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

Remember that 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 to see the effect of the convolution operation. An example on how this can be done in NumPy is

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

2) To illustrate the filtering action of the 3-point averager, it is informative to make a plot of the input signal and output signals together. Since $x[n]$ and $y[n]$ are discrete-time signals, a `stem` plot is needed. One way to put the plots together is to use subplots to make a two-panel display:

```
# Input signal x[n]

x = np.append(np.ones(10), np.zeros(5))
n = np.arange(n_start, n_end)   #

fig = plt.figure()

ax1 = fig.add_subplot(2, 1, 1)
ax1.stem(n, x[n])
ax1.set_xlabel("n")
ax1.set_ylabel("x[n]")

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

ax2 = fig.add_subplot(2, 1, 2)
ax2.stem(n, y[n])
ax2.set_xlabel("n")
ax2.set_ylabel("y[n] = x[n]*h[n]")
```

The code above assumes that the output from the filter operation is called `y`. Try the code with `n_start` equal to the beginning index of the input signal, and `n_end` set to the last index of the input. In other words,the plotting range for both signals will be equal to the length of the input signal (which was "padded" with extra zero samples). In other words, the plotting range for both signals will be equal to the length of the input signal, even though the output signal is longer.

3) Since the previous plot is quite crowded, it is useful 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 filter might 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.

3.4 Sampling and Aliasing

Use the JupyterLab-file `sampling-aliasing-demo.ipynb` to do the following problem

1) Input frequency is 12 Hz.
2) Sampling frequency is 15 Hz.
3) Determine the frequency of the reconstructed output signal.
4) Determine the locations in angular frequenc ω of the lines in the spectrum of the discrete-time signal. Give numerical values.
5) Change the sampling frequency to 12 Hz, and explain the appearance of the output signal.

3.5 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 "Exponential" signal type to create the signal.

2) Set the impulse response to

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

   Use again the "Exponential" signal type 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.

3.6 Loading Data

In order to exercise the basic filtering function , we will use 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 such as one might find 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 fs = 8000 samples/second.
`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. This is 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.

3.7 Filtering a Signal

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

1) First, do the filtering with a 5-point averager. The filter coefficient vector for this are as for the 3-point averager in (2) with 3 replaced by 5. Use the convolution function `scipy.signal.convolve` to filter `x1` to the output `y1`.

   How long are the input and output signals?

2) To illustrate the filtering action of the 5-point averager, you must make a plot of the input signal and output signal together. Since `x1[n]` and `y1[n]` are discrete-time signals, a stem plot is needed. An example of Python code to plot `x1[n]` and `y1[n]` in the same figure was given for the 3-point averager in Section 3.3.

   Plot the result starting at the beginning index of the input signal, and ending at the last index of the input. In other words, the plotting range for both signals will be equal to the length of the input signal, even though the output signal is longer.

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 a "smoothing" filter. Note how the transitions from one level to another have been "smoothed." Make a sketch of what would happen with a 2-point averager.

3.8 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, the 50th row of an image is the N-point sequence $xx[50, n]$ for $0 \leq n \leq N$, so we can filter this sequence with a 1-D filter using the `convolution` or `filter` functions in `scipy.signal`.

One objective of this lab is to show how simple 2-D filtering can be accomplished with 1-D 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) (a) Load in the image echart.mat with the load command (it will create the variable `echart` function size is 257 × 256). We can filter all the rows of the image at once with the `convlove` function. To filter the image in the horizontal direction using a first-difference filter, we form a row vector of filter coefficients and use the following statements

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

   In other words, the filter coefficients `b` for the first-difference filter are stored in a row vector and will cause `convolve`to filter all rows in the horizontal direction. Display the input image `echart` and the output image `echart` on the screen at the same time. Compare the two images and give a qualitative description of what you see.

2) Now filter the "eye-chart" image echart in the vertical direction with a first-difference filter to produce the image `echart_fy`. This is done by calling with a column vector of filter coefficients, as shown in the code below

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

   Display the image `echart_fy` on the screen and describe in words how the output image compares to the input.

4 Lab Exercises: FIR Filters

In the following sections we 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 would be called "ghosts."

2) Deconvolution: one FIR filter can (approximately) undo the effects of another—we will investigate a cascade of two FIR filters that distort and then restore an image. This process is called deconvolution.

4.1 Deconvolution Experiment for 1-D Filters

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

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

on the input signal $x[n]$ defined via the statement: xx = 256*(rem(0:100,50)¡10); In SciPy must define the vector of filter coefficients b needed in `filter`. The coefficient a ia always 1 for an FIR filter.

a) Plot both the input and output waveforms x[n] and w[n] on the same figure, using subplot. Make the discrete-time signal plots with MATLAB's stem function, but restrict the horizontal axis to the range $0 \leq n \leq 75$. Explain why the output appears the way it does by figuring out (mathematically) the effect of the filter coefficients in (3).

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. (If you need a hint refer to Section 1.2.)

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 (see the block diagram in Fig. 3). 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 (3) with FILTER-2 to obtain the output signal y[n].
2. Make stem plots of w[n] and y[n] using a time-index axis n that is the same for both signals. Put the stem plots in the same window for comparison—using a two-panel subplot.
3. Since the objective of the restoration filter is to produce a y[n] that is almost identical to x[n], make a plot of the error (difference) between x[n] and y[n] over the range $0 \leq nlt50$.

4.2.1 Worst-Case Error

a) Evaluate the worst-case error by doing the following: use MATLAB's max() function to find the maximum of the difference between y[n] and x[n] in the range $0 \leq nlt50$.

b) 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 so that it cannot be seen on a plot?

4.2.2 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:

a) You have an audio signal sampled at fs = 8000 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 seconds, and the strength of the echo should be 90% of the original. Determine the values of r and P in (4); make P an integer.

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

c) Implement the echo filter in (4) with the values of r and P determined in part (a). Use the speech signal in the vector x2 found in the file labdat.mat. Listen to the result to verify that you have produced an audible echo.

4.3 Cascading Two Systems

More complicated systems are often made up from simple building blocks. In the system of Fig. 3 two FIR filters are connected "in cascade." For this section, assume that the the filters in Fig. 3 are described by the two equations

$$w[n] = x[n] - qx[n-1] \qquad \text{FIR filter 2} \tag{9a}$$

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

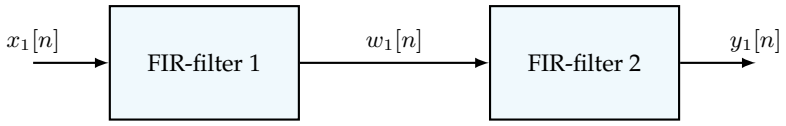

Figure 1. Cascade-coupled FIR-filters. The input signal $x[n]$ is processed in FIR-filter 1 resulting in the intermediate signal $w[n]$. This is then processed in FIR-filter 2 to produce the output signal $y[n]$. FIR-filter 2 can be selected to deconvolve the distortion introduced by FIR-filter 1, but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

4.3.1 Overall Impulse Response

a) Implement the system in Fig. 3 using MATLAB to get the impulse response of the overall cascaded system for the case where q = 0.9, r = 0.9 and M = 22. Use two calls to firfilt(). Plot the impulse response of the overall cascaded system.

b) Work out the impulse response h[n] of the cascaded system by hand to verify that your MATLAB result in part (a) is correct. (Hint: consult old Homework problems.)

c) In a deconvolution application, the second system (FIR FILTER-2) tries to undo the convolutional effect of the first. Perfect deconvolution would require that the cascade combination of the two systems be equivalent to the identity system: y[n] = x[n]. If the impulse responses of the two systems are h1[n] and h2[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.

4.3.2 Distorting and Restoring Images

If we pick q to be a little less than 1.0, then the first system (FIR filter 1) will cause distortion when applied to the rows and columns of an image. The objective in this section is to show that we can use the second system (FIR filter) to undo this distortion (more or less). 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.mat with the load command. It creates a matrix called echart.

b) Pick q = 0.9 in FILTER-1 and filter the image echart in both directions: apply FILTER-1 along the horizontal direction and then filter the resulting image along the vertical direction also with FILTER-1. all 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 by invoking your mathematical understanding of the cascade filtering process. Explain why you see "ghosts" in the output image, and use some 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.

4.3.3 A Second Restoration Experiment

a) Now try to deconvolve ech90 with several different FIR filters for FILTER-2. You should 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 by invoking your mathematical understanding of the cascade filtering process.

Hint 1: Determine the impulse response of the cascaded system and relate it to the visual appearance of the output image.

Hint 2: You can use dconvdemo to generate the impulse responses of the cascaded systems, like you did in the Warm-up.

b) Furthermore, when you consider that a 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 in part (a). 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.

4.4 Filtering a Music Waveform

Echoes and reverberation can be done by adding a delayed version of the original signal to itself. For this part, use the first 5 seconds of your synthesized song from Lab 4. In this experiment you will have to design FIR filters to process the music signal.

a) In order to produce an echo that is audible, the delay time has to be fairly long compared to the sampling period. A delay of one sample at fs = 11025 Hz is about Ts = 1/fs = 90.7 µsec. Instead, you need a delay of about 0.15 sec. for perception by the human hearing system. Determine the delay P needed in the following filter:

$$y[n] = \frac{1}{1+\alpha} w[n] + \frac{\alpha}{1+\alpha} w[n-P] \tag{10}$$

to produce an echo at 0.15 sec. The quantity α will control the strength of the echo—set it equal to 0.95 for this implementation. Then define the filter coefficients to implement the FIR filter in (5).

b) Filter the music signal with filter defined in part (a). Describe the sound that you hear and use the impulse response to explain why it sounds that way.

c) Reverberation requires multiple echoes. This can be accomplished by cascading several systems of the form (5). Use the parameters determined in part (a), and derive (by hand) the impulse response of a reverb system produced by cascading four "single echo" systems. Refer back to section 3.2 on cascading filters. Recall that two filters are said to be "in cascade" if the output of the first filter is used as the input to the second filter, and the output of the second filter is defined to be the output of the overall cascade system. This can be repeated for as many filters as are needed in the cascade system.

d) Filter the music signal with filter defined in part (c). Describe the sound that you hear and use the impulse response to explain why it sounds that way.

e) It will be difficult to make plots to show the echo and reverberation, but you should be able to do it with the M-file inout( ) which can plot two very long signals together on the same plot. It formats the plot so that the input signal occupies the first, third, and fifth lines, etc. while the output inout.m signal is on the second, fourth, and sixth lines etc. Type help inout to find out more. You should plot about 0.5 sec of the original and each processed music signal to show the delay effects that are producing the echo(es). Pick a segment containing only a few notes so that you can see the delayed

signals. Label the plots to point out the differences between the original and the echoed/reverb signals.

Note: it will be tricky to illustrate the effect that you want to explain, but you have to find a way to see the delayed versions of the original.

References

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

[2] J. H. McClellan, R. Schafer, and M. Yoder, "Lab P-3: Introduction to Complex Exponentials - Direction Finding," tech. rep., 2016.

[3] P. Raybaut, "Spyder," 2024.

[4] {Project Jupyter}, "Jupyter Lab."

[5] NumPy, "NumPy."

[6] Matplotlib, "Matplotlib."

[7] SciPy, "SciPy."