Introduction: Load Modules and Import Audio

Load Modules

In [1]:
```
###########################################################################################
# uncomment the following 2 lines to reload the modules automatically,
# such that changes to Plot_functions.py are reloaded without restarting the kernel!
#
#%load_ext autoreload
#%autoreload 2
#
###########################################################################################

import matplotlib.pyplot as plt
import ipywidgets as widgets
import librosa
import IPython.display as ipd
import numpy as np
import parselmouth
import soundfile as sf
import bokeh
import sounddevice as sd
import time as clock
#from plot_functions import SCplot # imports the necessary plot functions
import Plot_functions as SCplot

from pathlib import Path
from scipy import signal, fft, ifft
```

Load and Playback Audio File

In [2]:
```
def import_sound_file(fileName, soundFolder=Path('../sounds/')):

    filePath = soundFolder / fileName

    audio1, fs = sf.read(filePath)
    print(fileName+" loaded with f_s ={}".format(fs))

    snd = parselmouth.Sound(str(filePath))
    return snd, audio1, fs, filePath

fileName = 'f116.wav'
# fileName = 'f216.wav'

snd, audio1, fs, filePath = import_sound_file(fileName)
ipd.Audio(filePath) # show audio player
```

```
f116.wav loaded with f_s =16000
```

Out[2]:

0:00 / 0:03

Experiment 1: Time-Domain Analysis

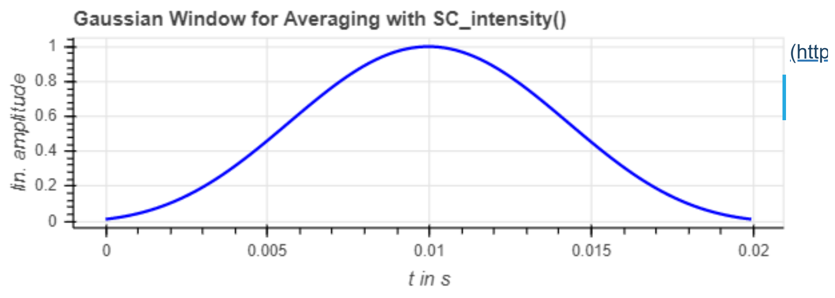
Short-Time Average Energy (Intensity)

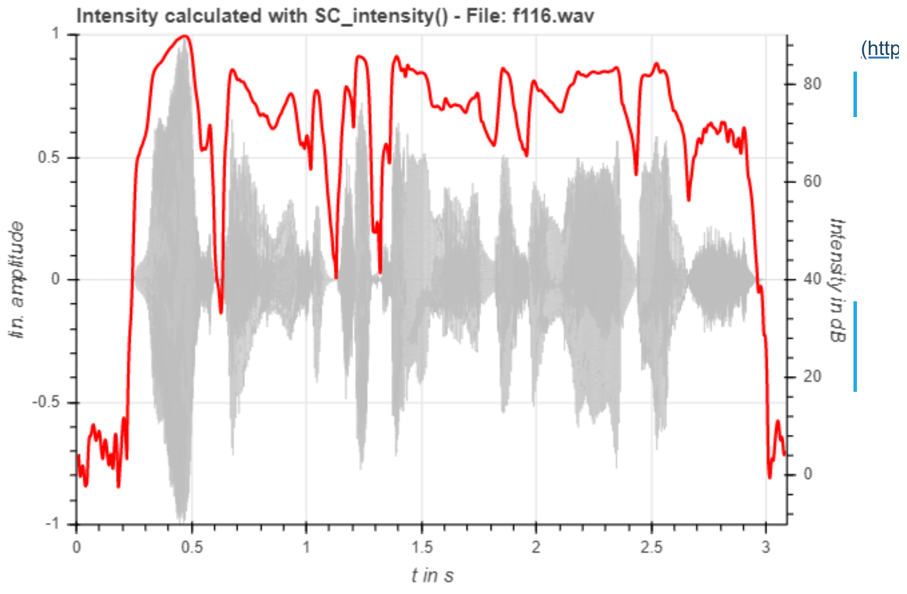
To calculate the short-time average intensity, we implement the function SC_intensity(). In this function, the signal gets averaged using a Gaussian window.

> TO DO: Convert the window length in milliseconds into a minimum pitch.

In [3]:
```
windowLength = 20 #millisecond
## TODO FOR STUDENT: Calculate the minimum pitch from the window length
#minimumPitch = ?
minimumPitch = 1000/windowLength
## END TODO FOR STUDENT

def SC_intensity(sound, minimumPitch, fs):
    #winLenEffective = np.round(3.2/minimumPitch * fs)  # Window length in samples; from Praat documentati
on; for pitch-synchronous intensity ripple
    winLenEffective = np.round(1/minimumPitch * fs)
    alpha = 2.5  # width factor alpha >= 0
    std = (winLenEffective-1)/(2*alpha)  # Matlab documentation

    win = signal.windows.kaiser(winLenEffective, 20/np.pi)

    sound = np.square(sound-np.mean(sound))

    intensity = np.convolve(sound, win, mode='valid') /np.sum(win)

    intensity = 10*np.log10(intensity/(4e-10)) # conversion to dB ; norm to (20 muPa)^2
#     intensity = 10*np.log10(intensity/(1e-12)) # conversion to dB; norm to 1 pW/m2
#     intensity = 20*np.log10(intensity/(2e-5)) # conversion to dB; norm to 20 muPa/m2
    print("SC_intensity: Intensity has {} samples".format(intensity.size))
    winLenEffectiveTime = winLenEffective / fs
    return intensity, win, winLenEffectiveTime

SC_intensity, gaussWin, winLenEffectiveTime = SC_intensity(np.squeeze(snd.values), minimumPitch, fs)

# Plot window function
plottitle = "Gaussian Window for Averaging with SC_intensity()"
dt_win = np.arange(0, gaussWin.size) / fs  # time axis for Gaussian Window
p_window = SCplot.get_plot_window(gaussWin, dt_win, plottitle)

# Plot intensity curve
snd_values = np.squeeze(snd.values)
dt_snd = np.arange(0, snd_values.size) / fs
dt_SC_intensity = np.arange(0,SC_intensity.size) / fs
dt_SC_intensity = dt_SC_intensity + winLenEffectiveTime/2 # shift to center the intensity curve bins in th
e windows

plottitle = 'Intensity calculated with SC_intensity() - File: ' + fileName
p_intensity = SCplot.get_plot_intensity(snd_values, dt_snd, SC_intensity, dt_SC_intensity, plottitle)

SCplot.plot_in_subplots(p_window, p_intensity)
```

```
SC_intensity: Intensity has 49105 samples
```

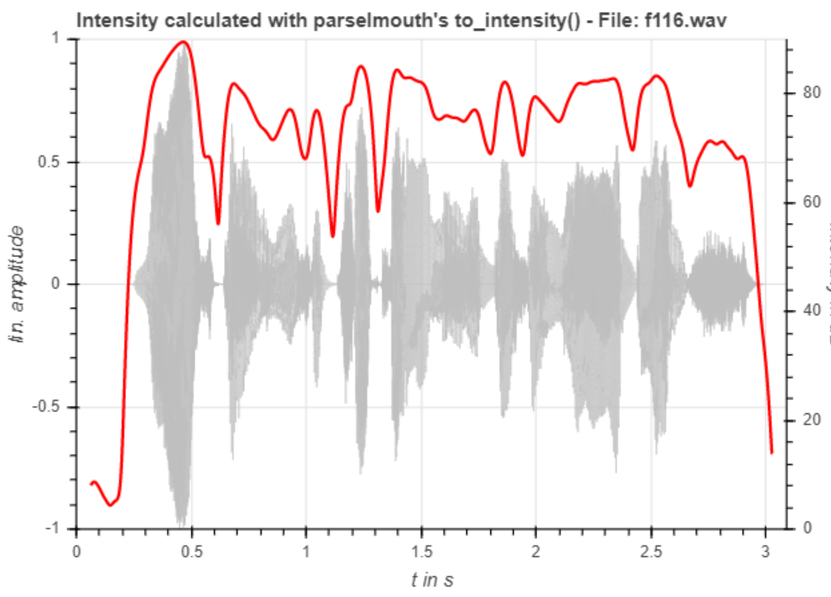




Also, the library praat-parselmouth provides us with functionality to calculate the intensity. To calculate the intensity with parselmouth, the member function to_intensity() is used, whith takes the minimum pitch as an input argument. To compare parselmouth's to_intensity() with our custom SC_intensity(), we use the same input parameters as before.

In [4]:
```
# PM_intensity = snd.to_intensity(minimum_pitch=minimumPitch, subtract_mean=False)  # intensity calculatio
n with parselmouth's function
PM_intensity = snd.to_intensity(minimum_pitch=minimumPitch, subtract_mean=False, time_step=1/fs)  # intens
ity calculation with parselmouth's function
print("PM_intensity: Intensity has {} samples".format(PM_intensity.get_number_of_frames()))

dt_PM_intensity = PM_intensity.x_grid()[:-1]
PM_intensity_val = np.squeeze(PM_intensity.values)
dt_snd = snd.x_grid()[:-1]

plottitle = "Intensity calculated with parselmouth's to_intensity() - File: " + fileName
SCplot.get_plot_intensity(np.squeeze(snd.values), dt_snd, PM_intensity_val, dt_PM_intensity, plottitle, sh
owPlot=True)
```

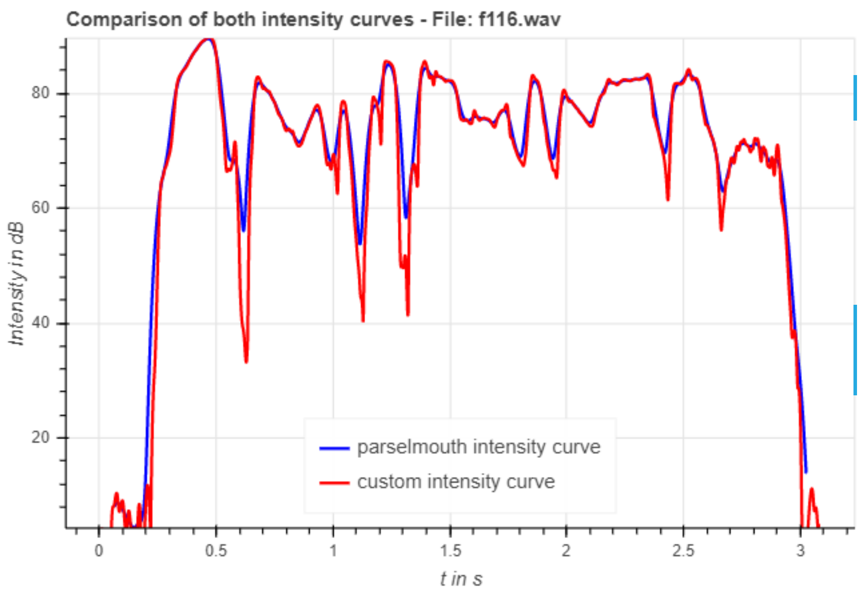
```
PM_intensity: Intensity has 47377 samples
```



(http

Compare the two intensity curves in the following plot:

In [5]:
```
# plot the 2 intensity curves in one plot
plottitle = "Comparison of both intensity curves - File: " + fileName
SCplot.plot_two_intensity_curves(dt_PM_intensity, PM_intensity_val, dt_SC_intensity, SC_intensity, plottit
le)
```



(http

Task 1: Describe the key differences of the intensity curve calculated with your own function SC_intensity() to parselmouth's to_intensity().

Expected Answers:

- different value range
- parselmouths intensity is more sparsely sampled
- parselmouths intensity has not the same time range as the audio file, whereas the custom intensity covers the whole time range of the audio file

Task 2: Is there a way to alter the intensity curve calculated with your own function SC_intensity(), such that it matches the intensity curve calculated with parselmouth better?

> Hint: see https://www.fon.hum.uva.nl/praat/manual/Sound__To_Intensity___.html (https://www.fon.hum.uva.nl/praat/manual/Sound__To_Intensity___.html)
>
> **Expected Answer:**
>
> - the student should recognize that parselmouth uses an effective window length that is 3.2 times the original window length
> - the student should modify the function SC_intensity() such that it uses this effective window length

Experiment 2: Frequency-Domain Analysis

Load and Playback Audio File

In [6]:
```
# Choose an Audio File
fileName = 'f116.wav'
#fileName = 'f216.wav'
#fileName = 'a_8000.wav'
#fileName = '1000hz_3sec.wav'

snd, audio1, fs, filePath = import_sound_file(fileName)
ipd.Audio(filePath) # show audio player
```

```
f116.wav loaded with f_s =16000
```

Out[6]:

0:00 / 0:03

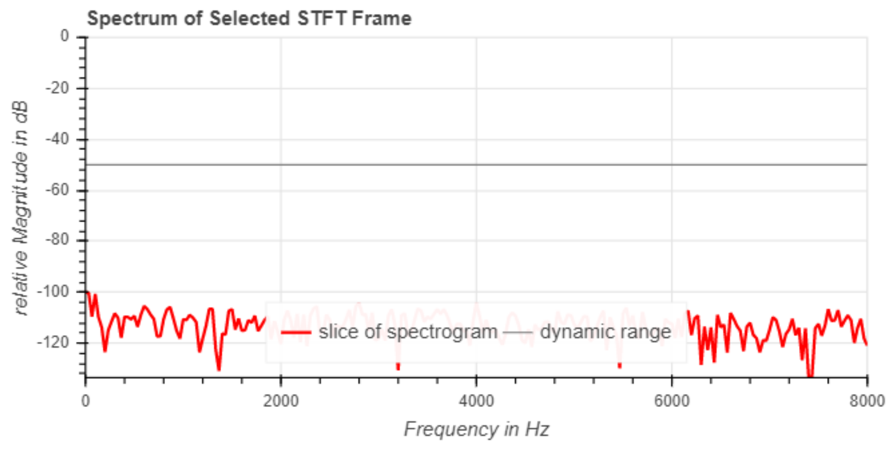
Wide- and narrow-band spectrograms

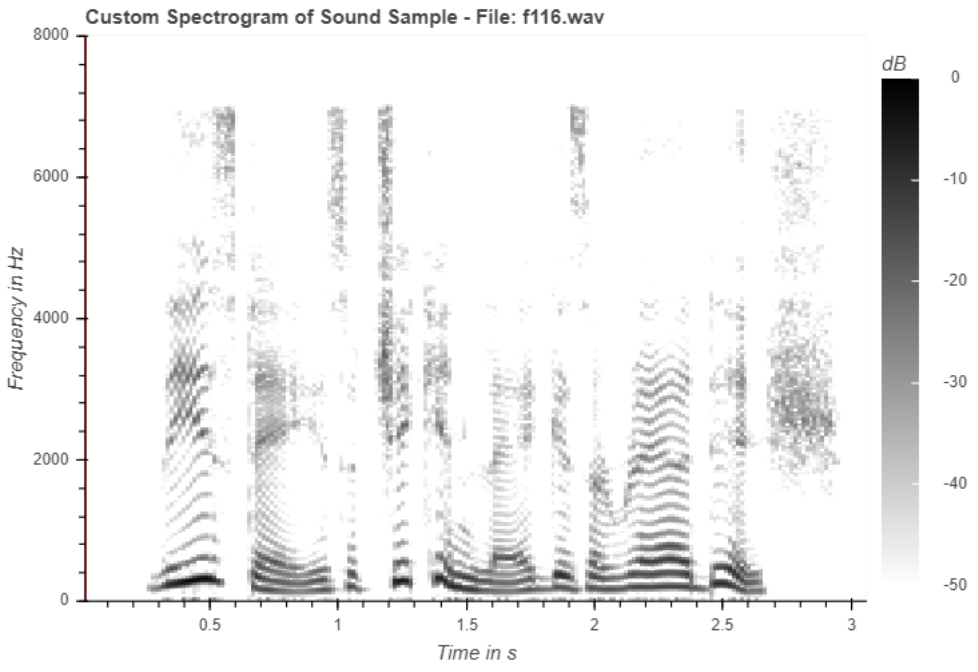
First, we start by calculating a spectrogram using the method scipy.signal.spectrogram().

In [7]:
```
windowlengthSec = 30 #ms
windowlength = np.round(fs * windowlengthSec/1000).astype(int)
#windowlength = 2048
print('Window Length in samples:', windowlength)
#overlap = windowlength-1
overlap = np.round(windowlength / 2)

#window = 'hann'
std = (windowlength - 1)/(2*2.5)  # Matlab documentation
window = ('gaussian', std)

SC_fVec, SC_tVec, SC_spectroData = signal.spectrogram(audio1, fs=fs, window=window, noverlap=overlap, nper
seg=windowlength, return_onesided=True, scaling='spectrum', mode='magnitude')

SC_spectroDataDB = 20*np.log10(SC_spectroData / np.max(SC_spectroData))

# plot interactive spectrogram
plottitle = "Custom Spectrogram of Sound Sample - File: " + fileName
SC_timeWidget = SCplot.plot_interactive_spectrogram(SC_spectroDataDB, SC_tVec, SC_fVec, plottitle)
widgets.HBox([SC_timeWidget])
```

```
Window Length in samples: 480
```

(https://bokeh.org/)

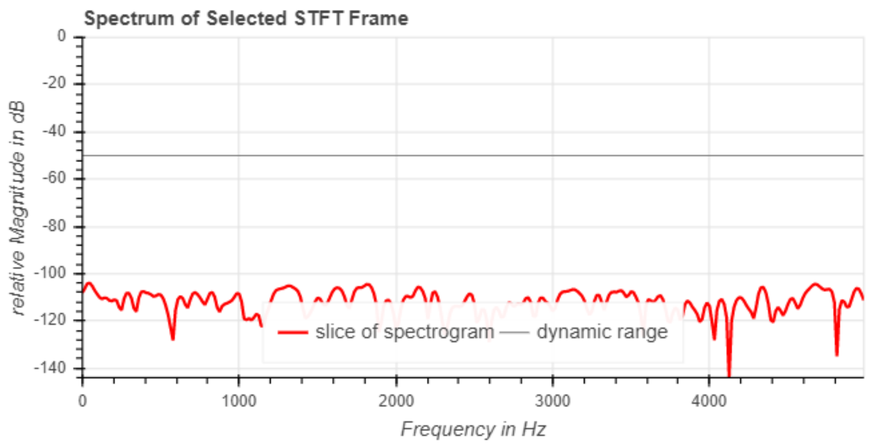


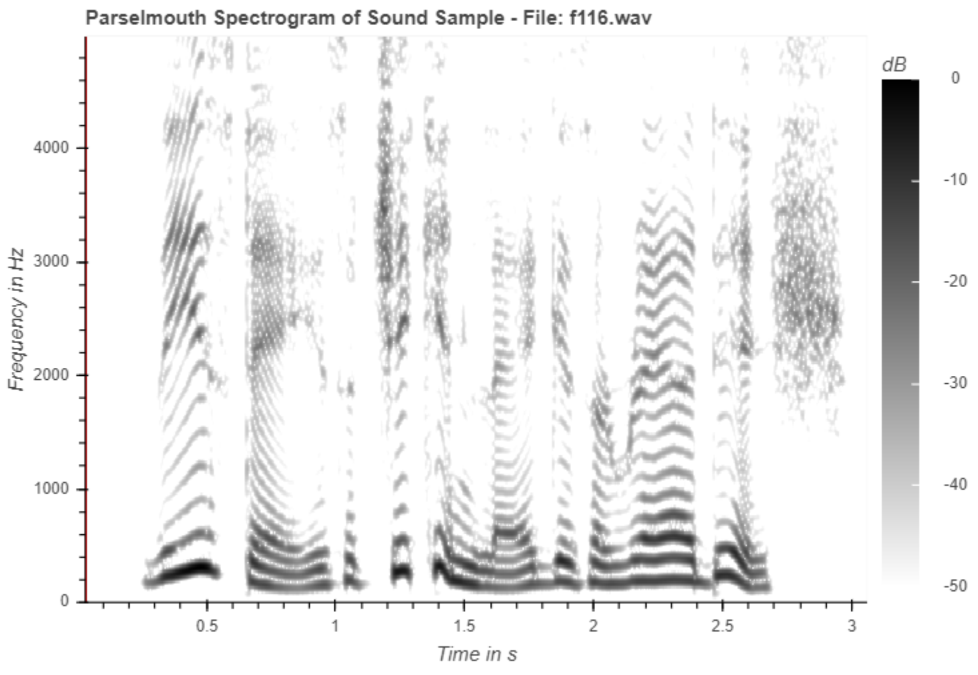


Now we use parselmouths to_spectrogram() to calculate a spectrogram. For Plotting the spectrogram, we use our custum function plot_interactive_spectrogram().

In [8]:
```
windowlengthSec = 30 #ms
maximumFrequency = 5000

def PM_get_spectrogram(snd, windowLengthMS, maximumFrequency):
    spectrogram = snd.to_spectrogram(window_length=windowLengthMS/1000, maximum_frequency=maximumFrequency
)
    PM_spectroData = spectrogram.values
    PM_tVec = spectrogram.ts()
    PM_fVec = spectrogram.ys()
    PM_spectroDataDB = 10*np.log10(PM_spectroData / np.max(PM_spectroData))
    return PM_spectroDataDB, PM_tVec, PM_fVec

PM_spectroDataDB, PM_tVec, PM_fVec = PM_get_spectrogram(snd, windowlengthSec, maximumFrequency)

plottitle = "Parselmouth Spectrogram of Sound Sample - File: " + fileName
PM_timeWidget = SCplot.plot_interactive_spectrogram(PM_spectroDataDB, PM_tVec, PM_fVec, plottitle)
widgets.HBox([PM_timeWidget])
```

(https://bokeh.org/)





f0 and Formants

To analyze the Formants, we use praat-parselmouths formant analysis methods.

Select sound file to analyze:

In [9]:
```
# Choose an Audio File
fileName = 'f116.wav'
#fileName = 'f216.wav'
#fileName = 'a_8000.wav'
#fileName = '1000hz_3sec.wav'

snd, _, fs, filePath = import_sound_file(fileName)
ipd.Audio(filePath) # show audio player
```

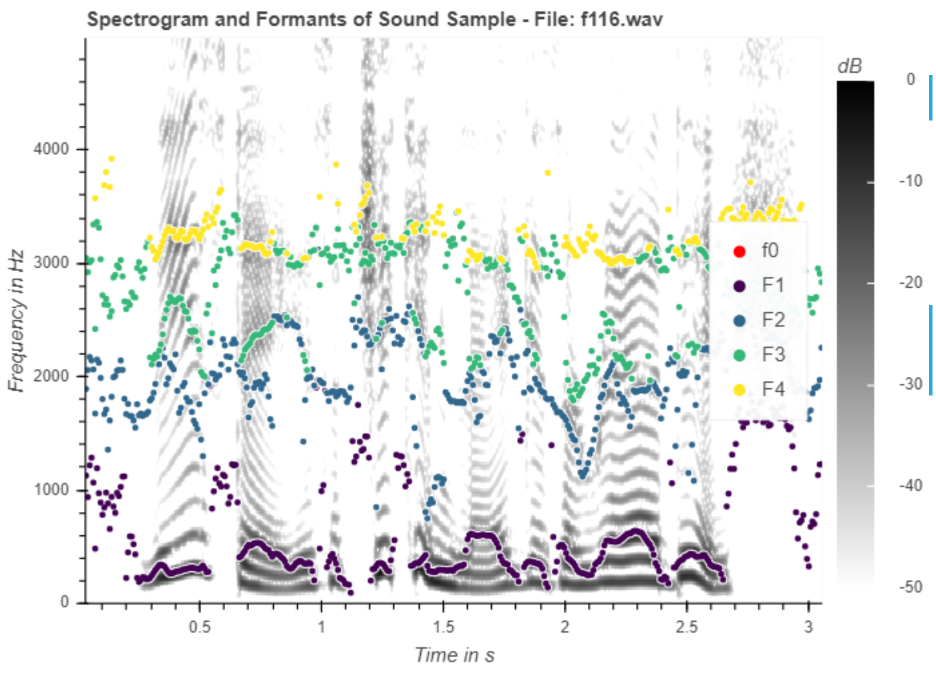
```
f116.wav loaded with f_s =16000
```

Out[9]:

0:00 / 0:03

Firstly, we analyse the formants F1 ... F4 only.

In [10]:
```
windowLength = 30 # ms
maxNumberFormants = 4
maxFormantFreq = 4000

PM_formants = snd.to_formant_burg(maximum_formant=maxFormantFreq, window_length=windowLength/1000,
                                  max_number_of_formants=maxNumberFormants)

formant_tVec = PM_formants.ts()
formantValues = np.zeros((maxNumberFormants,formant_tVec.size))

for timeIdx, time in enumerate(formant_tVec):
    for formantIdx in range(maxNumberFormants):
        formantValues[formantIdx,timeIdx] = PM_formants.get_value_at_time(formant_number=formantIdx+1, tim
e=time)

spectroDataDB, spec_tVec, spec_fVec = PM_get_spectrogram(snd, 30, maximumFrequency)

# plot spectrogram with formants
plottitle = "Spectrogram and Formants of Sound Sample - File: " + fileName
SCplot.plot_spectrogram_with_formants(spectroDataDB, spec_tVec, spec_fVec, formantValues, formant_tVec, pl
ottitle)
```



(http

Now, we analyse the fundamental frequency f0 and additionally the formants F1, ..., F4.

Select sound file to analyze:

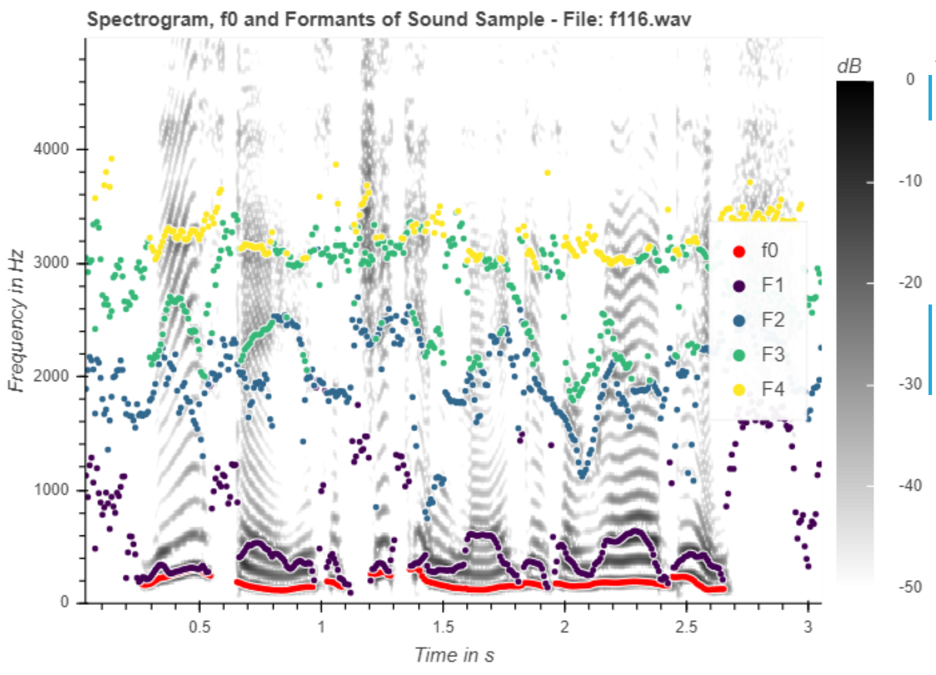
In [11]:
```
# Choose an Audio File
fileName = 'f116.wav'
#fileName = 'f216.wav'
#fileName = 'a_8000.wav'
#fileName = '1000hz_3sec.wav'

snd, _, fs, filePath = import_sound_file(fileName)
ipd.Audio(filePath) # show audio player
```

```
f116.wav loaded with f_s =16000
```

Out[11]:

0:00 / 0:03

In [12]:
```
pitchLo = 75 #Hz
pitchHi = 400 #Hz
pitchTimeStep = 30 #ms

PM_pitch = snd.to_pitch(pitch_floor = pitchLo, pitch_ceiling=pitchHi)

pitch_tVec = PM_pitch.ts()
pitchValues = np.zeros_like(pitch_tVec)

for timeIdx, time in enumerate(pitch_tVec):
    pitchValues[timeIdx] = PM_pitch.get_value_at_time(time=time)

# plot spectrogram with f0 and formants
plottitle = "Spectrogram, f0 and Formants of Sound Sample - File: " + fileName
SCplot.plot_spectrogram_with_formants(spectroDataDB, spec_tVec, spec_fVec, formantValues, formant_tVec,plo
ttitle,
                                      pitchValues=pitchValues, pitch_tVec=pitch_tVec)
```



(http

Experiment 3: Estimation of Vocal Tract using Cepstrum and LPC

LPC: Levinson Durbin Algorithm

The Levinson.Durbin-Algorithm is an elegant prcedure to calculate the LPC-coefficients recursively. This implementation uses the whole signal for processing.

Task 3.1: What kind of soundfile is suitable for LPC analysis? How can sentences be analysed?

Estimating the Vocal Tract Filter of speech signals (more generally: seperating the source signal from the transfair path) requires a stationary signal. If the transfair path changes inside the signal to analyse, the results are not representative. Therefore, sentences have to be buffered in order to analyse them!

Choose a suitable sound file for the Levinson-Durbin-Algorithm from the code lines below, or choose another from the file directory. You can listen to a file by uncommenting the related line in the code below.

Keep in mind that the whole signal is used for this particular implementation of the LPC

In [13]:
```
# Choose an Audio File

#fileName = 'f116.wav'
#fileName = 'f216.wav'
fileName = 'a_8000.wav'
#fileName = '1000hz_3sec.wav'

_, sig, fs, filePath = import_sound_file(fileName)
ipd.Audio(filePath) # show audio player
```

```
a_8000.wav loaded with f_s =8000
```

Out[13]:

0:00 / 0:00

Now set the LPC order below ("lpcOrder"). If there are confusingly many coefficients plotet, choose a suitalbe number of coefficients to display ("coefficientBoundary"). Run the code and take a look at the results!

You have the option to implement a pre-emphasis filter. Therefore call the function "pre_emphasis_filtering(sig, fs)".

Task 3.2 Implement the pre-emphasis filter and compare the results. What does a pre-emphasis filter do, and why does the algorithm perform better/worse?

Comparing the results, there is a significant drop to high frequencies in the vocal tract spectrum of the estimation without pre-emphasis. Also the autocorrelation of the pre-emphasis filtered signal drops to much smaller values. The results of the estimation with pre-emphasis seem more relyable.

The pre-emphasis filter whitens the spectrum and therefore decorrelates the signal (for Lags bigger than 0). The Levinson-Durbin Algorithm is constructed to perform best with decorrelated signals. In case of speech, the main signal energy is located in lower frequencies. Therefore the pre-emphasis filter is a low-cut filter, in this case a low-cut filter first order.

In the resulting plot below, the iterations of the Levinson-Durbin algorithm can be stepped through. The approach of the coefficients can be observed.

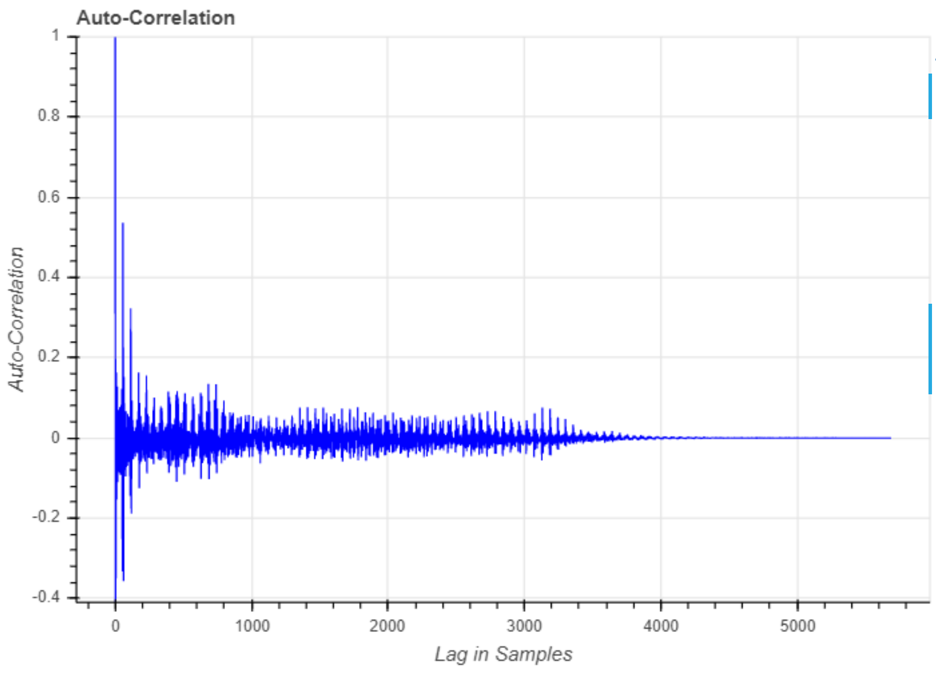
Task 3.3 Set the LPC order to a large number (arround 1000) and limit the displayed coefficients to a reasonable amount. Step through the iterations and note any irregularities, fore the estimation without pre-emphasis filtering.

no pre-emphasis: The estmation of the vocal tract is not representing the vocal tract for a large number of iterations. Some filter coefficient pole pairs converge from outside the unit circle and are therefore instable for the converging duration.

Task 3.4 How can you explain the prominent peaks in the estimated vocal tract spectrum for high LPC orders?

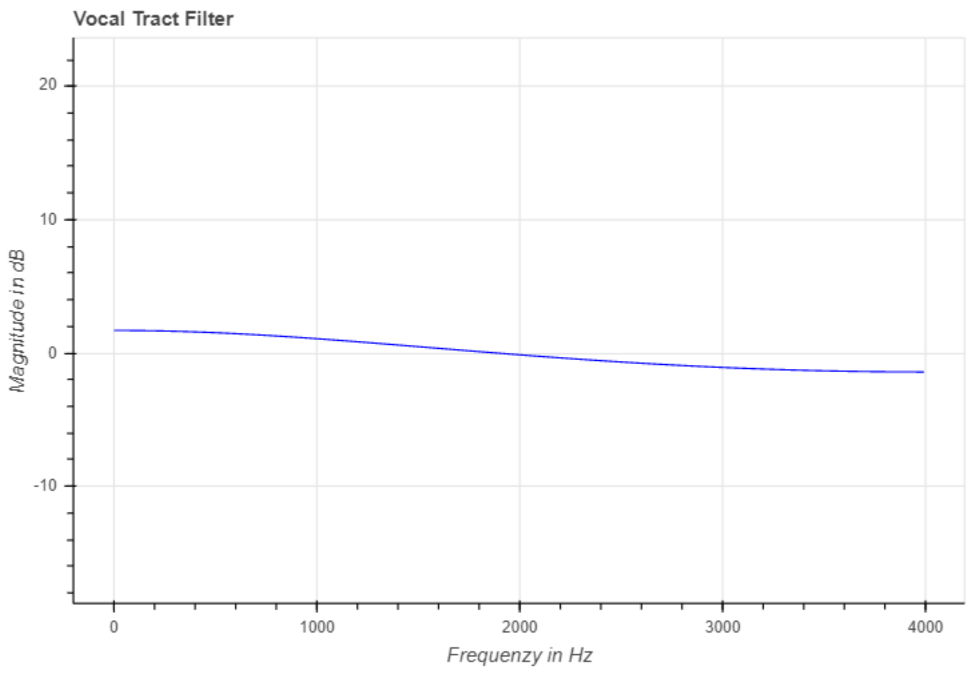
These peaks represent the harmonics of the speech signal. For too large numbers of coefficients these also get estimated.

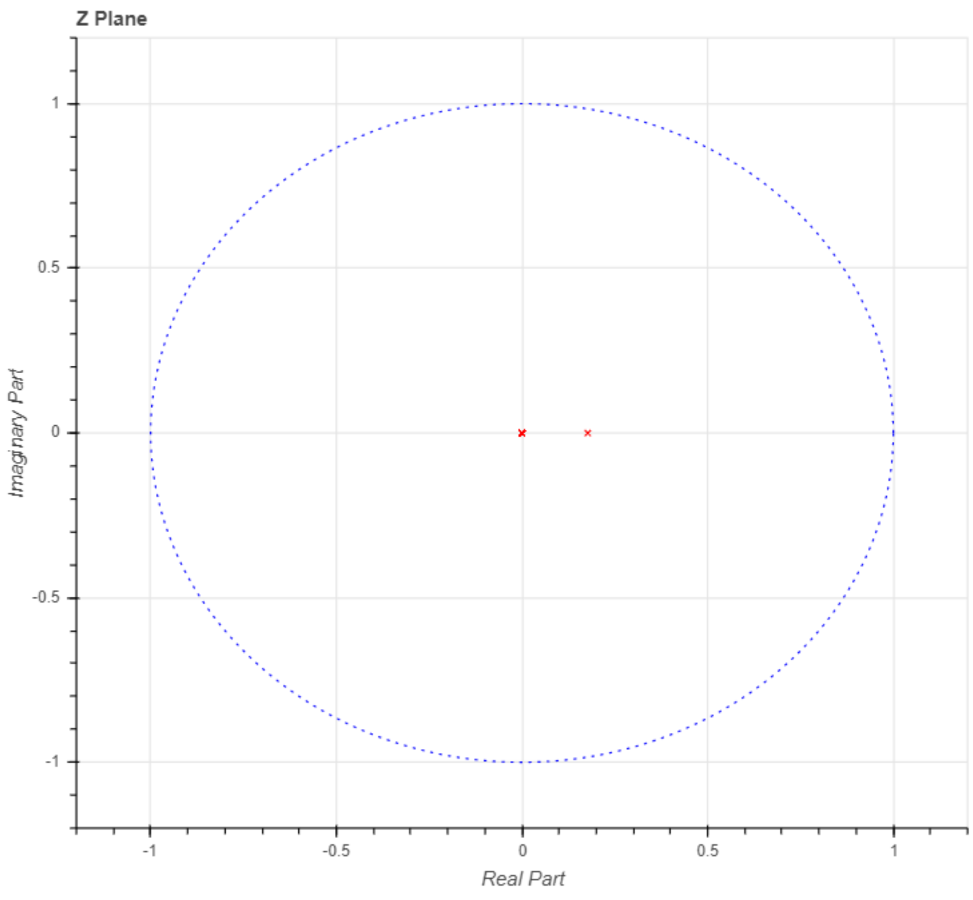
In [14]:
```
def autocorr(x, norm=True):
    result = np.correlate(x, x, mode='full')
    result = result[result.size // 2:]
    if norm:
        result = result/result[0]
    return result

def pre_emphasis_filtering(sig, fs):
    fPreEmph = 10
    # alpha is calculated as in egifa!
    alpha = np.exp(-2*np.pi*fPreEmph/fs)
    # 1. order Low-Cut
    sigPreEmphasis = signal.lfilter(np.append([1], -alpha),[1],sig)
    return sigPreEmphasis

##############################################################################################
#################

lpcOrder = 1000 # LPC order in samples
coefficientBoundary = lpcOrder # max number of filter coefficients in plot

E = np.zeros(lpcOrder-1) # define size of error vector
K = np.zeros(lpcOrder-1) # define size of reflection coefficient vector
a_all = np.zeros([lpcOrder-1, lpcOrder]) # define size of filter coefficient matrix (coefficient vector ea
ch row)

sigPreEmphasis = pre_emphasis_filtering(sig, fs) # pre-emphasis filtering to whiten the speech signal befo
re analyzing it

R = autocorr(sigPreEmphasis) # calc. the autocorrelation of the signal (only significant part)
#R = autocorr(sig) # calc. the autocorrelation of the signal (only significant part)
SCplot.plot_autocorr(R, 'Auto-Correlation') # plot auto-correlation

K[0] = R[1]/R[0] # initialize reflection coefficient
a = K[0] # initialize lpc coefficient
a_all[0,0] = 1
a_all[0,1] = -a # save initial lpc coefficient as IIR filter coefficient
E[0] = R[0] # initialize error

for i in range(1,lpcOrder-1):

    K[i] = (R[i+1] - np.dot(a, R[1:i+1])) / E[i-1] # calc new reflection coefficient

    a = np.append(K[i], a-K[i]*np.flip(a)) # calc new lpc coefficients

    E[i] = E[i-1] * (1-(abs(K[i])**2)) # calc new error

    a_all[i,0:i+2] = np.append([1], -np.flip(a)) # save current lpc coefficients as IIR filter coefficient
s


#a=a_all[-1]

#SCplot.plot_filter(a, fs, 'Vocal Tract Filter - Frequenzy Response')
#SCplot.plot_zplane([1], a, 'Vocal Tract Filter - Z Plane')

SC_iterationWidget = SCplot.plot_interactive_filter_zplane(a_all, coefficientBoundary, fs, 'Vocal Tract Fi
lter')
widgets.HBox([SC_iterationWidget])
```


Auto-Correlation
Auto-Correlation
Lag in Samples
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
0
1000
2000
3000
4000
5000

(http

(https://bokeh.org/)





In the section below the time signals and spectra of the original signal and the signal filtered by the estimated vocal tract (so the estimated glottis signal!) are displayed.

With the plot above, select a reasonable iteration number and number of coefficients. By writing these parameters into the section below ("selectedIteration", "limitCoefficients"), the figures correspond to the selected filter.

Task 3.5 Zoom into the Glottis time signal and measure the period using the crosshairs. Then calculate the frequency of the Glottis signal and compare it to the frequency from the vocaltract estimation.

measured from Glottis signal: T = 0.008s -> f0 = 125Hz

measured from Vocal Tract: f0 = 140Hz

The measurement from the glottis signal is a local observation in contrast to the global observation of the estimated vocal tract.

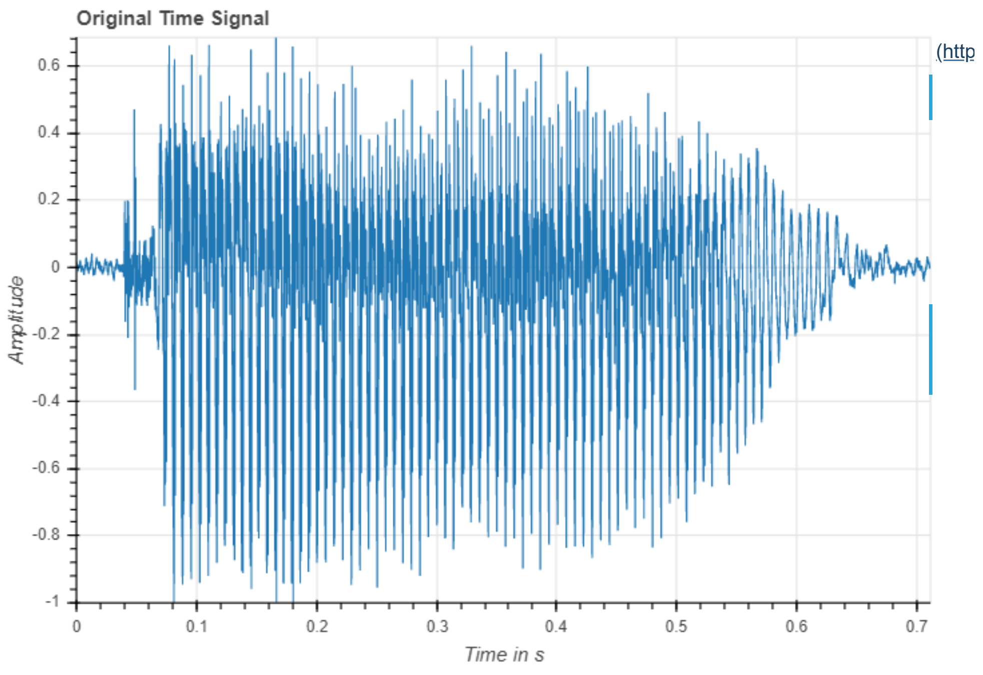
Task 3.6 Think of applications for low, midrange and high LPC orders (and therefore number of coefficients)!

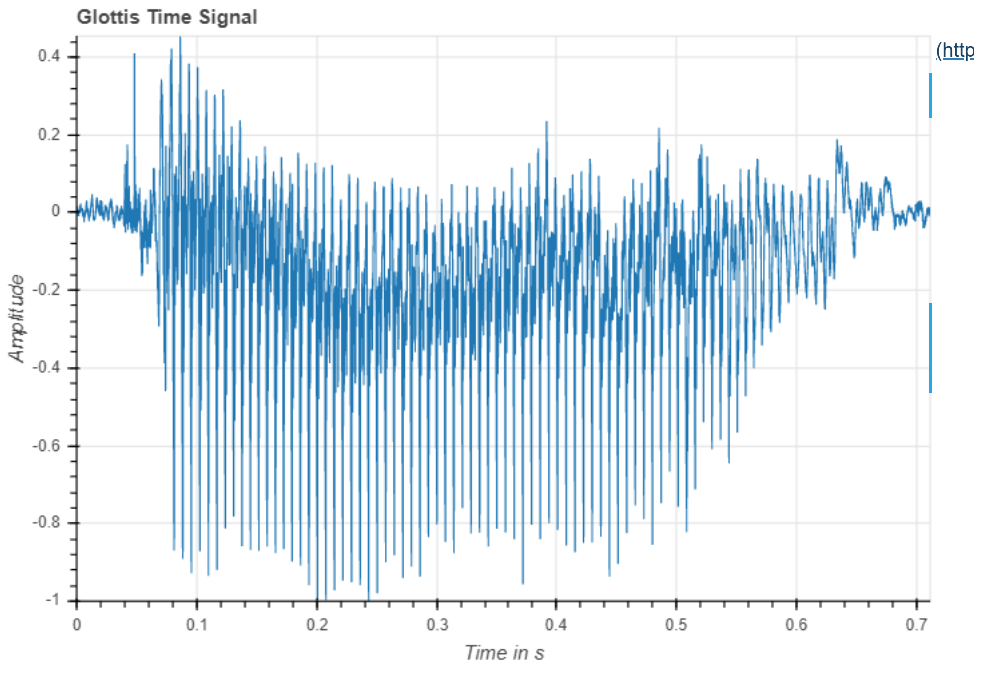
a low number of coefficients means high calculating efficiency but low vocal tract resolution -> vox box

a medium number of filter coefficients comes with larger computational cost but also better represents the real vocal tract -> telephone

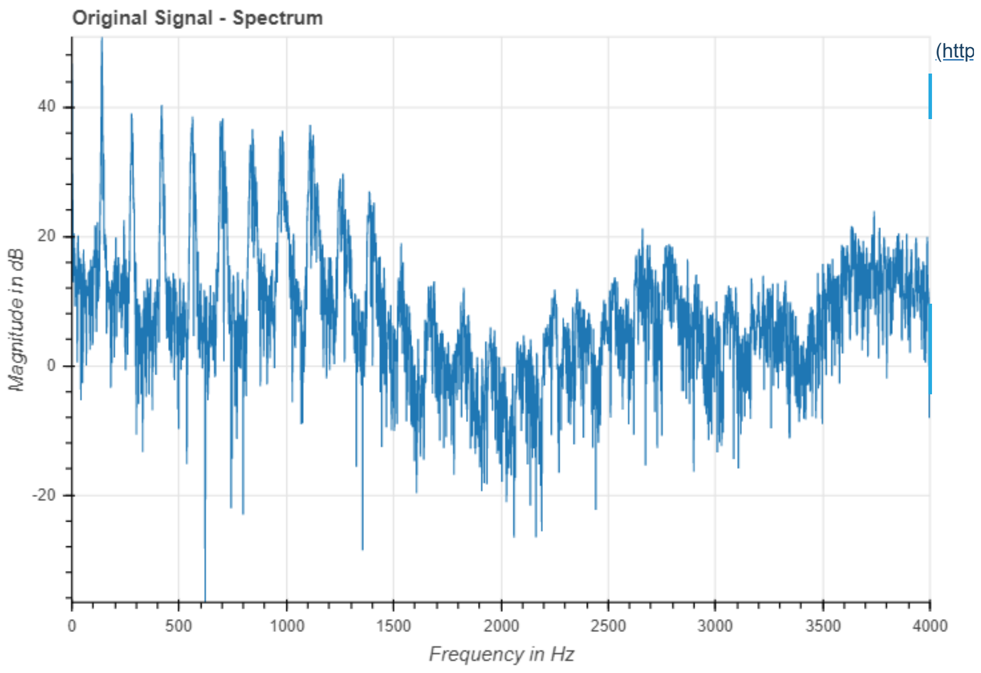
In [15]:
```
def get_spectrum(signalData, NFFT, window = True, logarithm = True):
    dataVec = np.zeros(NFFT)
    if len(signalData) > NFFT:
        dataVec = signalData[0:NFFT-1]
        win = signal.hann(NFFT)
    else:
        dataVec[0:len(signalData)] = signalData
        win = signal.hann(len(signalData))
        win = np.append(win, np.zeros(NFFT-len(signalData)))
    if window:
        dataVec = dataVec*win
    if logarithm:
        spectrum = 20*np.log10(abs(fft(dataVec)))
    else:
        spectrum = abs(fft(dataVec))
    return spectrum[0:round((len(spectrum)+1)/2)]

##########################################################################################################
#################

selectedIteration = 37 # select reasonalbe filter coefficients from the interactive plot
limitCoefficients = 0 # limit number of filter coefficients (if 0, no limit)

selectedFilter = np.trim_zeros(a_all[selectedIteration], 'b')
if limitCoefficients < len(selectedFilter) and limitCoefficients > 0:
    selectedFilter = selectedFilter[0:limitCoefficients]

print(selectedFilter)

filteredSignal = signal.lfilter([1], selectedFilter, sig)
filteredSignal = filteredSignal / max(abs(filteredSignal)) # normalize
inverseFiteredSignal = signal.lfilter(selectedFilter, [1], sig)
inverseFiteredSignal = inverseFiteredSignal / max(abs(inverseFiteredSignal)) # normalize

SCplot.plot_time_signal(sig, fs, 'Original Time Signal')
#SCplot.plot_time_signal(filteredSignal, fs, 'Vocal Tract Time Signal')
SCplot.plot_time_signal(inverseFiteredSignal, fs, 'Glottis Time Signal')

sigSpectrum = get_spectrum(sig, len(sig), window=True)
filteredSignalSpectrum = get_spectrum(filteredSignal, len(filteredSignal), window=True)
inverseFiteredSignalSpectrum = get_spectrum(inverseFiteredSignal, len(inverseFiteredSignal), window=True)

freqVector = np.linspace(0, fs/2, round(len(sig)/2+1))

SCplot.plot_spectrum(get_spectrum(sig, len(sig), window=True), freqVector, 'Original Signal - Spectrum')
#SCplot.plot_spectrum(get_spectrum(filteredSignal, len(filteredSignal), window=True), freqVector, 'Vocal T
ract - Spectrum')
SCplot.plot_spectrum(get_spectrum(inverseFiteredSignal, len(inverseFiteredSignal), window=True), freqVecto
r, 'Glottis - Spectrum')
```

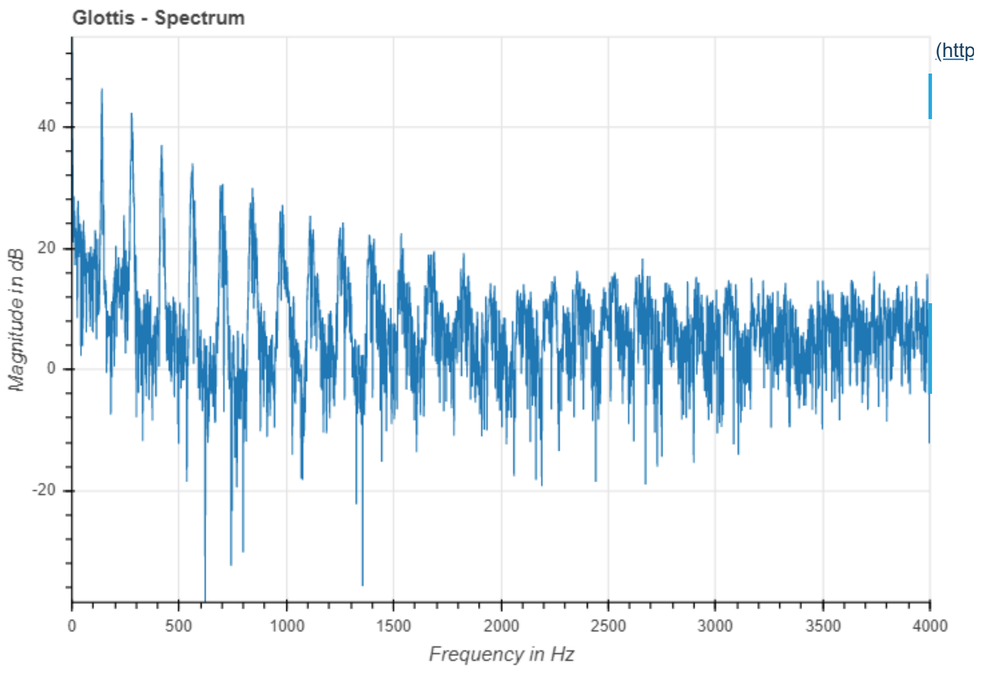
```
[ 1.00000000e+00 -9.67441394e-02 -4.45716380e-01  3.94131628e-01
  5.21319290e-01  1.64236826e-01 -4.03473498e-01  6.85512650e-02
  3.84039096e-01 -2.65817990e-02 -4.11988494e-03 -5.48674537e-02
  5.02565771e-02  5.48880921e-02 -1.97633619e-03 -2.15417626e-02
  9.26971299e-02 -3.43130199e-02 -4.04996077e-02  1.67644454e-01
  4.24183742e-02 -1.56001021e-02 -2.81740230e-02  8.02762395e-02
  1.33203959e-01 -9.78293474e-03  6.87312788e-02  6.75422693e-02
  4.07202455e-02  7.68109018e-02  1.27622494e-01  6.32992461e-02
  2.56211268e-02  8.08418472e-02  7.37089543e-02  6.35990567e-02
 -5.64377671e-04  2.92637692e-02  9.47647112e-02]
```





```
C:\Users\kraxi\anaconda3\lib\site-packages\ipykernel_launcher.py:13: DeprecationWarning: Using scipy.fft a
s a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.
  del sys.path[0]
```





Cepstrum Analysis

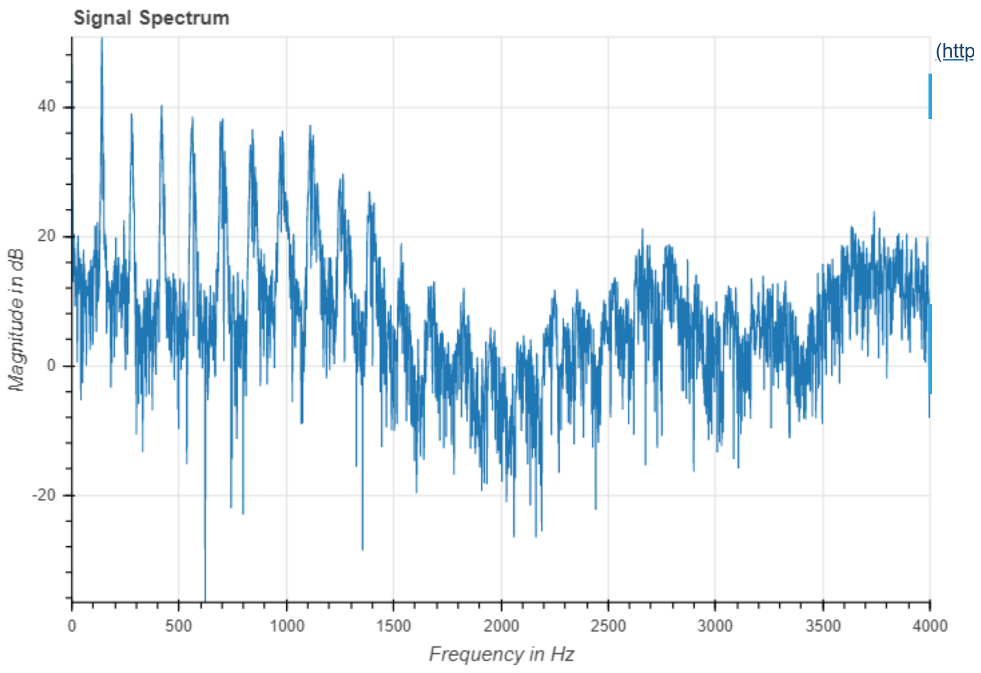
In this part the source, filter seperation is carried out by a cepstral analysis. Therefor the cepstrum of the signal is calculated (LINK to doku). In the cepstrum the slowly changing parts of the spectrum are located in the lower quefrencies and the fast oscillating parts in higher quefrencies. Therefore we can calculate the spectral envelope by liftering the cepstrum with a rectangular lifter.

Task 3.7 Choose the lifter length ("lifterLength") in samples so a reasonalbe vocal tract filter results.

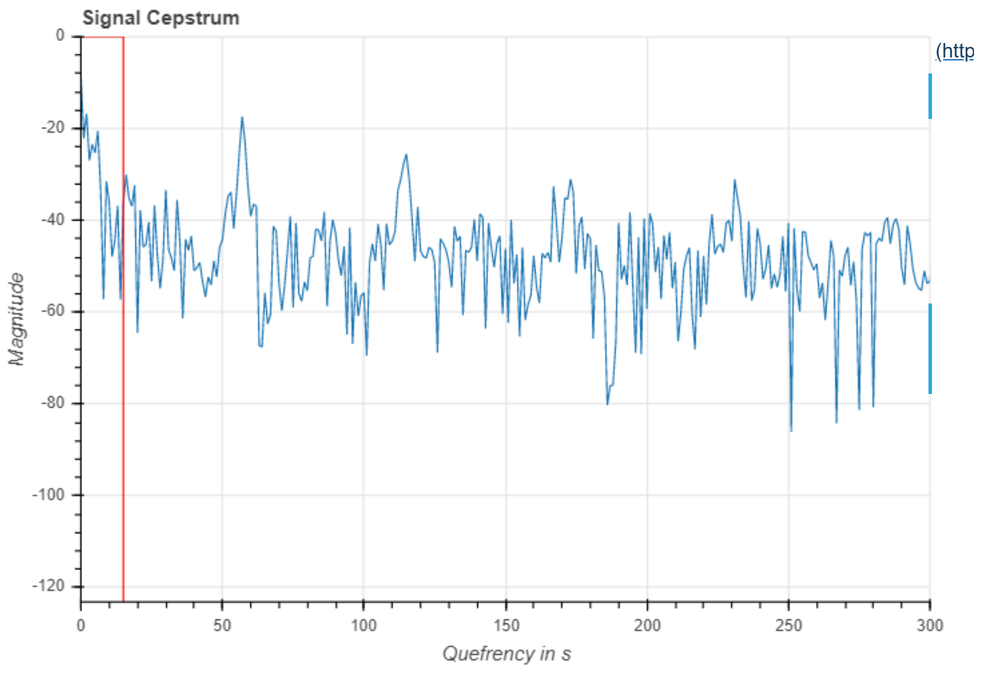
Task 3.8 Compare both vocal tract estimations and discus the differences.

In [16]:
```
def get_cepstrum(spectrum, logBase10 = True, mirrorSpectrum = False):
    print('before', len(spectrum))
    if mirrorSpectrum:
        spectrum = np.append(spectrum, np.flip(spectrum[0:-1]))
    print('after', len(spectrum))
    if logBase10:
        spectLog = np.log10(abs(spectrum))
    else:
        spectLog = np.log(abs(spectrum))
    #cepstrum = 4* abs(ifft(spectLog))**2
    cepstrum = abs(ifft(spectLog))
    return cepstrum

##########################################################################################################
#################

lifterLength = 15 # set length of cepstrum lifter

sigLength = len(sig)
fftLength = sigLength
timeVector = np.linspace(0, sigLength/fs, sigLength)
freqVector = np.linspace(0, fs/2, round(fftLength/2+1))

sigEmph = pre_emphasis_filtering(sig, fs) # pre-emphasis filtering to whiten the speech signal before anal
yzing it

SCplot.plot_spectrum(get_spectrum(sig, len(sig), window = True), freqVector, 'Signal Spectrum')

spectrum = get_spectrum(sig, len(sig), window = True, logarithm=False)
#spectrum = get_spectrum(sigEmph, len(sig), window = True, logarithm=False)

cepstrum = get_cepstrum(spectrum, mirrorSpectrum=True, logBase10=True)
SCplot.plot_cepstrum(20*np.log10(abs(cepstrum[0:round(len(cepstrum)/2+1)])), range(len(cepstrum[0:round(le
n(cepstrum)/2+1)])), lifterLength, 'Signal Cepstrum', lifterLP=True)
cepstrum[lifterLength:-lifterLength] = 0 #liftering

#inverseCepstrum = 10**(abs(fft(abs(cepstrum)**(1/2) / 4))) #for cepstrum calculated with log base of 10
inverseCepstrum = 10**abs(fft(abs(cepstrum)))
inverseCepstrum = inverseCepstrum[0:round(len(inverseCepstrum)/2)]


SCplot.plot_spectrum(20*np.log10(inverseCepstrum), freqVector, 'Vocal Tract Spectrum')

#gefiltert = np.divide(20*np.log10(spectrum), 20*np.log10(inverseCepstrum))
gefiltert = 20*np.log10(spectrum) - 20*np.log10(inverseCepstrum)

SCplot.plot_spectrum((gefiltert), freqVector, 'Glottis Signal Spectrum')
```

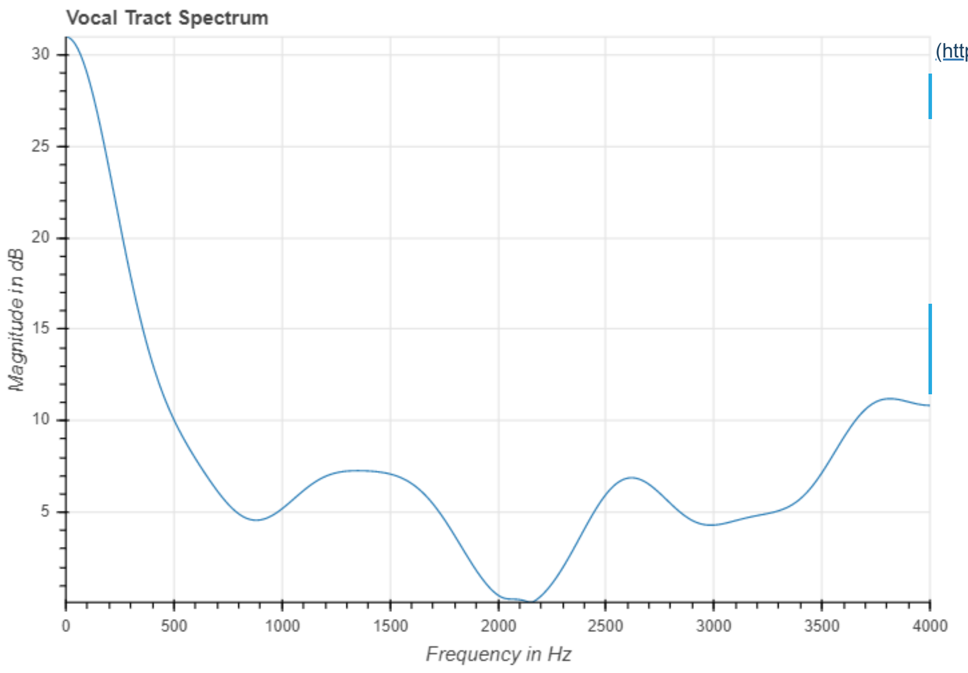
```
C:\Users\kraxi\anaconda3\lib\site-packages\ipykernel_launcher.py:13: DeprecationWarning: Using scipy.fft a
s a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.
  del sys.path[0]
```



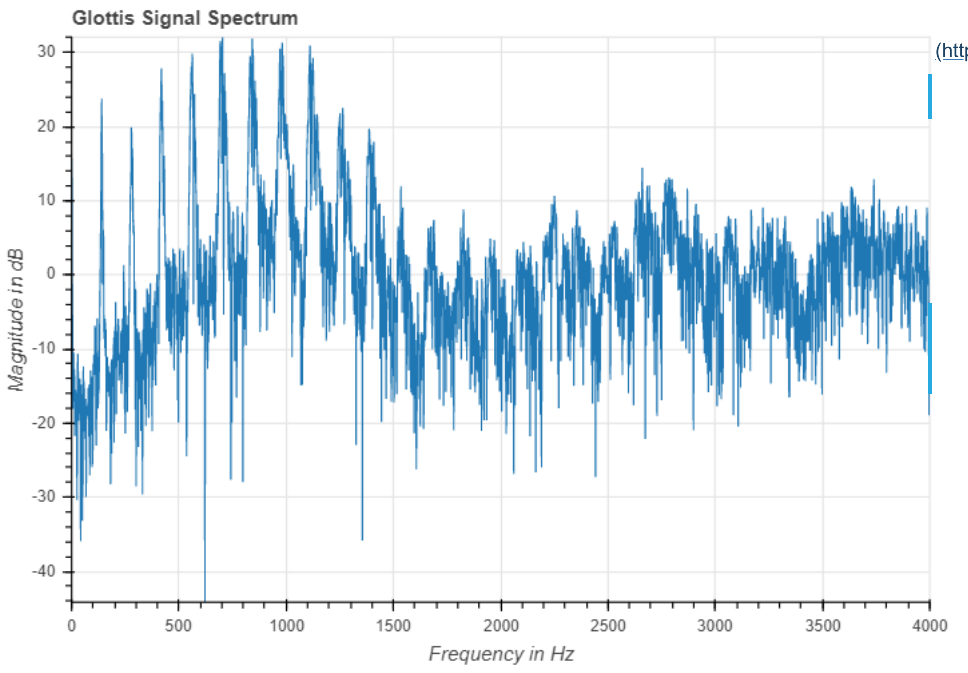
```
C:\Users\kraxi\anaconda3\lib\site-packages\ipykernel_launcher.py:15: DeprecationWarning: Using scipy.fft a
s a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.
  from ipykernel import kernelapp as app
C:\Users\kraxi\anaconda3\lib\site-packages\ipykernel_launcher.py:11: DeprecationWarning: scipy.ifft is dep
recated and will be removed in SciPy 2.0.0, use scipy.fft.ifft instead
  # This is added back by InteractiveShellApp.init_path()

before 2846
after 5691
```



```
C:\Users\kraxi\anaconda3\lib\site-packages\ipykernel_launcher.py:35: DeprecationWarning: Using scipy.fft a
s a function is deprecated and will be removed in SciPy 1.5.0, use scipy.fft.fft instead.
```


Vocal Tract Spectrum
Magnitude in dB
30
25
20
15
10
5
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency in Hz

(http


Glottis Signal Spectrum
Magnitude in dB
30
20
10
0
-10
-20
-30
-40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency in Hz

(http

Experiment 4: Formant Analysis

In this exercise you need to record your own speech-samples. Before recording is possible the input/output sounddevices have to be set. Use the first cell to display all possible sound-devices and select the wanted devices by assigneng the sounddevice-indices to the variables 'input_device' and 'output_device'.

Then you can use the given record, play and clear buttons to record your own speech-samples.

For this experiment please record 3 different versions of one sentence and analyze the time-domain signal and its formant and f0 structure (as already done in Experiment 2) and plot the results in separate cells. By plotting the results in separate cells it is possible to always use the same record button and audio-array for different recordings.

1.) Recording:

Record a standard sentence and plot the time-domain signal and the spectrogram/formant/f0 plot as you have already done in Experiment 2. For the time-domain signal use the provided function 'get_plot_time_domain_sig()'. The function-arguments are described in the corresponding function header. To analyze the spectrogram/frequency and f0 structure of the recorded sentence you can use the Parselmouth-analysis and the corresponding plot-functions of Experiment 2. Please plot the results in a separate Cell.

- Are there visible formant contours?
  - Answer: yes there should be visible formants assuming a standard sentence containing vowels was recorded.
- Can you distinguish between vowels, consonants and fricatives?
  - Answer: Yes vowels are distinguishable with a clear formant structure and fricatives don't show a clear formant-structure. For consonants like 'l', 'm' or 'n' there can be visible formant structures because they can also be used in a voiced way.

2.) Recording:

Record a so called Dada-sentence but try to keep the same pitch/pitch course as in the sentence before. The Dada-sentence should not consist of words but rather of sounds like 'da', 'la', 'fa' etc. (or maybe even a combination of meaningless sounds?). After recording please plot the time-domain signal and spectrogram/formant/f0-plot in a separate cell.

- Is the course of f0 the same as in the Recording before?
  - Answer: Should be the same if it has been possible to keep the same pitch as the sentence before, if not maybe try again.
- How do the formant-structures differ?
  - Answer: If only one sound e.g. 'da' was used the formant-structure should repeat it self independent of the course of f0. Small deviations in the formant-structures are possible due to a certain dependency of the formants towards f0.
- Are there any differences visible in the time-domain?
  - Answer: If a voiced sound like 'da' was used to record the sentence there are only voiced-segments and no "noisy" segments in the time-domain signal indicated by higher amplitudes due to the higher energy content.

3.) Recording:

Record the previous sentence but use a whispery voice and once more please plot the time-domain signal and the spectrogram/formant/f0-plot in a separate cell.

- How does the formant-structure differ in comparison to the first Recording?
    - Answer: Formant-Tracker results might vary more due to less energy in the speech signal. But some sort of Formant-structure should still be visible.
- How and why does the f0-course differ from the previous recordings?
    - Answer: F0-Tracking does not work. Whispery voice does not contain a f0 structure due to the fact that the glottis does not close properly when speaking with a whispery voice. The glottis closing impulses determine the f0 we percieve in speech signal. The missing or weakened glottis closing impulses lead to a missing f0 perception.

In [17]:
```
fs = 44100 #Hz

#show all possible sound-devices
sd.query_devices()
```

Out[17]:
```
   0 Microsoft Soundmapper - Input, MME (2 in, 0 out)
>  1 External Microphone (Conexant S, MME (2 in, 0 out)
   2 S/PDIF (M-Audio Fast Track Ultr, MME (2 in, 0 out)
   3 Line 3/4 (M-Audio Fast Track Ul, MME (2 in, 0 out)
   4 Line 1/2 (M-Audio Fast Track Ul, MME (2 in, 0 out)
   5 Multichannel (M-Audio Fast Trac, MME (2 in, 0 out)
   6 Line 5/6 (M-Audio Fast Track Ul, MME (2 in, 0 out)
   7 Microsoft Soundmapper - Output, MME (0 in, 2 out)
<  8 Line 1/2 (M-Audio Fast Track Ul, MME (0 in, 2 out)
   9 S/PDIF (M-Audio Fast Track Ultr, MME (0 in, 2 out)
  10 Line 3/4 (M-Audio Fast Track Ul, MME (0 in, 2 out)
  11 Lautsprecher (M-Audio Fast Trac, MME (0 in, 2 out)
  12 Line 5/6 (M-Audio Fast Track Ul, MME (0 in, 2 out)
  13 U28E570 (Intel(R) Display-Audio, MME (0 in, 2 out)
  14 Lautsprecher (Conexant SmartAud, MME (0 in, 2 out)
  15 Primärer Soundaufnahmetreiber, Windows DirectSound (2 in, 0 out)
  16 External Microphone (Conexant SmartAudio HD), Windows DirectSound (2 in, 0 out)
  17 S/PDIF (M-Audio Fast Track Ultra 8R), Windows DirectSound (2 in, 0 out)
  18 Line 3/4 (M-Audio Fast Track Ultra 8R), Windows DirectSound (2 in, 0 out)
  19 Line 1/2 (M-Audio Fast Track Ultra 8R), Windows DirectSound (2 in, 0 out)
  20 Multichannel (M-Audio Fast Track Ultra 8R), Windows DirectSound (2 in, 0 out)
  21 Line 5/6 (M-Audio Fast Track Ultra 8R), Windows DirectSound (2 in, 0 out)
  22 Primärer Soundtreiber, Windows DirectSound (0 in, 2 out)
  23 Line 1/2 (M-Audio Fast Track Ultra 8R), Windows DirectSound (0 in, 2 out)
  24 S/PDIF (M-Audio Fast Track Ultra 8R), Windows DirectSound (0 in, 2 out)
  25 Line 3/4 (M-Audio Fast Track Ultra 8R), Windows DirectSound (0 in, 2 out)
  26 Lautsprecher (M-Audio Fast Track Ultra 8R), Windows DirectSound (0 in, 2 out)
  27 Line 5/6 (M-Audio Fast Track Ultra 8R), Windows DirectSound (0 in, 2 out)
  28 U28E570 (Intel(R) Display-Audio), Windows DirectSound (0 in, 2 out)
  29 Lautsprecher (Conexant SmartAudio HD), Windows DirectSound (0 in, 2 out)
  30 Fast Track Ultra 8R ASIO (x64), ASIO (8 in, 8 out)
  31 Line 1/2 (M-Audio Fast Track Ultra 8R), Windows WASAPI (0 in, 2 out)
  32 S/PDIF (M-Audio Fast Track Ultra 8R), Windows WASAPI (0 in, 2 out)
  33 Line 3/4 (M-Audio Fast Track Ultra 8R), Windows WASAPI (0 in, 2 out)
  34 Lautsprecher (M-Audio Fast Track Ultra 8R), Windows WASAPI (0 in, 2 out)
  35 Line 5/6 (M-Audio Fast Track Ultra 8R), Windows WASAPI (0 in, 2 out)
  36 U28E570 (Intel(R) Display-Audio), Windows WASAPI (0 in, 2 out)
  37 Lautsprecher (Conexant SmartAudio HD), Windows WASAPI (0 in, 2 out)
  38 S/PDIF (M-Audio Fast Track Ultra 8R), Windows WASAPI (2 in, 0 out)
  39 Line 3/4 (M-Audio Fast Track Ultra 8R), Windows WASAPI (2 in, 0 out)
  40 External Microphone (Conexant SmartAudio HD), Windows WASAPI (2 in, 0 out)
  41 Line 1/2 (M-Audio Fast Track Ultra 8R), Windows WASAPI (2 in, 0 out)
  42 Multichannel (M-Audio Fast Track Ultra 8R), Windows WASAPI (2 in, 0 out)
  43 Line 5/6 (M-Audio Fast Track Ultra 8R), Windows WASAPI (2 in, 0 out)
  44 Multichannel (Fast Track Ultra 8R Multi In), Windows WDM-KS (8 in, 0 out)
  45 Multichannel (Fast Track Ultra 8R Multi Out), Windows WDM-KS (0 in, 8 out)
  46 Line 1/2 (Fast Track Ultra 8R In 1-2), Windows WDM-KS (2 in, 0 out)
  47 Line 3/4 (Fast Track Ultra 8R In 3-4), Windows WDM-KS (2 in, 0 out)
  48 Line 5/6 (Fast Track Ultra 8R In 5-6), Windows WDM-KS (2 in, 0 out)
  49 S/PDIF (Fast Track Ultra 8R In 7-8), Windows WDM-KS (2 in, 0 out)
  50 Line 1/2 (Fast Track Ultra 8R Out 1-2), Windows WDM-KS (0 in, 2 out)
  51 Line 3/4 (Fast Track Ultra 8R Out 3-4), Windows WDM-KS (0 in, 2 out)
  52 Line 5/6 (Fast Track Ultra 8R Out 5-6), Windows WDM-KS (0 in, 2 out)
  53 S/PDIF 1 (Fast Track Ultra 8R Out 7-8), Windows WDM-KS (0 in, 2 out)
  54 S/PDIF 2 (Fast Track Ultra 8R Out 7-8), Windows WDM-KS (0 in, 2 out)
  55 Mikrofonarray (Conexant HD Audio capture), Windows WDM-KS (2 in, 0 out)
  56 Speakers (Conexant HD Audio output), Windows WDM-KS (0 in, 2 out)
  57 Output (Intel(R) Display-Audio - Ausgang 1.1), Windows WDM-KS (0 in, 2 out)
```

In [18]:
```
#set default fs and number of channels
sd.default.samplerate = fs
num_channels = 2

# select sound-device by choosing ID of above shown list.
input_device = 0
output_device = 1

fs_target = 8000

sd.default.device = [input_device,output_device]
```

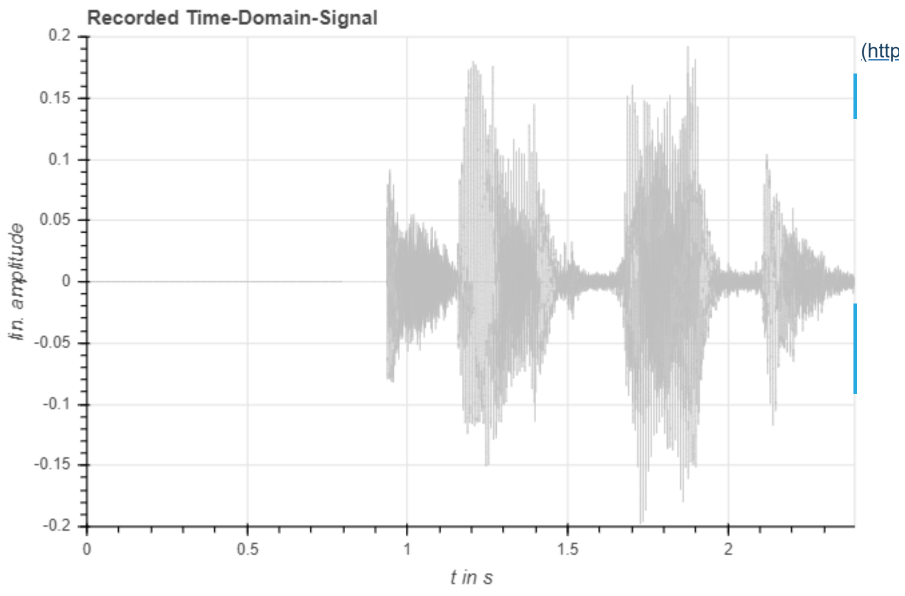
In [21]:
```
def on_click_Rec(button):
#callback function for Rec-Toggle button. Determines what happens if Rec-Button is switched On/Off
    value = button.new
    if (button.new == True):
        button.owner.button_style='danger'
        with out:
            print('Recording started!')
        sd.rec(out=indata,samplerate=fs)

    else:
        button.owner.button_style=''
        sd.stop()
        with out:
            print('Recording stopped!')

def deleteNan(indata):
    # helper function to delete Nans in record-array (indata). Trims data-vector to all values which are N
OT Nan.
    indata_no_Nan = np.zeros([np.sum(~np.isnan(indata[:,0])),indata.shape[1]])
    indata_no_Nan[:,0]=indata[~np.isnan(indata[:,0]),0]
    indata_no_Nan[:,1]=indata[~np.isnan(indata[:,0]),1]
    return indata_no_Nan

def on_click_Play(button):
    #callback function for Play-Toggle button. Determines what happens if Play-Button is switched On/Off
    if (button.new == True):
        button.owner.button_style='Success'
        with out:
            print('Playback started!')
        indata_no_Nan = deleteNan(indata)
        sd.play(indata_no_Nan)
        button.owner.description = 'Stop'
        button.owner.icon = 'stop-circle'
    if (button.new == False):
        sd.stop()
        with out:
            print('Playback stoped!')
        button.owner.button_style = ''
        button.owner.description = 'Play'
        button.owner.icon = 'play-circle'

def on_click_Clear(button):
    indata[:] = np.NaN
    with out:
        ipd.clear_output()
#         print('Audio-Array has been cleared!')
    return indata




#maximum-duration for recorded samples ==> recorded sample shouldn't be so long (nothing is recorded after
max_duration)
max_duration = 30 # seconds
indata = np.empty([max_duration*fs,num_channels])
indata[:] = np.NaN


toggleRec =widgets.ToggleButton(
    value=False,
    description='Recording',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Record_Button',
    icon='circle' # (FontAwesome names without the `fa-` prefix)
)

togglePlay = widgets.ToggleButton(
    value=False,
    description='Play',
    disabled=False,
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Play_Button',
    icon='play-circle' # (FontAwesome names without the `fa-` prefix)
)

clearButton = widgets.Button(
    description='Clear Audio-Array',
    button_style='', # 'success', 'info', 'warning', 'danger' or ''
    tooltip='Clear_Button',
    icon='trash' # (FontAwesome names without the `fa-` prefix)
)

toggleRec.observe(on_click_Rec, 'value')

togglePlay.observe(on_click_Play, 'value')

clearButton.on_click(on_click_Clear)

out = widgets.Output()
#display(out)

# object_methods = [method_name for method_name in dir(clearButton)
#                   if callable(getattr(clearButton, method_name))]
# print(object_methods)


box_layout = widgets.Layout(display='flex',
                flex_flow='column',
                align_items='center',
                width='100%')

widgets.HBox([toggleRec,togglePlay,clearButton,out],layout=box_layout)
```

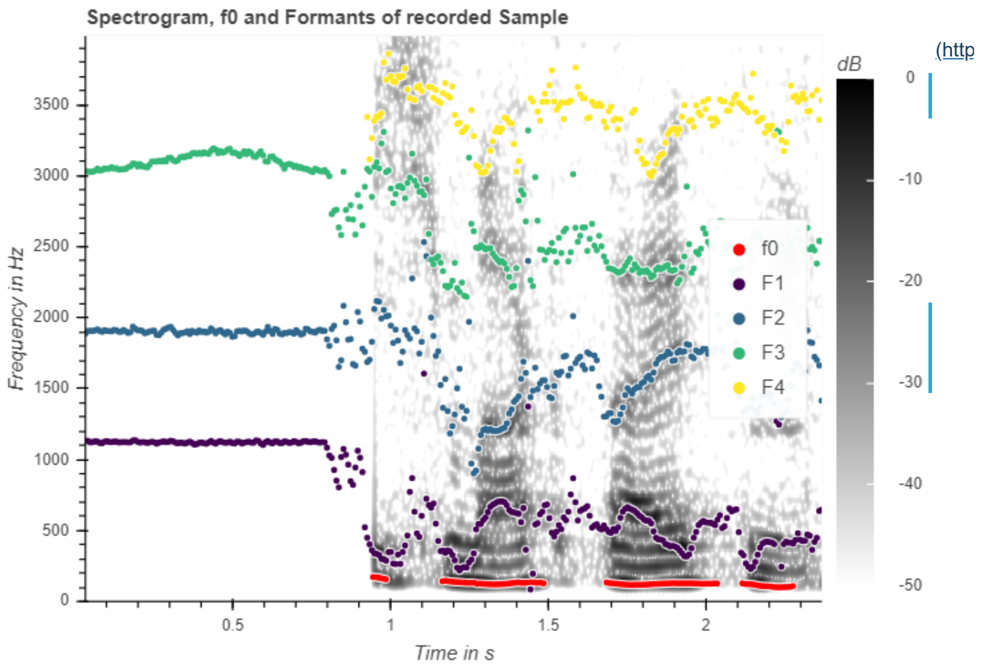
1. Plot: Recording of Standard Sentence

-time-domain signal plot

-Spectrogram/Formant/f0 plot

In [22]:
```
#prepare recordings for plots
indata_no_Nan = deleteNan(indata)
#convert to mono
recData = (indata_no_Nan[:,0]+indata_no_Nan[:,1])/2
rec_time = np.linspace(0,recData.shape[0]/fs,recData.shape[0])

#plot time-domain signal
SCplot.get_plot_time_domain_sig(recData,rec_time,'Recorded Time-Domain-Signal',showPlot=True)

# Analyze Formants and f0 with parselmouth:
pitchLo = 75 #Hz
pitchHi = 400 #Hz
pitchTimeStep = 30 #ms

snd = parselmouth.Sound(recData)

PM_pitch = snd.to_pitch(pitch_floor = pitchLo, pitch_ceiling=pitchHi)

pitch_tVec = PM_pitch.ts()
pitchValues = np.zeros_like(pitch_tVec)

for timeIdx, time in enumerate(pitch_tVec):
    pitchValues[timeIdx] = PM_pitch.get_value_at_time(time=time)

speech_spectro, speech_spectro_t, speech_spectr_f = PM_get_spectrogram(snd, 30, maximumFrequency=4000)

windowLength = 30 # ms
maxNumberFormants = 4
maxFormantFreq = 4000

PM_formants = snd.to_formant_burg(maximum_formant=maxFormantFreq, window_length=windowLength/1000,
                                  max_number_of_formants=maxNumberFormants)

formant_tVec = PM_formants.ts()
formantValues = np.zeros((maxNumberFormants,formant_tVec.size))

for timeIdx, time in enumerate(formant_tVec):
    for formantIdx in range(maxNumberFormants):
        formantValues[formantIdx,timeIdx] = PM_formants.get_value_at_time(formant_number=formantIdx+1, tim
e=time)

plottitle = 'Spectrogram, f0 and Formants of recorded Sample'
SCplot.plot_spectrogram_with_formants(speech_spectro, speech_spectro_t, speech_spectr_f, formantValues, fo
rmant_tVec,plottitle,
                                      pitchValues=pitchValues, pitch_tVec=pitch_tVec)
```



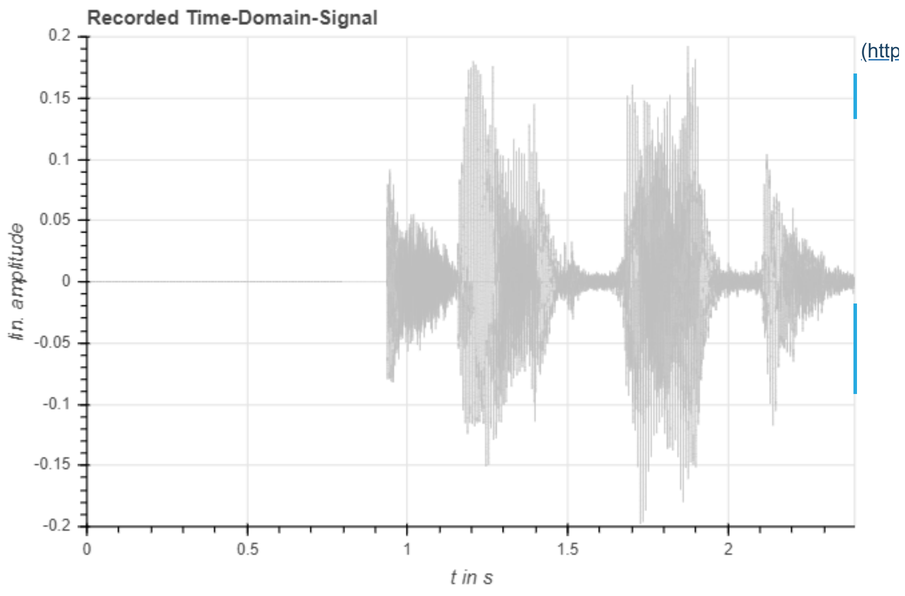


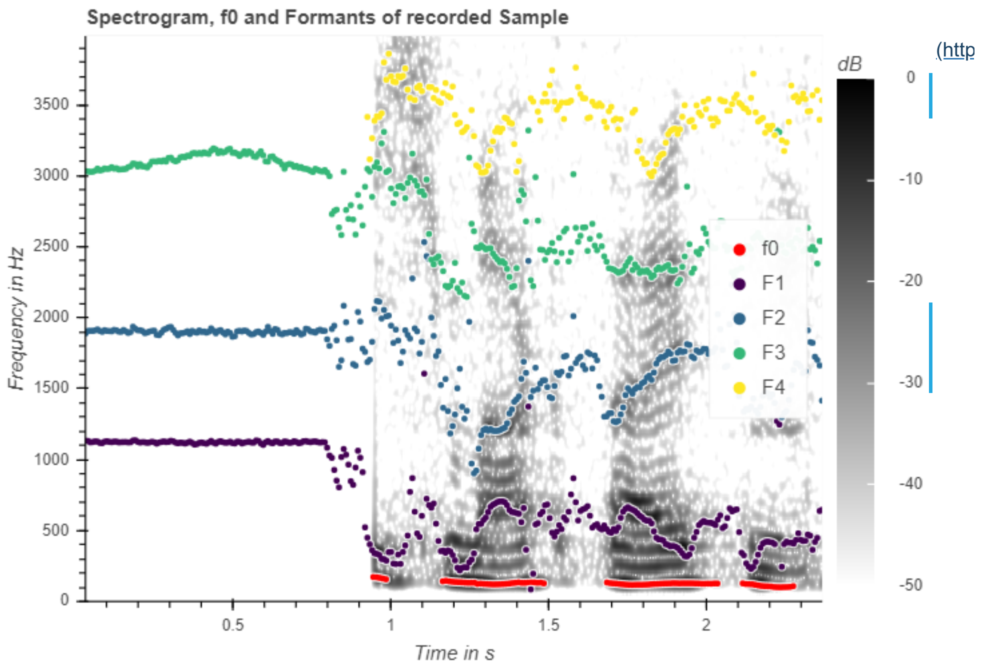
2. Plot: Recording of Dada-Sentence

Now, go back to the "record" button and record the sentence with the same voice melody, but just say "DaDa".

> -time-domain signal plot
>
> -Spectrogram/Formant/f0 plot

In [23]:
```
#prepare recordings for plots
indata_no_Nan = deleteNan(indata)
#convert to mono
recData = (indata_no_Nan[:,0]+indata_no_Nan[:,1])/2

rec_time = np.linspace(0,recData.shape[0]/fs,recData.shape[0])



#plot time-domain signal

SCplot.get_plot_time_domain_sig(recData,rec_time,'Recorded Time-Domain-Signal',showPlot=True)


# Analyze Formants and f0 with parselmouth:
pitchLo = 75 #Hz
pitchHi = 400 #Hz
pitchTimeStep = 30 #ms

snd = parselmouth.Sound(recData)

PM_pitch = snd.to_pitch(pitch_floor = pitchLo, pitch_ceiling=pitchHi)

pitch_tVec = PM_pitch.ts()
pitchValues = np.zeros_like(pitch_tVec)


for timeIdx, time in enumerate(pitch_tVec):
    pitchValues[timeIdx] = PM_pitch.get_value_at_time(time=time)

speech_spectro, speech_spectro_t, speech_spectr_f = PM_get_spectrogram(snd, 30, maximumFrequency=4000)


windowLength = 30 # ms
maxNumberFormants = 4
maxFormantFreq = 4000

PM_formants = snd.to_formant_burg(maximum_formant=maxFormantFreq, window_length=windowLength/1000,
                                  max_number_of_formants=maxNumberFormants)

formant_tVec = PM_formants.ts()
formantValues = np.zeros((maxNumberFormants,formant_tVec.size))


for timeIdx, time in enumerate(formant_tVec):
    for formantIdx in range(maxNumberFormants):
        formantValues[formantIdx,timeIdx] = PM_formants.get_value_at_time(formant_number=formantIdx+1, tim
e=time)

plottitle = 'Spectrogram, f0 and Formants of recorded Sample'
SCplot.plot_spectrogram_with_formants(speech_spectro, speech_spectro_t, speech_spectr_f, formantValues, fo
rmant_tVec,plottitle,
                                      pitchValues=pitchValues, pitch_tVec=pitch_tVec)
```





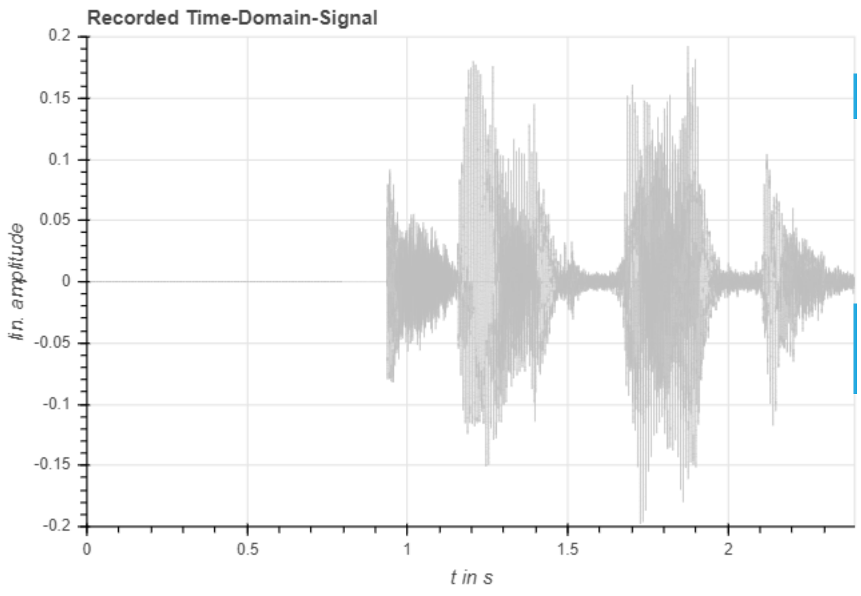
3. Plot: Recording of whispered Sentence

Now, go back to the "record" button and record the sentence with the same voice melody, but whisper only.

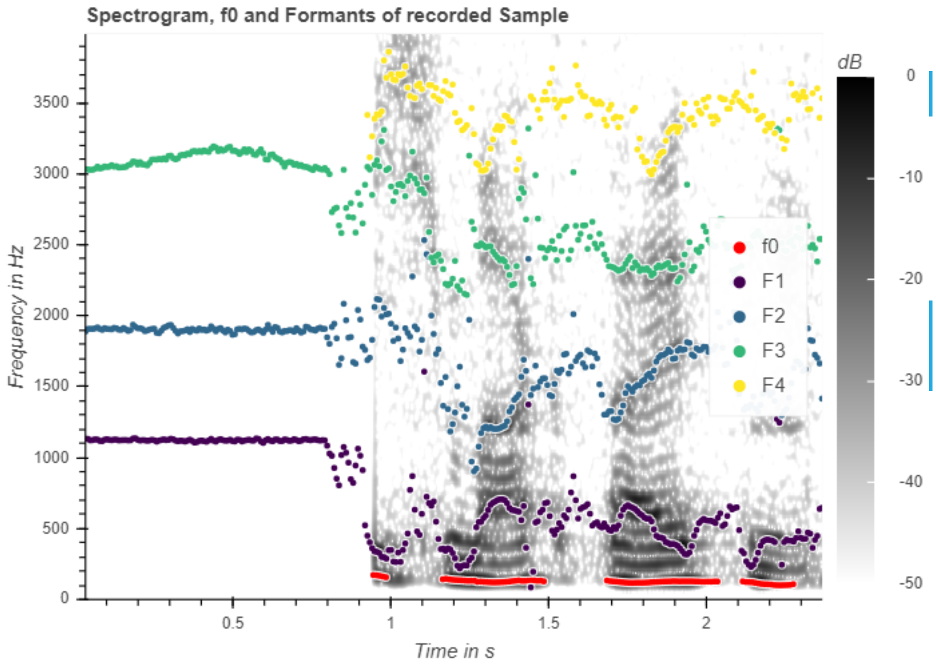
-time-domain signal plot

-Spectrogram/Formant/f0 plot

In [24]:
```
#prepare recordings for plots
indata_no_Nan = deleteNan(indata)
#convert to mono
recData = (indata_no_Nan[:,0]+indata_no_Nan[:,1])/2

rec_time = np.linspace(0,recData.shape[0]/fs,recData.shape[0])

#plot time-domain signal
SCplot.get_plot_time_domain_sig(recData,rec_time,'Recorded Time-Domain-Signal',showPlot=True)

# Analyze Formants and f0 with parselmouth:
pitchLo = 75 #Hz
pitchHi = 400 #Hz
pitchTimeStep = 30 #ms

snd = parselmouth.Sound(recData)

PM_pitch = snd.to_pitch(pitch_floor = pitchLo, pitch_ceiling=pitchHi)

pitch_tVec = PM_pitch.ts()
pitchValues = np.zeros_like(pitch_tVec)


for timeIdx, time in enumerate(pitch_tVec):
    pitchValues[timeIdx] = PM_pitch.get_value_at_time(time=time)

speech_spectro, speech_spectro_t, speech_spectr_f = PM_get_spectrogram(snd, 30, maximumFrequency=4000)


windowLength = 30 # ms
maxNumberFormants = 4
maxFormantFreq = 4000

PM_formants = snd.to_formant_burg(maximum_formant=maxFormantFreq, window_length=windowLength/1000,
                                  max_number_of_formants=maxNumberFormants)

formant_tVec = PM_formants.ts()
formantValues = np.zeros((maxNumberFormants,formant_tVec.size))

for timeIdx, time in enumerate(formant_tVec):
    for formantIdx in range(maxNumberFormants):
        formantValues[formantIdx,timeIdx] = PM_formants.get_value_at_time(formant_number=formantIdx+1, tim
e=time)

plottitle = 'Spectrogram, f0 and Formants of recorded Sample'
SCplot.plot_spectrogram_with_formants(speech_spectro, speech_spectro_t, speech_spectr_f, formantValues, fo
rmant_tVec,plottitle,
                                      pitchValues=pitchValues, pitch_tVec=pitch_tVec)
```


Recorded Time-Domain-Signal
0.2
0.15
0.1
0.05
0
-0.05
-0.1
-0.15
-0.2
lin. amplitude
0
0.5
1
1.5
2
t in s

(http


Spectrogram, f0 and Formants of recorded Sample
3500
3000
2500
2000
1500
1000
500
0
Frequency in Hz
0.5
1
1.5
2
Time in s
f0
F1
F2
F3
F4
dB
0
-10
-20
-30
-40
-50

(http