University of South-Eastern Norway

Touch-tone telephone system (DTMF)

TSE2280 Signal Processing

Lab 4, spring 2025

1 Introduction

This lab introduces a practical application where sinusoidal signals are used to transmit information, showing how bandpass FIR filters can be used to extract information encoded in the waveforms. The task is to generate a system to encode and decode signals used in touch-tone telephones. The lab demonstrates concepts from Chapters 6 and 9 in the textbook by McClellan et al [1], covering FIR-filters and frequency response. The lab is based on *Lab 09: Encoding and Decoding Touch-Tone (DTMF) Signals* [2, **?**] that comes with the textbook. [1]. The original lab has been modified and converted from Matlab to Python.

1.1 Background: Telephone Touch Tone Dialing

Telephone touch-tone pads generate *dual tone multiple frequency* (DTMF) signals to dial a number [3]. When a key is pressed, two tones are generated and summed. The frequencies of these two sinusoids represent one row and one column on the dial pad, as shown in Table 1. As an example, pressing the ′5′ key generates a signal containing the sum of the two tones at $770\,\mathrm{Hz}$ and $1336\,\mathrm{Hz}$ together.

The frequencies in Table 1 were chosen to miniize the chances of error from distortion in the telephone lines: Harmonics are avoided, meaning no frequency is an integer multiple of another, the difference between any two frequencies does not equal any of the frequencies, and the sum of any two frequencies does not equal any of the frequencies. To get more information, search for “DTMF” on the Internet.

1.2 DTMF Decoding

When the telephone central receives a DTMF-signal, the tones must be decoded back to a telephone number. This can be done by the following steps

1) Split the received time signal into shorter time segments representing individual key presses.

2) Filter the individual segments to identify frequency components. This can be done by a filter bank of bandpass filters at the DTMF frequencies listed in Table 1.

3) Determine which two frequency components are present in each time segment. This can be done by comparing the amplitudes of the output signal from the bandpass filters.

4) Determine which key was pressed, 0–9, A–D, *, or # by converting the row and column frequencies into key names using Table 1.

The core of the decoding system is a filter bank for identifying the frequencies in the received signal. This simple FIR filters and is illustrated in Figure 1. This filter bank consists of eight bandpass filters, where each pass only one of the eight possible DTMF frequencies. The input signal $x[n]$ for all the filters is the same DTMF signal The amplitudes of the output signals $y_k[n]$ will vary depending on which frequencies are present in the input $x[n]$.

Table 1. Extended DTMF encoding table for Touch Tone dialing. When any key is pressed the tones of the corresponding column and row are generated and summed. Keys A-D (in the fourth column) are not implemented on commercial and household telephone sets, but are used in some military and other signaling applications.

Frequencies [Hz]	1209	1336	1477	1633
697	1	2	3	A
770	4	5	6	B
852	7	8	9	C
941	*	0	#	D

Table 2. 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 scipy.signal as signal       # Signal processing functions
import matplotlib.pyplot as plt     # Show results as graphs and images
from math import pi, sqrt           # Mathematical functions on scalars
from cmath import exp               # Complex mathematical functions on scalars
import sounddevice as sd            # Play NumPy array as sound
```

2 Programming Tips and Exercises

2.1 Software Tools: Python with Spyder and JupyterLab

This lab uses the same tools as the previous labs.

Python is used for for programming, with Spyder [4] as the programming environment and JupyterLab [5] for reporting. The signals are represented as NumPy [6, 7] arrays and plotted in Matplotlib [8, 9]. The signals are analysed with functions from the signal processing modules in SciPy [10, 11].

Decoding of the tones is done using FIR filters, and the convolution and frequency analysis tools from the signal processing module in SciPy are central in this lab, `scipy.signal`.

You will present the tones as spectrograms, and play them from the computer's sound card

The recommended way to import the Python modules is shown in Table 2. You are free to do this in other ways, but the code examples in this text assumes modules are imported as described here.

2.2 Joining Signals

The DTMF tones are generated by joining NumPy arrays representing the individual DTMF tones and the intervals between them. This joining operation can be solved in several ways, e.g., by using the NumPy functions `concatenate` and `append`. The perhaps simplest method to create a vector `s` from individual tones `x[k]` and intervals `interval` is using NumPy function `append`, as sketched below.

```
s = []  # Define an empty array
for k in range(no_of_tones)
    s = np.append(s, x)         # Add an array x to the end of s
    s = np.append(s, interval)  # Add an interval to separate the tones
```

Be aware that the append method described here is inefficient, as it allocates a new section of memory every time the signal is appended and copies the new array into this section. This is no problem for the short sequences in this lab, but can slow down the program for very long signals A more efficient method would be to pre-allocate the complete signal from the beginning, and then put the new. parts into the correct section of this array. However, this can only be done if the final signal length is known from the start.

2.3 Simple Bandpass Filter Design

The L-point averaging filter is a *lowpass filter*, as shown in Chapter 6 in the textbook *DSP First* [1]. Its passband width is inversely proportional to filter length L. This filter can be converted *bandpass* *filter* with similar characteristics by designing an L-point FIR filter where the impulse response $h[k]$ is a cosine-function with frequency $\hat{\omega}_c$,

$$h[n] = \beta \cos(\hat{\omega}_c n) \,, \tag{1}$$

where $\hat{\omega}_c$ is the centre frequency of the filter. β is a scaling constant that defines the gain of the filter. This can be set so that the the maximum value of the frequency response magnitude is one, i.e., the filter gain is one at the center frequency $\hat{\omega}_c$. The bandwidth of the bandpass filter is controlled by L, the larger the value of L, the narrower the bandwidth. This filter is also discussed in Chapter 6 of the course textbook [1]. An example of magnitude response with passband and stopband for this type of bandpass filters is illustrated in Figure 2.

Excercise: Design a Bandpass Filter

The DTMF decoding is done by a bank of bandpass filters centered at the DTMF tone frequencies. This exercise will will show how to design and analyse this filter bank.

a) Use (1) to generate a bandpass filter centered at $\hat{\omega}_c = 0.2\pi$. Make the filter length L=51.

b) SciPy has a function that calculates the frequency response of a filter, this is called `freqz`. The call

```
    w, Hw = signal.freqz(h)
```

will return the frequency response $H\left(e^{j\hat{\omega}}\right)$ in `Hw` and the normalised frequency in `w`, when `h` is the filter impulse response $h[n]$.

Set $\beta = 1$ and use `freqz` to find the frequency response of your filter. Plot the magnitude and phase of the frequency response $|H\left(e^{j\hat{\omega}}\right)|$ as function of frequency $\hat{\omega}$.

c) Find the value of β so that the maximum value of the frequency response magnitude is one.

This can be done numerically by reading maximum value of $|H\left(e^{j\hat{\omega}}\right)|$, but can also be calculated analytically.

d) Use the value for β that makes $|H\left(e^{j\hat{\omega}}\right)|_{max} = 1$ and plot $H\left(e^{j\hat{\omega}}\right)$ on a decibel-scale. Set the minimum on the y-axis scale to -40 dB.

e) The passband of the bandpass-filter is defined by the frequency range where the magnitude is close to one. The most common definition is the range where $|H\left(e^{j\hat{\omega}}\right)|$. is greater than $1/\sqrt{2} = 0.707$. On a decibel-scale, this equals -3 dB.

Use the plot of the frequency response to determine the limits of the passband.

Note: you can use the NumPy function `argwhere` to find the frequencies where the magnitude satisfies $H\left(e^{j\hat{\omega}}\right) \geq 1/\sqrt{2}$.

f) Determine the analog frequency components that will be passed by this bandpass filter if the sampling rate is f_s=8000 Hz. Use the passband limits you found previously to make this calculation.

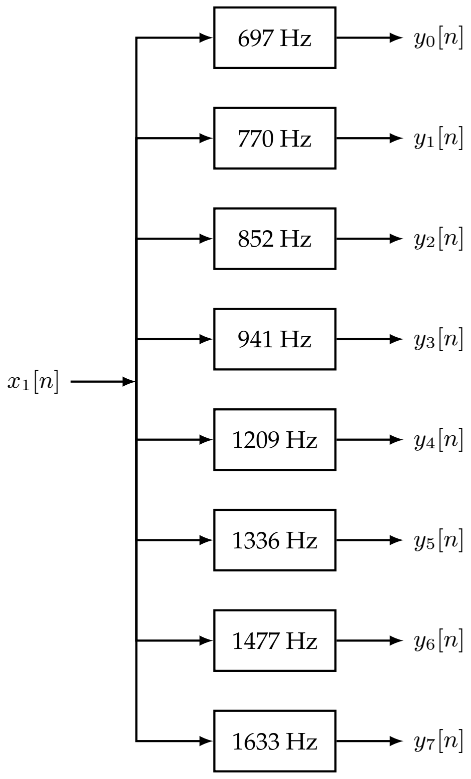

Figure 1. Filter bank consisting of bandpass filters (BPFs) which pass frequencies corresponding to the eight DTMF component frequencies listed in Table 1. The number in each box is the center frequency of the BPF.

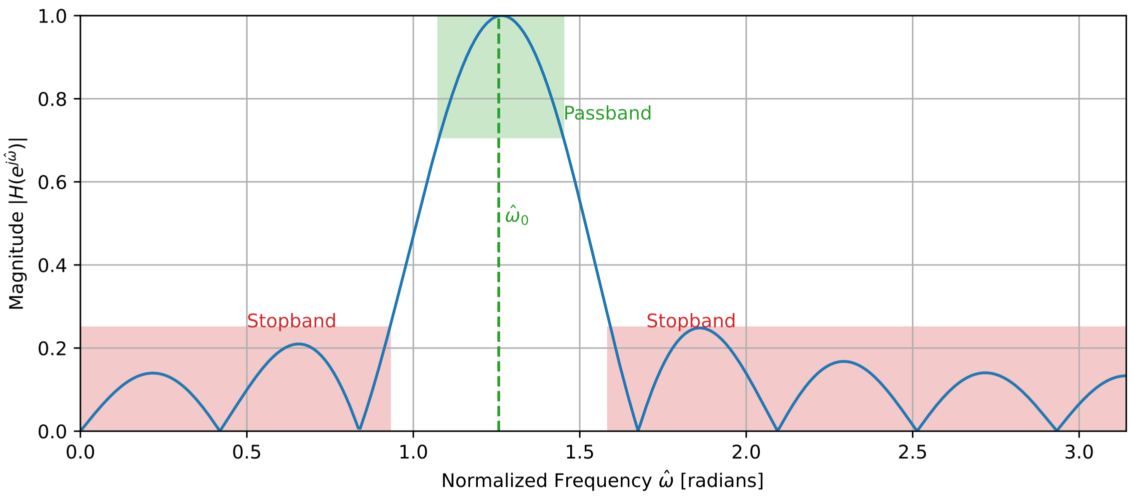

Figure 2. The frequency response of an FIR bandpass filter with its passband and stopband regions marked. The center frequency of this filter is $\hat{\omega}_0 = 0.4\pi \approx 1.26\,\text{radians}$ and the filter length is $L = 15$

3 Lab Excercise 1: Dialing - Generate a DTMF Signal

The task in this exercise is to make the dialling part of a DTMF telephone system. The result shall be a function `dial(number_string)` that can be called with a number string and will return the DTMF tones corresponding to this number. The tones shall be shown in a spectrogram and played as a sound, and this sound can be used to dial a number from a real telephone.

The skeleton for the final function is shown in Table 3.

Table 3. Skeleton for function to make a touch-tone DTMF signal from a telephone number entered as a string.

```
def dial(number):
    """Generate a DTFM tone sequence from a telephone number.

    Parameters
    ----------
    number : str
        Telephone number as a string

    Returns
    -------
    tone : 1D array of float
        Sequence of DTMF tones repreenting the telephone number
    """
    dialpad, freq, sample_rate = get_dtmf()

    #--- Your code comes here ---#

    sd.play(tone, sample_rate)  # Play the DTMF signal over the sound card
    return tone
```

This function calls another function `get_dtmf` that contains the layout of the keypad, the row and column frequencies, and the sample rate. The contents of this is shown in Table **??**.

Write the function `dial()` by using the give example code and following the steps below.

1) Create a function that converts the dial pad and row and column frequencies into a Python *dictionary* with keys equal to the numbers and characters on the dial pad, and values equal to the two frequencies belonging to that key. A suggested skeleton for this function `make_freq_table` is shown below

2) Write a function that generates the tone sequence from a telephone number formulated as a string. The requirements to this function are:
   - The input to the function is a string of the characters representing a telephone number, the frequency table as a dictionary, and the sample rate.
   - Any character not in the dialpad shall be ignored.
   - The tone is made from cosine-waves sampled at the specified sample rate. The cosine-waves can be synthesized using the function `make_summed_cos` written as part of Lab. 1.
   - The duration of each tone shall be 0.20 s, the interval between the tones shall be 0.05 s. The tones and intervals can be joined together witn NumPy's `append`-function.
   - The output of the function is the DTMF tone signal

3) Inspect the resulting tone sequence by plotting the tone signal as function of time in a figure.

4) Play the tone sequence over the computer's sound card. Does it sound like a telephone number?

5) Show the tone sequence in a spectrogram to check that the frequency combinations and the tone and interval durations are correct.

6) Play the tone sequence through a telephone to dial a number.

Table 4. Function to Create dial pad with DTMF tone frequencies.

```python
def get_dtmf():
    """Create dial pad and tone frequencies.

    Sets the standard parameters for a touch-tone telephone system

    Returns
    --------
    dialpad : 2D array of characters
    The telephone dial pad
    freq : 2D array of float
    The DTMF frequencies, rows [0] and columns [1]
    sample_rate : float
    System ample rate [samples/s]

    """
    dialpad = np.array([["1", "2", "3", "A"],
                        ["4", "5", "6", "B"],
                        ["7", "8", "9", "C"],
                        ["*", "0", "#", "D"]])    # Dial pad as 2D array

    # Frequencies associated with dial lines and columns
    freq = np.array([[697, 770, 852, 941],
                     [1209, 1336, 1477, 1633]],
    dtype=np.float64)

    sample_rate = 8000

    return dialpad, freq, sample_rate
```

Table 5. Skeleton of a function to build a dictionary from the frequencies in the touch-tone dial pad.

```python
def make_freq_table(dialpad, freq):
    """Create the touch-tone dial pad as a dictionary.

    Parameters
    ----------
    dialpad : 2D array of characters
        The telephone dial pad
    freq : 2D array of float
        The DTMF frequencies, rows [0] and columns [1]

    Returns
    --------
    freq_table : Dictionary with fields
        key : char
            Dial key as character
        values : list of float
            The two frequncies representing the key, [row_freq, column_freq]
    """

    #--- Your code comes here ---#

    return freq_table
```

4 Lab Excercise 2: Decoding - Identify a DTF Tone Sequence as Telephone Number

The task in this exercise is to make the decoding part of a DTMF telephone system. This identifies the tones sent from a telephone handset as a telephone number, and uses this to transfer the call to the correct address.

The DTMF decoding system needs two pieces

1. A set of bandpass filters that isolate individual frequency components. This is often called a filter bank and is illustrated in Figure 1.

2. A detector to determine whether or not a given component is present. The detector compares the output levels from the filters (y_k in Figure 1) and determines which two frequencies are most likely to be contained in the DTMF tone.

   In a practical system where noise and interference are also present, this scoring process is a crucial part of the system design, but we will only work with noise-free signals to understand the basic functionality in the decoding system.

The result of this part shall be a function `decode(tone)` that accepts a DTMF tone sequence `tone` as input and returns the telephone number represented by this tone. A skeleton of the final function is shown in Table 6.

Table 6. Skeleton of a function to decode a tone signal as a telephone number.

```python
def decode(tone):
    """Decode tone signal as telephone number.

    Parameters
    ----------
    tone : 1D array of float
        Sequence of DTMF tones repreenting the telephone number

    Returns
    -------
    number : str
        Telephone number identified from the tone
    """
    # Initialise
    dialpad, freq, sample_rate = get_dtmf()
    s = split(tone, sample_rate)

    #--- Your code comes here ---#

    return number
```

As for the dialling function, the decode function also uses the initialisation function `get_dtmf()` to set the parameters of the DTMF system.

Interpretation of the individual tones requires the DTMF sequence to be split into the individual numbers. A function `split` has been supplied to do this task, to save time and let you concentrate on the central parts of the lab. The use of `split` has been inclided into the skeleton for the function `decode`. The description of `split` is shown in Table 7.

Make the DTMF decoding function by the following steps

1) Make a filter bank for the filters in Table 1.

   Use FIR bandpass filters as shown in (1) and let the filter length L be a parameter that can be varied. Set the scaling factor β for each filter so that the filter gain at the centre frequency is one.

   Set the filter length to L=40 Plot the magnitude response $|H\left(e^{j\hat{\omega}}\right)|$ of all the filters in the filterbank in the same diagram. This should contain responses for all the 8 filters corressponding to the 8 DTMF frequencies in Table 1.

   Set the filter length to and L=80 and repeat the step above. Comment the result.

Table 7. Use of the function `split` to split the DTMF tone sequence into individual tones.

```
def split(tone, sample_rate=8000):
"""Split DTMF sequence into individual tones.

Parameters
----------
tone : 1D array of float
    Sequence of DTMF tones repreenting the telephone number
sample_rate :  float
    Sample rate for tone [samples/s]

Returns
-------
s : list of 1D array of float
    List of tones split into the individual numbers
```

2) The filter bank shall be designed so it can separate the DTMF frequencies.

   Define the passband as the frequencies $\hat{\omega}$ where $|H\left(e^{j\hat{\omega}}\right)| > 1/\sqrt{2} = 0.707$ and the stopband by the frequencies where $|H\left(e^{j\hat{\omega}}\right)| < 0.25$.

   Select a filter length L common for all filters so so that only one frequency lies within the passband of the BPF and all otherlie in the stopband.

   Which of the 8 filters limits the problem when the same value of L is used for all of them, i.e., for which center frequency is it hardest to meet the specifications for the chosen value of L?

3) Score the results from the filter bank.

   The next step is to evaluate if a frequency component is present in the output $y_k[n]$ from a filter in the filter bank.

   Split the DTMF tone sequence into individual tones by running the function `split` and run each of these tones through the filer bank.

   Remember that each tone is a synthesis of two tones, each with amplitude one, so the maximum amplitude of the DTMF sequence is close to 2.

   if one of these tones match the filter, the output should be close to one. If not, it will be close to zero.

   Set a detection limit using the maximum amplitude of the output signal $y_k[n]$ by using the same limit as for the passband. The frequency f_k for filter k is present if the output from this filter is above $1/\sqrt{2}$,

$\max\limits_{n} \lvert y_k[n] \rvert \geq \dfrac{1}{\sqrt{2}}$	Frequency is present
Otherwise	Frequency is not present

   Run all tones in the DTMF signal through all filters in the filterbank. Use the criterion above to determine which two frequencies are present in each tone. The result may e.g. be implemented as a 2D boolean array, a scoring table with tone number as the first index and frequency number as the second.

4) Identify the telephone number.

   For each tone, use the criterion above to find the two frequencies present in the signal. Use this to identify the number or symbol on the dialpad represented by this tone.

   Each tone should match exactly two frequencies on the dialpad. If this is not the case, something went wrong in the interpretation and the function shall return an error message.

   Combine the resulting numbers and symbols to a telephone number which may also include special signs like $\star$ and #.

5) Test on a real telephone.

   The decoding may be tested on the sound from a real telephone set. Record the sound while dialling a number on a telephone. Convert this recording to a NumPy array and run it through the `decode()` function.

   Is your program able to recognise the number from the tones?

References

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

[2] ——, "Lab P13: Encoding and Decoding Touch-Tone (DTMF) Signals," 2016. [Online]. Available: https://dspfirst.gatech.edu/database/?d=labs

[3] International Telecommunication Union, "Technical features of push-button telephone sets. ITU-T Recommendation Q.23," 1988. [Online]. Available: https://www.itu.int/rec/T-REC-Q.23/en

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

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

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

[7] 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

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

[9] 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

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

[11] 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