# Speech Communication Lab - Speech Analysis

May 27, 2020

# 1 Introduction: Load Modules and Import Audio

### 1.0.1 Load Modules

[1]:
```
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

from pathlib import Path
from ipywidgets import interact, interact_manual, Layout
from bokeh.plotting import ColumnDataSource, figure, output_file, show
from bokeh.io import push_notebook, output_notebook
from bokeh.layouts import gridplot, column, row
from bokeh.models import ColorBar, LogColorMapper, LogTicker, CustomJS, Slider,␣
 ↪LinearAxis, Range1d
from scipy import signal
from bokeh.palettes import Greys256

Greys256.reverse() # reverse the grey color palette such that black is the␣
 ↪maximum and white is the minimum

output_notebook(hide_banner=True) # suppress Bokeh banner when loading plots
```

### 1.0.2 Load and Playback Audio File

[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
```

```
f216.wav loaded with f_s =16000
```

[2]:
```
<IPython.lib.display.Audio object>
```

# 2 Experiment 1: Time-Domain Analysis

## 2.1 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.

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

def SC_intensity(sound, minimumPitch, fs):
    winLen = np.round(3.2/minimumPitch * fs)  # Window length in samples; from␣
↪Praat documentation
    alpha = 2.5  # width factor alpha >= 0
    std = (winLen-1)/(2*alpha)  # Matlab documentation

    gaussWin = signal.gaussian(winLen, std)
    sound = np.square(sound-np.mean(sound)) # remove mean before convolution
    intensity = np.convolve(sound, gaussWin, mode='valid')

    intensity = 10*np.log10(intensity) # conversion to dB
    print("SC_intensity: Intensity has {} samples".format(intensity.size))

    return intensity, gaussWin

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


# Plot function for window function
def get_plot_window(window, dt_win, plottitle, showPlot=False):
```

```
    p = figure(title=plottitle, plot_width=600, plot_height=200)
    p.line(dt_win, window, line_width = 2, color = 'blue')
    p.xaxis.axis_label = 't in s'
    p.yaxis.axis_label = 'lin. amplitude'
    if showPlot:
        show(p, notebook_handle=False)
        pass
    else:
        return p

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


# Plot function for intensity curve
def get_plot_intensity(snd, dt_snd, intensity, dt_intensity, plottitle,␣
 ↪showPlot=False):
    p = figure(title=plottitle,plot_width=600, plot_height=400,␣
 ↪x_range=(dt_snd[0], dt_snd[-1]), y_range=(np.floor(np.min(snd)*10)/10, np.
 ↪ceil(np.max(snd)*10)/10))
    p.line(dt_snd, snd, line_width = 0.5, color = '#BEBEBE')
    p.xaxis.axis_label = 't in s'
    p.yaxis.axis_label = 'lin. amplitude'

    p.extra_y_ranges = {"intensity": Range1d(start=np.floor(np.min(intensity)/
 ↪10)*10, end=np.ceil(np.max(intensity)/10)*10)}
    p.line(dt_intensity, intensity, line_width = 2, color = 'red',␣
 ↪y_range_name="intensity")
    p.add_layout(LinearAxis(y_range_name="intensity", axis_label='Intensity in␣
 ↪dB'), 'right')
    if showPlot:
        show(p, notebook_handle=False)
        pass
    else:
        return p


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

# Show 2 subplots
```

```
def plot_in_subplots(p1, p2):
    show(column(p1, p2), notebook_handle=False)


plot_in_subplots(p_window, p_intensity)
```

```
SC_intensity: Intensity has 54596 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.

[4]:
```
PM_intensity = snd.to_intensity(minimum_pitch=minimumPitch,␣
 ↪subtract_mean=False)  # intensity 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
get_plot_intensity(np.squeeze(snd.values), dt_snd, PM_intensity_val,␣
 ↪dt_PM_intensity, plottitle, showPlot=True)
```

```
PM_intensity: Intensity has 210 samples
```

[5]:
```
# plot the 2 intensity curves in one plot
p = figure(title="Comparison of both intensity curves - File: " + fileName,␣
 ↪plot_width=600, plot_height=400, y_range=(np.floor(np.
 ↪min(PM_intensity_val)*10)/10, np.ceil(np.max(PM_intensity_val)*10)/10))
p.line(dt_PM_intensity, PM_intensity_val, line_width = 2, color = 'blue',␣
 ↪legend_label="parselmouth intensity curve")
p.xaxis.axis_label = 't in s'
p.yaxis.axis_label = 'Intensity in dB (parselmouth)'
p.yaxis.axis_line_color = 'blue'
p.yaxis.major_tick_line_color= 'blue'

p.extra_y_ranges = {"intensity-custom": Range1d(start=np.floor(np.
 ↪min(SC_intensity)/10)*10, end=np.ceil(np.max(SC_intensity)/10)*10)}
p.line(dt_SC_intensity, SC_intensity, line_width = 2, color = 'red',␣
 ↪y_range_name="intensity-custom", legend_label="custom intensity curve")
p.add_layout(LinearAxis(y_range_name="intensity-custom", axis_label='Intensity␣
 ↪in dB (custom function)', axis_line_color = 'red', major_tick_line_color=␣
 ↪'red'), 'right')
```

```
p.legend.location = "bottom_center"
show(p, notebook_handle=False)
```

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

### 2.1.1 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

## 3 Experiment 2: Frequency-Domain Analysis

### 3.0.1 Load and Playback Audio File

[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
```

[6]:
```
<IPython.lib.display.Audio object>
```

### 3.0.2 Wide- and narrow-band spectrograms

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

[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, nperseg=windowlength, return_onesided=True,␣
↪scaling='spectrum', mode='magnitude')

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

def plot_interactive_spectrogram(spectroData, tVec, fVec, plottitle,␣
↪dynamicRange=50):
    layout=Layout(width='650px')
    timeWidget = widgets.FloatSlider(min=tVec[0], max=tVec[-1],␣
↪step=tVec[1]-tVec[0], value=tVec[0],
                                     description="Time in s")
    timeWidget.layout = layout
    timeInStft = timeWidget.value
    stftFrame = (np.abs(tVec - timeInStft)).argmin()

    TOOLS="hover,crosshair,pan,wheel_zoom,box_zoom,save,reset"
    TOOLTIPS = [
        ("index", "$index"),
        ("(x,y)", "($x, $y)"),
    ]
    p1 = figure(title=plottitle,plot_width=650, plot_height=450,␣
↪x_range=(tVec[0],tVec[-1]),
                y_range=(fVec[0],fVec[-1]), tools=TOOLS, tooltips=TOOLTIPS)
    p1.xaxis.axis_label = 'Time in s'
    p1.yaxis.axis_label = 'Frequency in Hz'

    color_mapper = bokeh.models.LinearColorMapper(palette=Greys256,␣
↪low=spectroData.max()-dynamicRange, high=spectroData.max())
    color_bar = ColorBar(color_mapper=color_mapper, title='dB',␣
↪title_text_align='left',
                         label_standoff=12, border_line_color=None, location=(0,0))

    p1.add_layout(color_bar, 'right')

    p1.image(image=[spectroData], x=tVec[0], y=fVec[0], dw=tVec[-1],␣
↪dh=fVec[-1], color_mapper=color_mapper)
    p1.grid.visible=False
    spectrumLine = p1.line([timeInStft, timeInStft], [fVec[0], fVec[-1]],␣
↪line_width = 2, color = 'red')

    p2 = figure(title="Spectrum of Selected STFT Frame",plot_width=650,␣
↪plot_height=300, x_range=(fVec[0],fVec[-1]),
                y_range=(np.min(spectroData[:,stftFrame]),0), tools=TOOLS,␣
↪tooltips=TOOLTIPS)
    p2.xaxis.axis_label = 'Frequency in Hz'
```

```
    p2.yaxis.axis_label = 'relative Magnitude in dB'
    spectrumPlot = p2.line(fVec, spectroData[:,stftFrame], line_width = 2,␣
 ↪color = 'red', legend_label='slice of spectrogram')
    p2.line(fVec, spectroData.max()-dynamicRange, line_width=1, color='grey',␣
 ↪legend_label='dynamic range')
    p2.legend.location = "bottom_center"
    p2.legend.orientation = 'horizontal'

    pAll = gridplot([[p2], [p1]])
    show(pAll,notebook_handle=True)

    def update_plot(timeInStft, stftFrame):
        spectrumLine.data_source.data['x'] = [timeInStft, timeInStft]
        spectrumPlot.data_source.data['y'] = spectroData[:,stftFrame]
        push_notebook()

    def on_value_change(change):
        timeInStft = timeWidget.value
        stftFrame = (np.abs(tVec - timeInStft)).argmin()
        update_plot(timeInStft, stftFrame)

    timeWidget.observe(on_value_change, names='value')
    return timeWidget

plottitle = "Custom Spectrogram of Sound Sample - File: " + fileName
SC_timeWidget = plot_interactive_spectrogram(SC_spectroDataDB, SC_tVec,␣
 ↪SC_fVec, plottitle)
widgets.HBox([SC_timeWidget])
```

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

```
HBox(children=(FloatSlider(value=0.015, description='Time in s', layout=Layout(width='650px'),
```

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

[8]:
```
windowlengthSec = 5 #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 = plot_interactive_spectrogram(PM_spectroDataDB, PM_tVec,␣
 ↪PM_fVec, plottitle)
widgets.HBox([PM_timeWidget])
```

```
HBox(children=(FloatSlider(value=0.005500000000000133, description='Time in s', layout=Layout(
```

### 3.0.3 f0 and Formants

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

Select sound file to analyze:

[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
```

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

[9]:
```
<IPython.lib.display.Audio object>
```

Firstly, we analyse the formants F1 … F4 only.

[11]:
```
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, time=time)

def plot_spectrogram_with_f0_and_formants(spectroData, tVec, fVec,␣
↪formantValues, formant_tVec, plottitle,
                                          pitchValues=np.array([np.nan, np.
↪nan]), pitch_tVec=np.array([np.nan, np.nan]),
                                          dynamicRange=50):
    TOOLS="hover,crosshair,pan,wheel_zoom,box_zoom,save,reset"
    TOOLTIPS = [
        ("index", "$index"),
        ("(x,y)", "($x, $y)"),
    ]

    p1 = figure(title=plottitle,plot_width=650, plot_height=450,␣
↪x_range=(tVec[0],tVec[-1]),
                y_range=(fVec[0],fVec[-1]), tools=TOOLS, tooltips=TOOLTIPS)
    p1.xaxis.axis_label = 'Time in s'
    p1.yaxis.axis_label = 'Frequency in Hz'


    color_mapper = bokeh.models.LinearColorMapper(palette=Greys256,␣
↪low=spectroData.max()-dynamicRange, high=spectroData.max())
    color_bar = ColorBar(color_mapper=color_mapper, title='dB',␣
↪title_text_align='left',
                     label_standoff=12, border_line_color=None, location=(0,0))

    p1.add_layout(color_bar, 'right')

    p1.image(image=[spectroData], x=tVec[0], y=fVec[0], dw=tVec[-1],␣
↪dh=fVec[-1], color_mapper=color_mapper)
    p1.grid.visible=False

    if ~np.isnan(pitchValues.all()):
        p1.scatter(pitch_tVec, pitchValues, size=6, line_color=None,␣
↪fill_alpha=0.8,
                       fill_color='white')
        p1.scatter(pitch_tVec, pitchValues, size=4, line_color=None,␣
↪fill_alpha=1,
                       fill_color='red', legend_label='f0')

    for formantIdx in range(maxNumberFormants):
        p1.scatter(formant_tVec, formantValues[formantIdx,:], size=6,␣
↪line_color=None, fill_alpha=0.8,
                   fill_color='white')
```

```
        p1.scatter(formant_tVec, formantValues[formantIdx,:], size=4,␣
↪line_color=None, fill_alpha=1,
                    fill_color=bokeh.palettes.
↪viridis(maxNumberFormants)[formantIdx], legend_label='F{}'.
↪format(formantIdx+1))

    p1.legend.location = "center_right"
    p1.legend.orientation = 'vertical'
    show(p1,notebook_handle=True)
    pass


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

plottitle = "Spectrogram and Formants of Sound Sample - File: " + fileName
plot_spectrogram_with_f0_and_formants(spectroDataDB, spec_tVec, spec_fVec,␣
↪formantValues, formant_tVec, plottitle)
```

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

Select sound file to analyze:

[12]:
```
# 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
```

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

[12]:
```
<IPython.lib.display.Audio object>
```

[13]:
```
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)
```

```
def plot_spectrogram_with_f0_and_formants(spectroData, tVec, fVec,␣
↪formantValues, formant_tVec, plottitle,
                                          pitchValues=np.array([np.nan, np.
↪nan]), pitch_tVec=np.array([np.nan, np.nan]), dynamicRange=50):
    TOOLS="hover,crosshair,pan,wheel_zoom,box_zoom,save,reset"
    TOOLTIPS = [
        ("index", "$index"),
        ("(x,y)", "($x, $y)"),
    ]

    p1 = figure(title=plottitle,plot_width=650, plot_height=450,␣
↪x_range=(tVec[0],tVec[-1]),
                y_range=(fVec[0],fVec[-1]), tools=TOOLS, tooltips=TOOLTIPS)
    p1.xaxis.axis_label = 'Time in s'
    p1.yaxis.axis_label = 'Frequency in Hz'


    color_mapper = bokeh.models.LinearColorMapper(palette=Greys256,␣
↪low=spectroData.max()-dynamicRange, high=spectroData.max())
    color_bar = ColorBar(color_mapper=color_mapper, title='dB',␣
↪title_text_align='left',
                     label_standoff=12, border_line_color=None, location=(0,0))

    p1.add_layout(color_bar, 'right')

    p1.image(image=[spectroData], x=tVec[0], y=fVec[0], dw=tVec[-1],␣
↪dh=fVec[-1], color_mapper=color_mapper)
    p1.grid.visible=False

    if ~np.isnan(pitchValues.all()):
        p1.scatter(pitch_tVec, pitchValues, size=6, line_color=None,␣
↪fill_alpha=0.8,
                       fill_color='white')
        p1.scatter(pitch_tVec, pitchValues, size=4, line_color=None,␣
↪fill_alpha=1,
                       fill_color='red', legend_label='f0')

    for formantIdx in range(maxNumberFormants):
        p1.scatter(formant_tVec, formantValues[formantIdx,:], size=6,␣
↪line_color=None, fill_alpha=0.8,
                   fill_color='white')
        p1.scatter(formant_tVec, formantValues[formantIdx,:], size=4,␣
↪line_color=None, fill_alpha=1,
                   fill_color=bokeh.palettes.
↪viridis(maxNumberFormants)[formantIdx], legend_label='F{}'.
↪format(formantIdx+1))
```

```
    p1.legend.location = "center_right"
    p1.legend.orientation = 'vertical'
    show(p1,notebook_handle=True)
    pass



plottitle = "Spectrogram, f0 and Formants of Sound Sample - File: " + fileName
plot_spectrogram_with_f0_and_formants(spectroDataDB, spec_tVec, spec_fVec,␣
 ↪formantValues, formant_tVec,plottitle,
                                       pitchValues=pitchValues,␣
 ↪pitch_tVec=pitch_tVec)
```

## 4 Experiment 3: Estimation of Vocal Tract using Cepstrum and LPC

[ ]:

```

```

## 5 Experiment 4: Formant Analysis

[ ]:

```

```