Home exam

FYS-2006- Signal Processing

Canditate: 44

31. oktober 2018

Inneholder 22 sider, inkludert forside.

Institutt for fysikk og teknologi

1 Summary

In this task a signal which is a complex base band signal recorded from a single radio antenna. The complex base band signal represents the electric potential received by the radio antenna as a function of time. In the data file we use there are alot of potential radio stations as the data is within 5Mhz of bandwidth. In this task we will use spectral analysis to identify stations and other signal processing techniques as filtering to get audible sound files.

2 Theory and definitions

One of the most important aspects of signal processing is the Fourier transform, as we are working with discrete chunks of the signal when we use a computer we will be using the FFT which is the "Fast Fourier Transform". The reason for using FFT is that it needs less operations $\mathcal{O}(N \log N)$ and $\mathcal{O}(N^2)$ for the DFT(Discrete Fourier Transform) and therefore faster to compute, which is very good as we work with alot of data. The DFT is a discrete frequency domain representation of periodic discrete-time signals. A periodic signal can be represented using infinitely many complex sinusoidal components which can be represented as the following equation,

$$x[n] = \sum_{-\infty}^{\infty} X[k] e^{i \frac{2\pi}{T} kn}$$

Here x[n] is the discrete time representation of the signal, X[k] is the frequency representation and the exponential is sinusoid which helps make sinusoidal weights in the DFT. But because of aliasing we only have N unique discrete-time complex sinusoids of length N. Which is also alot more convenient as we cant have infinite sum in real life. This leads to the Discrete Fourier Transform,

$$\hat{X}[k] = \sum_{n=0}^{N-1} x[n] e^{-i \frac{2\pi}{N} nk} \tag{1}$$

The fast version of this will be used in this exam to get the frequency domain of the signal. To get from frequency to time domain we use its inverse,

$$x[n] = \frac{1}{N} \sum_{n=0}^{N-1} \hat{X}[k] e^{-i \frac{2\pi}{N} nk} \tag{2}$$

In our code we will use the numpy library in python with the following methods,

```
numpy.fft.fft # https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fft.html
numpy.fft.fftfreq # https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftfreq.html
numpy.fft.fftshift # https://docs.scipy.org/doc/numpy/reference/generated/numpy.fft.fftshift.html
```

Window functions is also used, window functions are zero outside the interval this is to extract a finite section of a long signal, this allows us to get a given window length of the signal. This is good because if we want to low pass a signal we can use the ideal low pass (sinc function), but the sinc function goes to infinity, so practically we need to limit it, and this can be done with the window function.

Oppgave 5

5.1)

Starting by importing the signal

```
# Importing necessary libs
import numpy as np
import h5py
import scipy
import matplotlib.pyplot as plt
from scipy.io.wavfile import read,write


h=h5py.File("hf_12.5_5MHz.h5","r")
z=np.copy(h["z"].value)
h.close()

```

Each index in the array is one sample, so the total amount of samples is the total length of the array,

```
# Samples long:
print(str(len(z))+' samples')

```

This is 50000000 samples.

5.2)

```
# Sampled with a 5MHz sample rate—> 1e6 samples/second
print(str(len(z)/(5*1e6))+' Seconds')

```

Can find the the total time of the signal by taking the amount of samples and divide by the sampling rate, which in this case is $f_s = 5 \cdot 10^6$ samples/s which is found to be 10.0Seconds.

5.3)

The dependant variable is z(n), the complex base-band signal representation, and the independent variable is time, in this case the sampling number "n".

5.4)

$$\hat{Z}[k] = \sum_{k=0}^{N-1} z[n] e^{-i\frac{2\pi}{N}nk}$$

If the signal was real all that would happen is that the sign of the complex exponential in the Fourier transform will change and we would have spectral component which are conjugate symmetric. If the the signal is complex we would have changes in the sign of the signal and in the complex exponential of the Fourier transform and we would not always have conjugate symmetric spectral components.

5.5)

The practical benefit of using complex base-band signals for representing radio signals is that it uses less of the frequency spectrum since it does not have a complex conjugate.

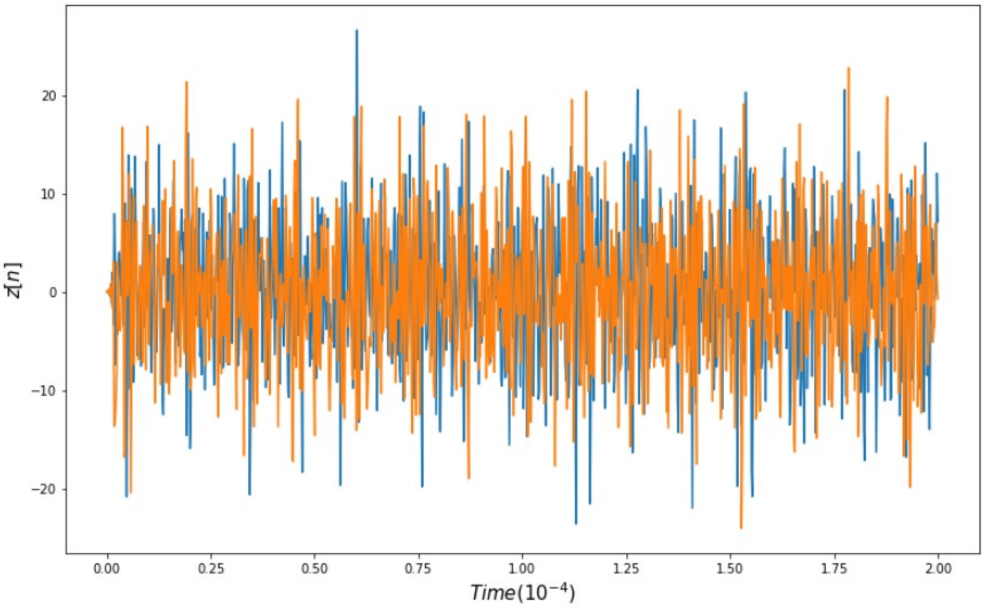
Oppgave 6

6.1)

```
# Finding how long the time array should be
print(str(1000/(5*1e6))+' Seconds')

# Plotting real and imaginary part of the signal
time_vector = np.linspace(0,2,len(z[0:1000]))
plt.figure(figsize = ((13,8)))
plt.plot(time_vector,z[0:1000].real)
plt.plot(time_vector,z[0:1000].imag)
```

```
plt.xlabel('$Time (10^{-4})$',fontsize = '15')
plt.ylabel('$z[n]$',fontsize = '15')

plt.show()
```

Plot is shown in figure(1) below



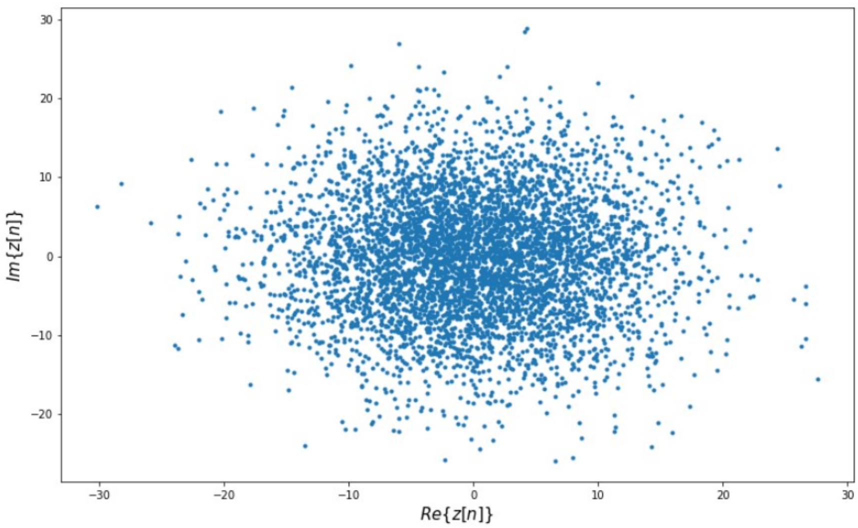
Figur 1: Plot of the signal for 0.0002 Seconds

6.2) The length of the signal is 0.0002 Seconds

6.3) Looks like the signal is full of noise

6.4) Since T = 1/f we have that, $T_s = 1/f_s = 1/(5 \cdot 10^6 Seconds) = 2 \cdot 10^{-7} Seconds$

```
    # Plotting the complex plane (x = real , y = complex)
time_vector = np.linspace(0,1,len(z[0:5000]))
plt.figure(figsize = ((13,8)))
plt.plot(z[0:5000].real ,z[0:5000].imag ,'.')
plt.xlabel('$Re\{z[n]\}$',fontsize = '15')
plt.ylabel('$Im\{z[n]\}$',fontsize = '15')

plt.show()

# Finding mean of the signal
mean_real = np.mean(z[0:5000].real)
mean_imag = np.mean(z[0:5000].imag)
mean_len = np.sqrt((mean_real)**2 + (mean_imag)**2)

print('Magnitude of the mean of the signal is {:.5f}'.format(mean_len)
    )
print('Approx = {}'.format(int(mean_len)))

```



Figur 2: Plot of the the complex plane.

Magnitude of the mean of the signal is 0.08555 which is ≈ 0, so it seems to be totally random around the origin. We see that the signal is full of noise which is to be expected since there are a lot of signals together.

Oppgave 7

7.1)

We have that $\hat{\omega} = \frac{2\pi f}{f_s}$, solving for frequency when $\hat{\omega} = \pm\pi$ we get, $f = \pm\frac{f_s}{2}$

```
 # Sample rate Hz
fs = 5e6
f0 = 12.5e6
# pi
print('f+ = {:.0f} Hz'.format(fs/2))
# -pi
print('f_ = {:.0f} Hz'.format(-fs/2))
# Frequency shifted
print('f+s = ',f0 + fs/2)
print('f-s = ',f0 -fs/2)

```

The frequencies represent $f_- = -2500000Hz$ and $f_+ = 2500000Hz$ and if we frequency shift back to the original signal we get, $f_{s-} = 10MHz$ and $f_{s+} = 15MHz$

7.2)

```
 print('The frequency resolution is {:.2f} Hz'.format(5e6/131072))

```

The frequency resolution is 38.15 Hz. Here we took the sampling rate/number of FFT points. This gives the frequency step in the frequency domain. So if we have two signal close together and its difference is lower then 38.15Hz we wont find it. If we increase the number of FFT points we get sharper frequency domain, but this increases the step size in time so we lose information in the time domain, this is called the time-frequency ambiguity.

7.3)

```
# Sample rate Hz
fs = 5e6
# Center frequency
f0 = 12.5e6
# Making window with length of the signal
N = len(z)
# Frequency spectrum of the signal
```

```
spectrum = np.fft.fftfreq(N,1/(5e6))
# Shift the frequency from 0,2pi to -pi to pi
freq = np.fft.fftshift(spectrum)
# Adding the center frequency
resultfreq = freq + f0

print(resultfreq)

>>[ 10000000.   10000000.1  10000000.2 ...,  14999999.7  14999999.8
  14999999.9]

```

Here we made the frequency spectrum with $N = 131072$ and and then we shift the signal to so that we get our frequency between $-\pi$ and π, after adding the center frequency of 12.5MHz we get resulting array going rom 10MHz to 15MHz. Here in this exam we will use the FFT(Fast Fourier Transform) for all Fourier transforms, the reason for using FFT is that is needs less operations $\mathcal{O}(N \log N)$ and $\mathcal{O}(N^2)$ for DFT(Discrete Fourier Transform) and therefore faster to compute, which is very good as we work with alot of data.

7.4)

```
# importing signal package from scipy
from scipy import signal
# selecting hann signal and selecting window length of 131072
tapering_window = signal.hann(131072)

```

Here i used the Hann window because it has smoother sides of window so that it has better transition between the pass band and the band stop. This gives us better rejection of out of band signals which is what we wanted to do.

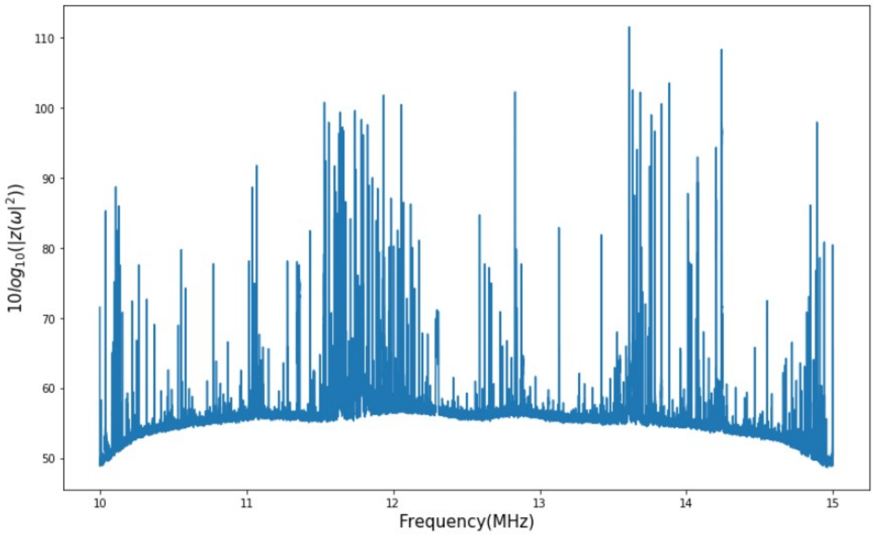
7.5)

```
# Sample rate Hz
fs = 5e6
# Center frequency
f0 = 12.5e6
# fft length
N = 131072
# number of windows to avarage
n_windows = int(len(z)/N - 1)
# initialize spectrum with zeros
spectrum = np.zeros(N)
# Avarage all spectra
for i in range(n_windows):
    # tapered DFT of slice N length of signal
    Z = np.fft.fft(tapering_window*z[(i*N):(i+1)*N])
    # sum the magnitude squared
    spectrum += np.abs(Z)**2
# Taking the avarage
spectrum/= n_windows
# shift to -pi to pi from 0 to 2pi
spectrum= 10*np.log10(np.fft.fftshift(spectrum))

#Find frequency
freq = np.fft.fftfreq(N,1/fs)
# Shift frequency to +- pi
freq = np.fft.fftshift(freq)
# add the center frequency
freq += f0
# Change unit to MHz
freq *= 1/1e6

```

Here i used the example code and added spectrum/ = n_windows to average the the power spectrum and added the frequency.

7.6)

```
#Plotting
plt.figure(figsize = ((13,8)))
plt.plot(freq,spectrum)
plt.ylabel('$10log_{10}(|z(\omega|^2))$', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()

```



Figur 3: Plot of the whole frequency spectrum with the magnitude on the y-axis.

Here we see the power spectrum of the whole frequency spectrum of the signal. Here there are frequencies who are potential radio station since they have strong averaged power. Here we used decibel for the y-axis to easier see the difference between the magnitudes, because otherwise some very strong frequencies would dominate and it would be harder to analyze the signal.

7.7)

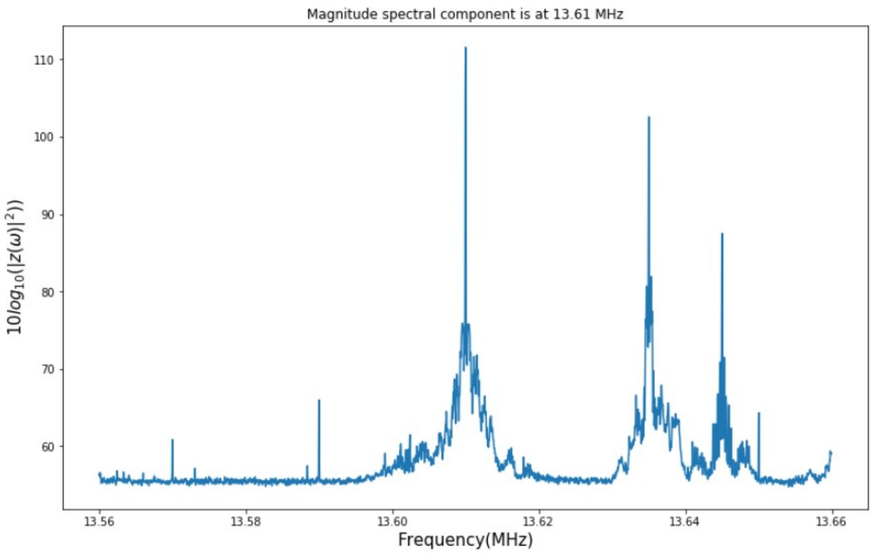
```
 # Initializing numpy arrays
freq = np.copy(np.array(freq))
spectrum = np.copy(np.array(spectrum))

# Getting frequency with highest magnitude
largest_magnitude = freq[spectrum.argmax()]
print('Largest magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude))

```

The signal with largest avarage magnitude spectral component is at 13.61 MHz.

7.8)

```
# every jump is 38.16 Hz
delta_f = 38.16

# difference from center
khz100 = int(50e3)

# Getting lowest and highest value of the local spectra
low = int(spectrum.argmax()-khz100/delta_f)
high = int(spectrum.argmax()+khz100/delta_f)

# Getting frequency with highest magnitude
largest_magnitude = freq[spectrum.argmax()]
print('Largest magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude))
```

```

# Plotting
plt.figure(figsize = ((13,8)))
plt.plot(np.copy(freq[low:high]),np.copy(spectrum[low:high]))
plt.title('Magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude))
plt.ylabel('$10log_{10}(|z(\omega)|^2))$', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()
```



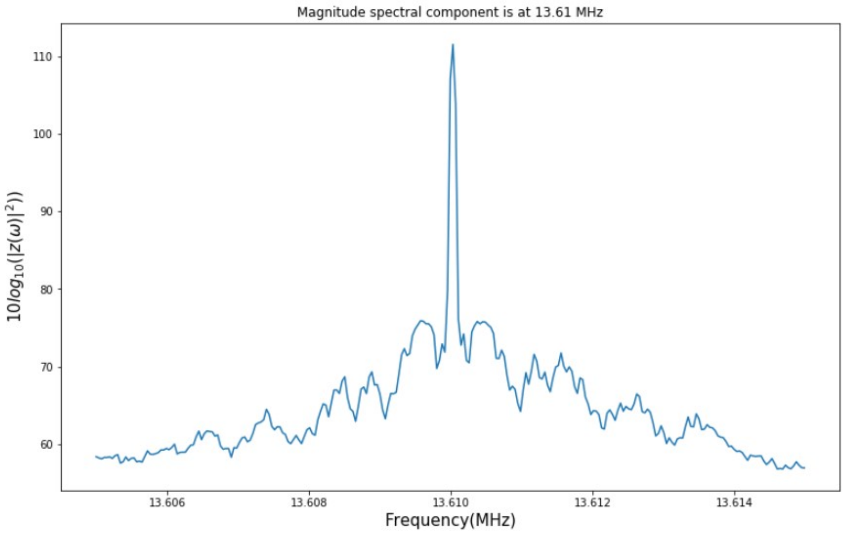
Figur 4: Plot of the whole frequency spectrum with the magnitude on the y-axis.

Here we have the frequencies in the spectrum ±50kHz around the strongest signal. One of signals are dominating at frequency around 13.610MHz. Here we can see a big spike in the middle that looks like a Dirac delta function. The frequency resolution was also used to find the number of steps needed between lowest and highest frequency.

7.9)

```
# every jump is 38.16 Hz
delta_f = 38.16

# difference from center
khz100 = int(5e3)

# Getting lowest and highest value of the local spectra
low = int(spectrum.argmax()-khz100/delta_f)
high = int(spectrum.argmax()+khz100/delta_f)

# Getting frequency with highest magnitude
largest_magnitude = freq[spectrum.argmax()]
print('Largest magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude))

# Plotting
plt.figure(figsize = ((13,8)))
plt.plot(np.copy(freq[low:high]),np.copy(spectrum[low:high]))
plt.title('Magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude))
plt.ylabel('$10log_{10}(|z(\omega)|^2))$', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()
```

In figure(5) below we can see how the spectral component look like in the ±10kHz

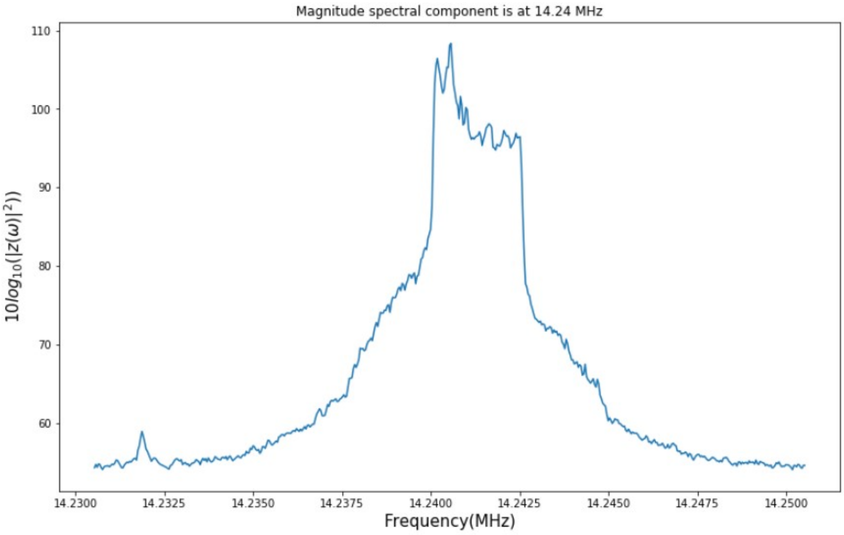


Figur 5: Plot of the whole frequency spectrum with the magnitude on the y-axis.

range. It looks very symmetric around the max value, and also has a very narrow spike in magnitude.

7.10)

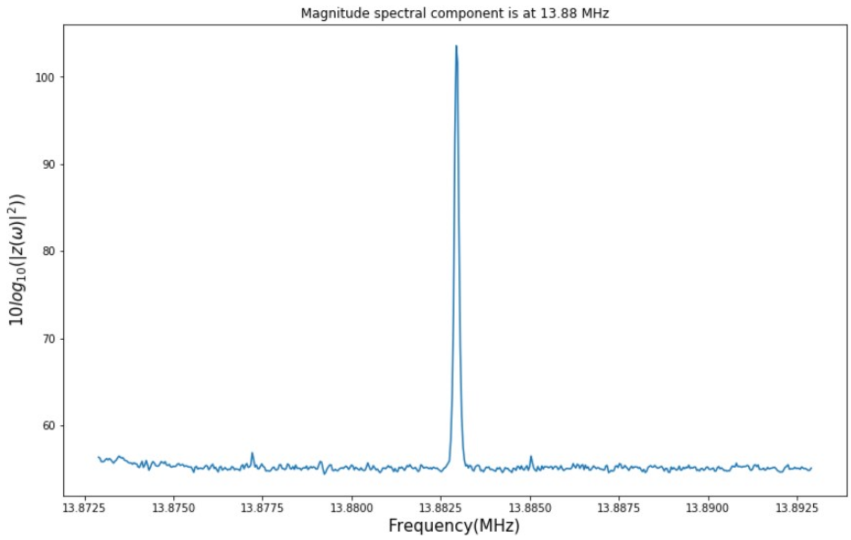
```
# Observing that the second biggest peak is at higher frequency then
    the biggest one so i narrow the spectrum down from the highest
    frequency of the largest magnitude frequency
spectrum_new = np.copy(spectrum)[high:]
freq_new = np.copy(freq)[high:]


largest_magnitude_new = freq_new[spectrum_new.argmax()]
print('Second largest magnitude spectral component is at {:.2f} MHz'.
    format(largest_magnitude_new))

# every jump is 38.16 Hz
delta_f = 38.16

# difference from center (10kHz)
diff_second = int(10e3)

# Getting lowest and highest value of the local spectra
low_new = int(spectrum_new.argmax()-diff_second/delta_f)
high_new = int(spectrum_new.argmax()+diff_second/delta_f)

# Plotting
plt.figure(figsize = ((13,8)))
plt.plot(np.copy(freq_new[low_new:high_new]),np.copy(spectrum_new[
    low_new:high_new]))
plt.title('Magnitude spectral component is at {:.2f} MHz'.format(
    largest_magnitude_new))
plt.ylabel('$10log_{10}(|z(\omega)|^2))$', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()
```



Figur 6: Plot of the whole frequency spectrum with the magnitude on the y-axis.

The second largest in magnitude doesn't have the same level of symmetry as the first, but there are some symmetry. This one unlike the others is very broad in frequency.

```
 # Observing that the third biggest peak is at lower frequency then
     the biggest one so i narrow the spectrum down from the highest
     frequency of the largest magnitude frequency

spectrum_third = np.copy(spectrum)[high:high+low_new]
freq_third = np.copy(freq)[high:high+low_new]

print(freq_third)

largest_magnitude_third = freq_third[spectrum_third.argmax()]
print('Second largest magnitude spectral component is at {:.2f} MHz'.
     format(largest_magnitude_third))

# every jump is 38.16 Hz
delta_f = 38.16

# difference from center (10kHz)
diff_third = int(10e3)

# Getting lowest and highest value of the local spectra
low_third = int(spectrum_third.argmax()-diff_third/delta_f)
high_third = int(spectrum_third.argmax()+diff_third/delta_f)


freq_third = freq_third[low_third:high_third]
spectrum_third = spectrum_third[low_third:high_third]


# Plotting
plt.figure(figsize = ((13,8)))
plt.plot(freq_third,spectrum_third)
plt.title('Magnitude spectral component is at {:.2f} MHz'.format(
     largest_magnitude_third))
plt.ylabel('$10log_{10}(|z(\omega)|^2))$', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()

```



Figur 7: Plot of the whole frequency spectrum with the magnitude on the y-axis.

This third largest shown in figure(7) is also symmetric and has a very narrow magnitude spike.

Oppgave 8

8.1)

```
 # Importing signal
h=h5py.File("hf_12.5_5MHz.h5","r")
z=np.copy(h["z"].value)
h.close()

# fft length
N = 131072
# Overlap
step = int(N/2)
# Length of signal
length = len(z)

# steps
n_steps = int(length/step)

# Using Hann window
tapering_window = signal.hann(N)

# Initialize matrix to contain N spectral components
# as a function of time (n_steps)
S = np.zeros([n_steps-1,N], dtype= np.float32)
# # Go through all time steps

for i in range(n_steps):

    zin = z[i*step:(i*step+N)]
    if len(zin) == N:
        # Filling up rows(time) with the signals magnitude over the
    window
        S[i,:] = np.abs(np.fft.fftshift(np.fft.fft(tapering_window*zin
    )))**2.0


```

Here we use a Hann window(because it is good to reject out of band signals) to ”scan” our signal in such a way that we only take the FFT over steps of the signal and zero out information outside the window. For each iteration we assign it as a step in time in our matrix. The result for each iteration is Fourier transformed to frequency domain and shifted to origin. Then we take the magnitude to get the strength of the signal. This whole operation is essentially the formula given below,

$$\hat{X}[t,k] = \sum_{n=0}^{M-1} x[n + t\Delta n] w_N[n] e^{-i\frac{2\pi}{M}kn}$$

Where $\hat{X}$ is the 2d matrix. Δn is our time step, w is the window and the complex exponential is part of the Fourier transform.

8.2)

```
print('N/2 represents {:.3f} seconds'.format(step/(5*1e6)))

```

N/2 represents 0.013 seconds. Here we took the sampling with N/2 and divide by the sampling rate which is 5MHz

8.3)

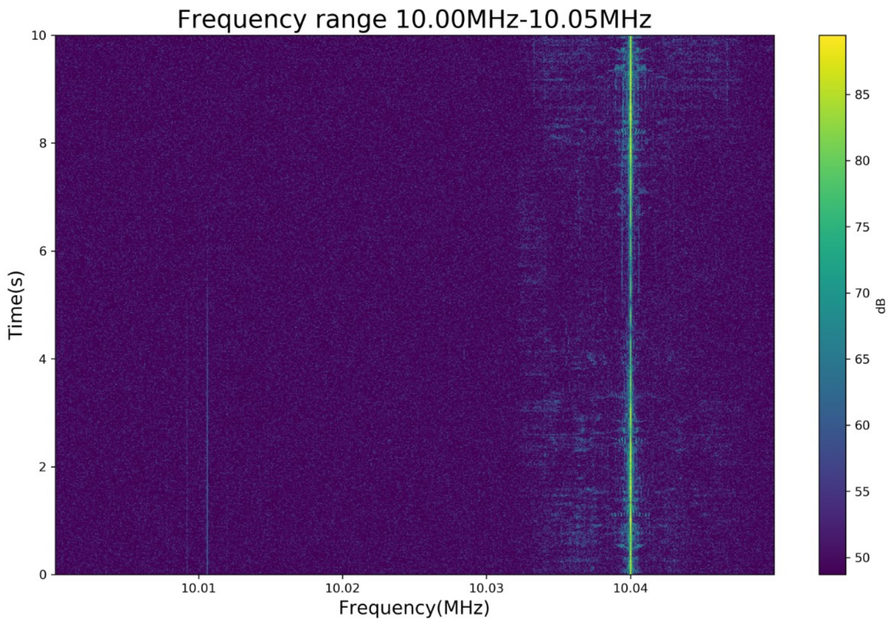
```
# Making time vector from 0 to 10 with len(S) -> 762 points /long
tvec = np.linspace(0,10,len(S))

```

Here we made the time vector with 762 points in time between 0.10 seconds, this will be used in our spectrogram plot.

8.4)

```
increment = int(50e3)/1e6
f0 = int(10e6)/1e6
f1 = f0 + increment

#/int(1e6)

# Using the frequency from Task 7.5
fvec = freq


for i in range(100):
    freq_idx = np.where((fvec > f0 )&(fvec < f1))[0]

    # Db scale
    dB = 10.0*np.log10(S[:,freq_idx])
    plt.figure(figsize = ((13,8)))
    # Use the median noise floor as the lowest color to make it easier
     to distinguish the differents signals
    plt.pcolormesh(fvec[freq_idx],tvec,dB,vmin = np.nanmedian(dB))
    plt.title('Frequency range %.2fMHz-%.2fMHz'%(f0,f1),fontsize = '20
')
    plt.ylabel('Time(s)',fontsize = '15')
    plt.xlabel('Frequency(MHz)',fontsize = '15')
    zcol = plt.colorbar()
    zcol.set_label('dB')
    plt.savefig('spectrum/spectrum-{}'.format(i+1),dpi = 300)
    # Increasing the frequency band for each iteration
    f0 += increment
    f1 += increment
    plt.show()
    plt.close()


```

8.5)



Figur 8: Spectrogram plot in the range 10.00MHz - 10.05MHz

Here we see the spectrogram plot in the range 10.0MHz - 10.05MHz. We can observe that we have a strong signal 10.04MHz and that it has around 20kHz bandwidth.

Oppgave 9

9.1)

The spectrum is symmetric since we since the original is real.

9.2)

I chose signal at 11.53 MHz and used my class definition to get the signal as described in 9.7-9.13 in this section. It sounded like it was some kind of turkish? music/dance channel.

```
signal1 = radio_station(z,11.53e6,h1,'turkish dance channel?')

```

9.3)

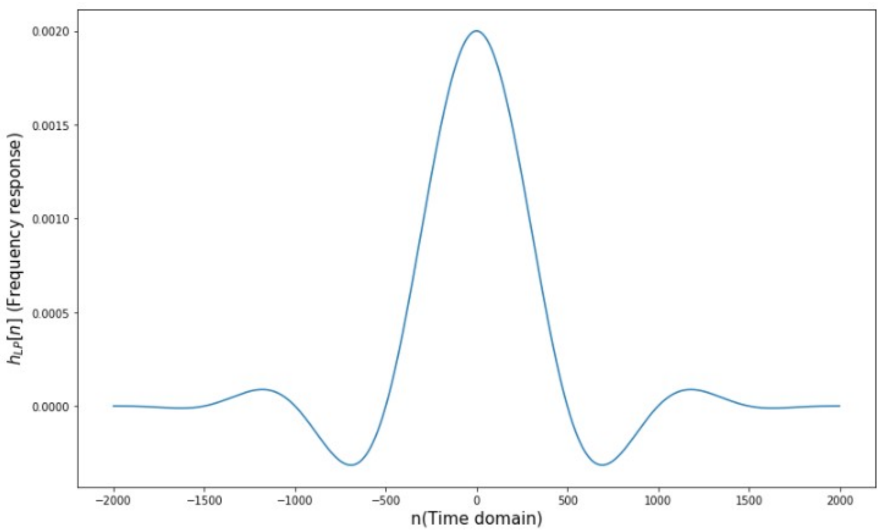
```
from scipy.signal import hann

def hann_window(length,cutoff):
    """
    Creates the the ideal*window filter

    Param 1: length of filter
    Param 2: Cutoff frequency

    Returns samples,filter h (time domain)

    """

    # ideal filter length
    samp = length
    # sampling rate
    fs = int(5e6)
    # Cutoff frequency
    f_cut = cutoff
    # sample range
    n = np.arange(-samp/2,samp/2)
    ## Filter cutoff frequency
    hatw0 = 2*np.pi*f_cut/fs
```

```python
    # Make finite impulse filter

    h = hatw0/np.pi*np.sinc(hatw0*n/np.pi)

    # Window
    wl = length
    W = hann(wl)

    # ideal with window
    h = W*h
    return n,h
n,h = hann_window(4000,int(5e3))

```

Here i have a filter h with 4000 samples. Here i first made the ideal filter using the sinc function (the Fourier transform of a rectangular function). Since with the ideal filter we also need the signal to be infinite in time, since this is not possible or practical, we use a window (in this case Hann window) to cut off all frequency outside the window. The reason for choosing Hann window is because it has better rejection of out of band frequencies. I also defined it in a function so i can alter the filter properties later.

9.4)

```python
# Impulse response
plt.figure(figsize = ((13,8)))
plt.ylabel('$h_{LP}[n]$ (Frequency response)', fontsize = '15')
plt.xlabel('n(Time domain)',fontsize = '15')
plt.plot(n,h,'-')
plt.show()

```



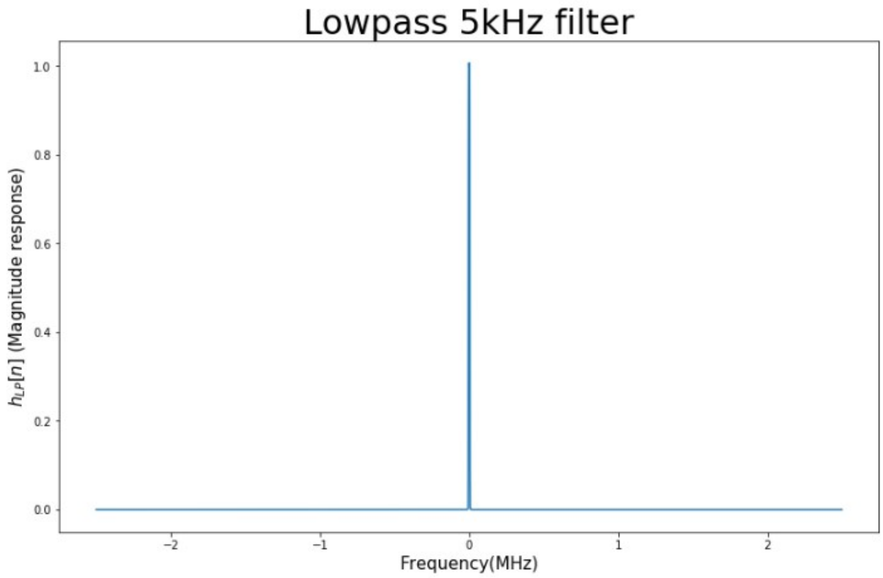
Figur 9: Impulse respons of the low pass filter

Here we have impulse response of the low-pass hann-window filter. Here we see how it drops outside the our "searching area". We will later use this to traverse our signal and hopefully only get frequencies below 5kHz.

9.5)

```python
# Frequency vector
fvec = np.fft.fftshift(np.fft.fftfreq(len(h), d = 1/5e6))

# Frequency range
f0 = -2.5e6
f1 = 2.5e6

```

```python
# Find index between the range.
fvec_2c5M = np.where((fvec >= f0)& (fvec <= f1))[0]


# Magnitude response
mag_resp = np.abs(np.fft.fftshift(np.fft.fft(h)))

# Plot figure
plt.figure(figsize = ((13,8)))
plt.plot(fvec[fvec_2c5M]/1e6,mag_resp)
plt.title('Lowpass 5kHz filter', fontsize = '30')
plt.ylabel('$h_{LP}[n]$ (Magnitude response)', fontsize = '15')
plt.xlabel('Frequency(MHz)', fontsize = '15')
plt.show()
```

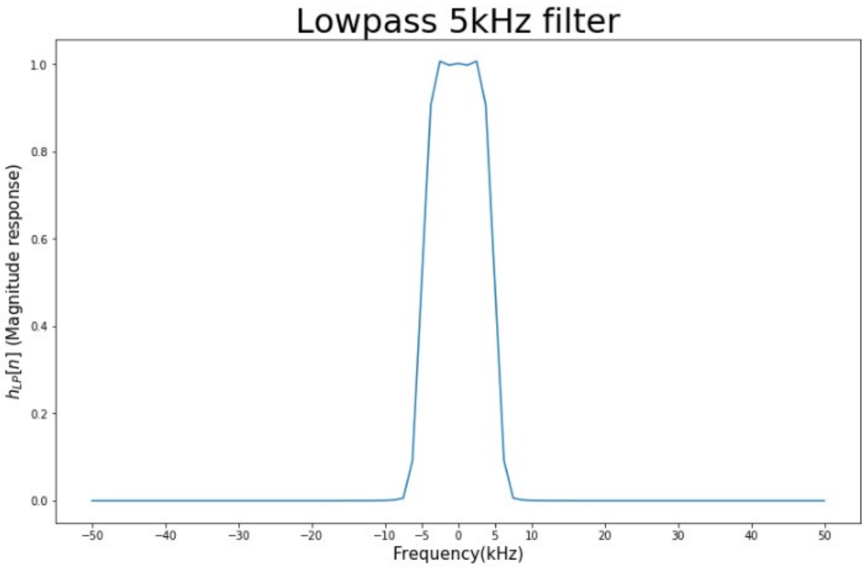


Figur 10: Magnitude response of the low pass filter

The frequency domain is so large in terms of the frequency range of the filter (it only lets in frequencies less then 5kHz) so we only see a line.

9.6)

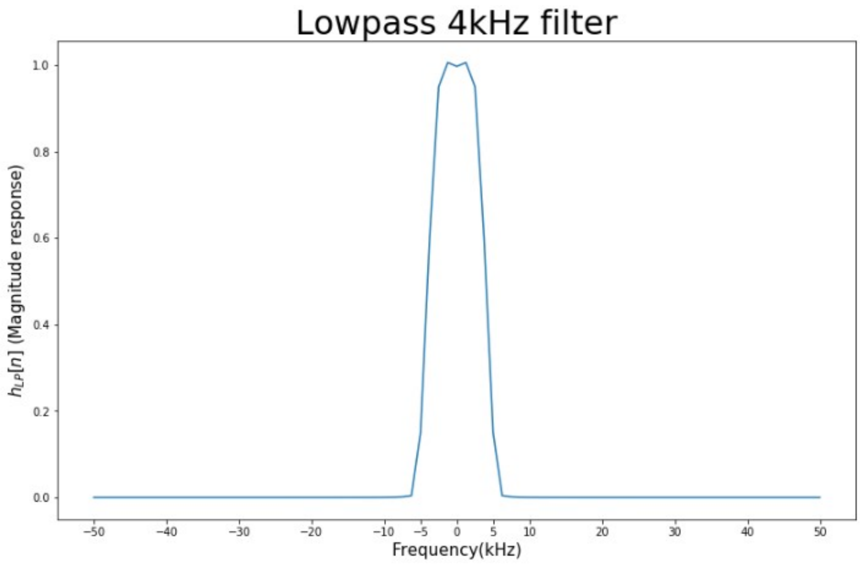
```python
# Frequency vector
fvec = np.fft.fftshift(np.fft.fftfreq(len(h), d = 1/5e6))
# Make mangitude vector
mag_resp = np.abs(np.fft.fftshift(np.fft.fft(h)))
# Frequency range
f0 = -50e3
f1 = 50e3

# Get index of values in range
fvec_50k = np.where((fvec >= f0)& (fvec <= f1))[0]

# Plot
plt.figure(figsize = ((13,8)))
plt.plot(fvec[fvec_50k]/1e3,mag_resp[fvec_50k])
plt.title('Lowpass 5kHz filter', fontsize = '30')
plt.ylabel('$h_{LP}[n]$ (Magnitude response)', fontsize = '15')
plt.xlabel('Frequency(kHz)', fontsize = '15')
plt.xticks([-50,-40,-30,-20,-10,-5,0,5,10,20,30,40,50])
plt.show()
```



Figur 11: Magnitude response of the low pass filter

Here we changed the range of frequency to -50kHz to 50kHz to see better what the filter actually does. Here we can see that the filter rejects frequencies over ≈ 6kHz so the filter does not filter exactly at 5kHz. If we change the cutoff frequency to 4kHz we get the plot in figure(12) below.

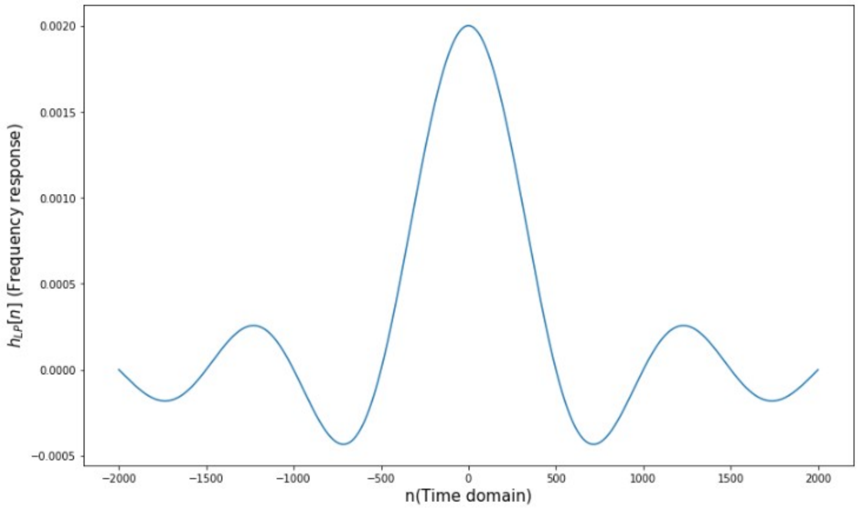
```
# Change filter to be sharper around 5kHz
n,h1 = hann_window(4000,int(4e3))
```



Figur 12: Magnitude response of the low pass filter

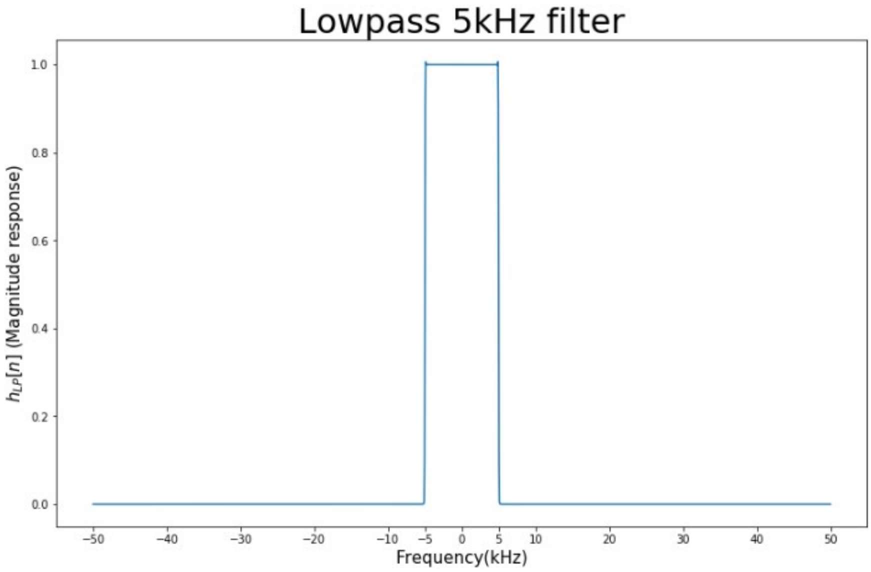
We can use the filter with cutoff at 4kHz to get a better rejection over 5kHz.

We can change some values for the filter and watch the time-frequency ambiguity. Here in figure(13) we can observe that we have more uncertainty/ less sharp in time as it wobbles alot more as it traverse away from origin.



Figur 13: Impulse response of the low pass filter with window length 100000

In figure (14) we have sharper cutoff in frequency.



Figur 14: Magnitude response of the low pass filter with window length 100000

So its very clear how the ambiguity is manifesting.

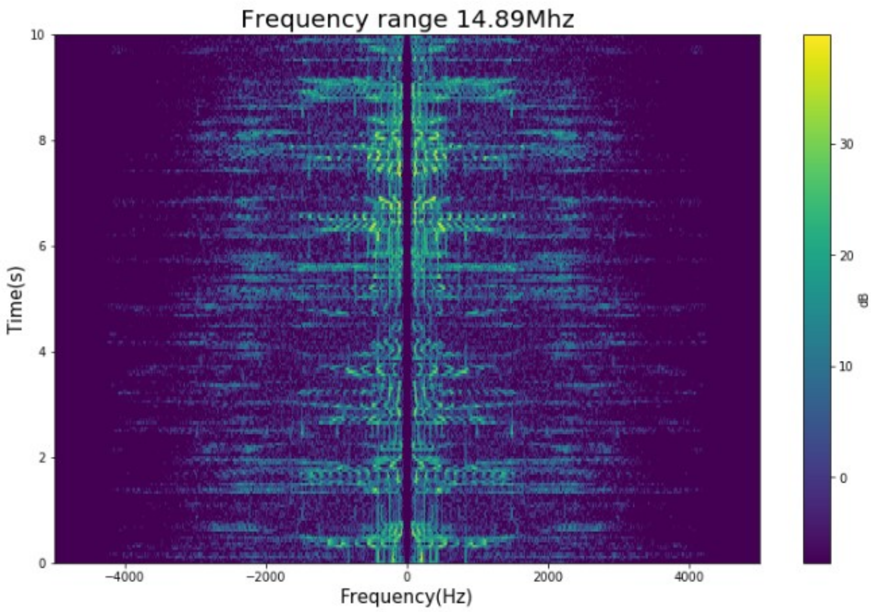
9.7-9.13)

```python
import matplotlib.pyplot as plt
import numpy as np
from scipy.io.wavfile import write

class radio_station():
    """

    Summary:

        Object is the processes signal for a radio frequency.

    Parameters:

        Param1:(list) The signal
        Param2:(int) The radio station frequency
        Param3:(str) 'Optional' Label the station

    Functions:

        object.writefile():
            Writes out the soundfile

        object.spectrogram():
            Plots spectromgram with matplotlib in the frequency -5000
    to 5000 Hz
            and 0 to 10 seconds.

    """


    def __init__(self,signal,radio_frequency,h1, label = ''):

        #Initializing object values
        z = signal
        self.radio_frequency = int(radio_frequency)
        self.label = label
        self.h = h1

        # Radio station frequency
        f0 = int(self.radio_frequency)
        # Sampling rate
        sr = int(5e6)
        # Decimation by 500 to go from 5MHz to 10kHz
        dec = 500
        # center frequency
        cf = 12.5e6
        low_pass_filter = self.h
        # length of the filter
        filter_length = len(low_pass_filter)
        # Number of time steps at 10 kHz
        n_steps = int(((len(z)-filter_length))/dec)
        # time vector
        t = np.arange(filter_length , dtype = np.float)/sr
        # frequency shift to DC, radio station at frequency f0
        df = f0-cf
        # complex exponential with negative frequency
        csin = np.exp(-1j*2.0*np.pi*t*df)
        z_10kHz =np.zeros(n_steps,dtype = np.complex64)
        # phase variable to store the current phase of the complex
    exponential signal
        phase0 = 0.0

        for i in range(n_steps):
            # initial phase of the sinusoid
            phase = np.exp(1j*phase0)
            z_10kHz[i] = np.sum(low_pass_filter*z[(i*dec):(i*dec +
    filter_length)]*csin*phase)
            # the store the initial phase of the exponential signal
    for the next step
```

```
                phase0 = np.fmod(phase0-2.0*np.pi*(dec*df/sr),2.0*np.pi)


        # Clean up signal
        # FFT
        Z = np.copy(np.fft.fft(z_10kHz))
        # filter out negative frequencies
        Z[int(len(Z)/2):len(Z)] = 0.0

        # Remove low frequency carrier signal, sampling freq 10kHz
        fre = np.fft.fftfreq(len(Z),d = 1/10e3)
        # Get index of values where frequency is below 50, since we
have already removed negative, we can just remove the positive
side.
        ind = np.where(fre < 50)[0]
        # Set values in the spectral component to zero (remove them).
        Z[ind] = 0.0

        # Inverse FFT back to time domain
        z_10kHz = np.fft.ifft(Z)
        # Assign processed signal to self property
        self.z_10kHz = z_10kHz


    def writefile(self):
        """
        Takes the real value of the processed signal and then writes
it out as a .wav file.
        """
        # Take real values of the signal
        audio = np.real(self.z_10kHz)
        # Scale to unity
        audio = audio/np.max(np.abs(audio))
        # Write soundfile
        write('%s.wav'%self.radio_frequency,int(10e3),audio)
        print('File {}.wav has been written'.format(str(self.
radio_frequency)))



    def spectrogram(self,N = 1000):
        """
        Summary:
            Plots the dynamic spectrum

        Parameters:
            Param1: (int) N is the fft length
        """
        # Get real values of signal
        audio = np.real(self.z_10kHz)
        # Overlap
        step = int(N/2)
        # Length of signal
        length = len(audio)

        # steps
        n_steps = int(length/step)


        # Using Hann window
        tapering_window = signal.hann(N)

        # Initialize matrix to contain N spectral components
        # as a function of time (n_steps)
        S = np.zeros([n_steps-1,N], dtype= np.float32)
        ## Go through all time steps
        for i in range(n_steps-1):

            zin = audio[i*step:(i*step+N)]
                # Filling up rows(time) with the signals magnitude
```

```
    over the window
            S[i,:] = np.abs(np.fft.fftshift(np.fft.fft(tapering_window*zin)))**2.0


        # Initialize time vector
        tvec = np.linspace(0,10,len(S))
        # Initialize frequency vector transpose to get column dimension length
        fvec = np.linspace(-5000,5000,len(S.T))

        # Db scale
        dB = 10.0*np.log10(S)
        plt.figure(figsize = ((13,8)))
        # Use the median noise floor as the lowest color to make it easier to distinguish the differents signals
        # Plotting
        plt.pcolormesh(fvec,tvec,dB,vmin = np.nanmedian(dB))
        plt.title('Frequency range %sMhz'%(self.radio_frequency/1e6), fontsize = '20')
        plt.ylabel('Time(s)',fontsize = '15')
        plt.xlabel('Frequency(Hz)',fontsize = '15')
        zcol = plt.colorbar()
        zcol.set_label('dB')
        plt.show()
        plt.close()



# Change filter to cutoff frequency at 4000 to be sharper around 5kHz
n,h1 = hann_window(4000,int(4e3))
signal1 = radio_station(z,11.53e6,h1,'turkish dance channel?')
signal2 = radio_station(z,11.66e6,h1, 'Canadian? talking about prince harry')
signal3 = radio_station(z,13.61e6,h1 , 'Arabic')
signal4 = radio_station(z,11.935e6,h1, 'Islamic call to prayer?')
signal5 = radio_station(z,14.24e6,h1, 'Russian? speaking some english')
signal6 = radio_station(z,14.89e6,h1, 'task11 frequency')

# Plot their spectrograms
signal1.spectrogram()
signal2.spectrogram()
signal3.spectrogram()
signal4.spectrogram()
signal5.spectrogram()
signal6.spectrogram()

# write out soundfile for signals
signal1.writefile()
signal6.writefile()


>>File 11530000.wav has been written
>>File 14890000.wav has been written

```

Here i did rest of the task in a class so that i could make each station its own object with dynamic spectrum and writefile(.wav) properties. Here the signal z will be processed by using the code given in the example and the low pass filter already made. Here the code shifts our center frequency of the signal to origin(DC) and re-sampled to 10kHz. Then we remove the negative frequencies and filter away the DC and some other frequencies lower then 50 Hz. Then we take the the signal back from frequency domain and into time domain, we now have the processed signal as property in our object. When using the dynamic spectrum method on our object we get the image in figure(15) below (for signal 6). Both the dynamic spectrum and savefile method converts the processed signal to real values before plotting/ saving file. In figure 15 below FFT with 1000 points has been used.



Figur 15: Dynamic spectrum of the radio station 13.89MHz

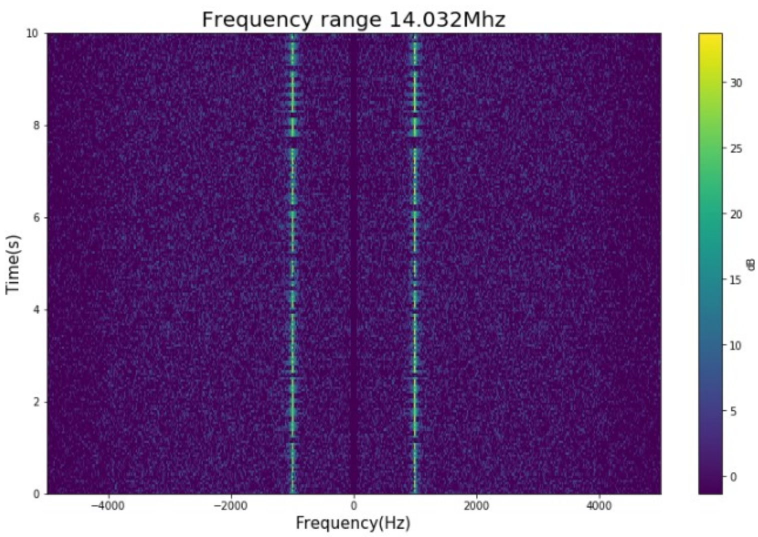
We can clearly see in figure(15) that the carrier frequency has been removed. After saving the file and playing it we get some noise, but seem to get in a russian? commercial. We might want to remove some noise to get a clearer sound.

Oppgave 10

10.1)

```
morse = radio_station(z,14.032e6,h1)
morse.spectrogram()

```

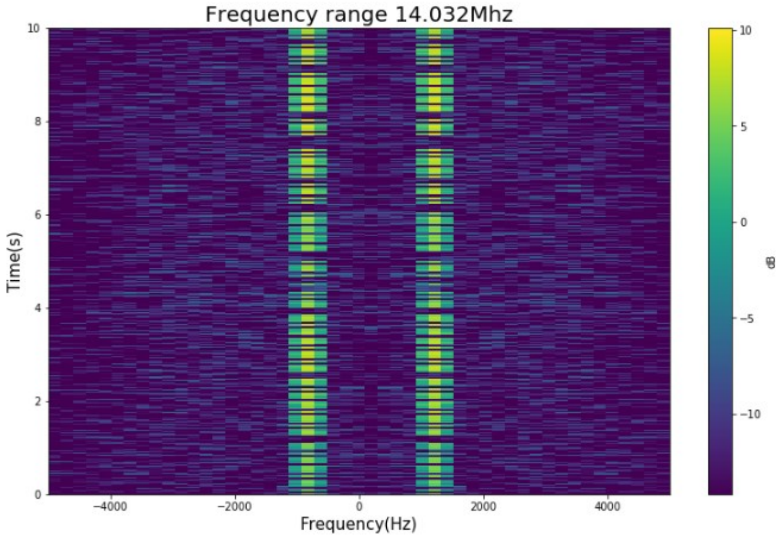
Here in figure(14) below we see one of the few Morse codes, the sound was good, but it is a bit hard to read it off the dynamic spectrum.



Figur 16: Dynamic spectrum of one of the Morse code at frequency 14.032MHz

Changed the length of the Fourier transform to 50 to get a sharper time as seen in figure(17) below.

```
morse.spectrogram(50)

```



Figur 17: Dynamic spectrum of one of the Morse code at frequency 14.032MHz with sharper time (N = 50)

Here we can see that we got less sharp frequency, but its easier to see where in time the sound stop and starts, this way its easier to read off the Morse code.

10.2)

```
morse.writefile()
>> File 14032000.wav has been written

```

According to the Morse decoder from https://morsecode.scphillips.com/labs/audio-decoder-adaptive/we get ”ETTTX HR SUNNY NICE AUTMN DAY HE I”.

10.3)

```
task103 = radio_station(z,11.66e6,h1)
task103.writefile()
>>File 11660000.wav has been written

```

Here the announcer says ”Britains prince Harry, the duke of Sussex mentioned his impending fatherhood for the first time on Tuesday....”

10.4)

Here i got the follow, where i labeled what i was able to hear.

```
signal1 = radio_station(z,11.53e6,h1,'turkish dance channel?')
signal2 = radio_station(z,11.66e6,h1, 'Canadian? talking about prince
    harry')
signal3 = radio_station(z,13.61e6,h1 , 'Arabic')
signal4 = radio_station(z,11.935e6,h1, 'Islamic call to prayer?')
signal5 = radio_station(z,14.24e6,h1, 'Russian? speaking some english'
    )
signal6 = radio_station(z,14.89e6,h1, 'task11 frequency')
```

10.5)

```
signal5 = radio_station(z,14.24e6,h1, 'Russian? speaking some english
    ')

```

Here we have a few seconds of silence and then we have somone say ”Heelllooo? my friend (Some name?) in moscow? Propagation very good thank thank for contact spasiba (thanks in russian?)”