SIGNAL AND SYSTEMS TERM PROJECT

Berru Karakaş

150190733

Parta.pdf – Includes theoretical calculations to find pseudo code for the given band pass filter.

Explanations are given prior to code snippets.

Filter.py LIBRARY

By using the pseudo code coefficients calculated in a, python code is written.

Variables:

K : calculated for the filter

T: 1/samplerate given as 44100 Hz

J: array for intermediate values in direct programming J1,J2….

Intermediate coefficients for simplification:

```
A = np.pi*T*fcL
    B = np.pi*T*fcH
C = 8*K*(A**3)*(B**3)/(T**3)
    D = A*B + A + B + 1
    E = 2*A*B -2
    F = A*B - A - B + 1
```

b0,b1,b2… coefficients for direct programming

```
b = [C, 0, -3*C, 0, 3*C, 0, -C]
```

a0,a1,a2… coefficients for direct programming

```
a = [D**3,       3*E*(D**2),
        3*(D**2)*F + 3*D*(E**2),
        6*D*E*F + (E**3),
        3*(E**2)*F + 3*D*(F**2),
        3*E*(F**2),        F**3]
```

Equation for J1 and output

```
#Transfer Function
    J[1] = X/a[0] - J[2]*(a[1]/a[0]) - J[3]*(a[2]/a[0]) - J[4]*(a[3]/a[0]) -
    J[5]*(a[4]/a[0]) - J[6]*(a[5]/a[0]) - J[7]*(a[6]/a[0])

    Y = (b[0]/a[0])*( J[1] + -3*J[3] + 3*J[5] - J[7])
```

Assigning each value to the previous one to create unit delay

```
for i in range(7,1,-1): # 7 6 5 4 3 2
        J[i] = J[i-1]
```

Audio.py LIBRARY

Open_wav Function

Wave library is used to acquire the audio as frames, then unpack to a numpy array with struct library to work with the data

```
def open_wav(file_name):
    sound = wave.open(file_name, 'r')  # reads file
    length = sound.getnframes()   # get frames
    data = sound.readframes(length)
    data = struct.unpack('{n}h'.format(n=length), data) #unpack to array
    data = np.array(data) #transforms to np array
    sound.close()
    return data
```

Save_wav Function

Struct library is used to pack array into frames and write to file with wave.

```
def save_wav(file_name,audio):

    #Open wav file
    wav_file=wave.open(file_name,"w")

    #Set parameters
    wav_file.setparams((1, 2, sample_rate, len(audio), "NONE", "not compressed"))

    #Pack samples
    for sample in audio:
        wav_file.writeframes(struct.pack('{n}h'.format(n=1), int(sample)))

    #Close file
    wav_file.close()
```

BPF Function

As input, audio data array and frequency points are given. Initial values are assumed to be zero and initialized as such. In a for loop of length data, filter is applied to each value then stored in an array.

```
def BPF_helper(data,fcL,fcH):

    output = np.zeros(len(data)) #initialize with zeros
    J = np.zeros(8) #initialize with zeros

    #Pass through filter for each frame
    for i in range(len(data)):
        y, Jnew = BandPassFilter(data[i],J,fcL,fcH,sample_rate)
        J = Jnew
        output[i]= y

    return output
```

BSF Function

As input, audio data array and frequency points are given. Initial values are assumed to be zero and initialized as such. In a for loop of length data, filter is applied to each value and subtracted from the original, then stored in an array.

```
def BSF_helper(data,fcL,fcH):
    output = np.zeros(len(data))
    J = np.zeros(8)
    for i in range(len(data)):
        y, Jnew = BandPassFilter(data[i],J,fcL,fcH,sample_rate)
        J = Jnew
        output[i]= data[i] - y
    return output
```

GRAPHICAL USER INTERFACE

Code Structure:

Class MainWindow has

-variables shared among the methods
-methods linked with widgets

Shared variables:

```
#Class variables
freq = 0
bw = 0
mode = False
output = []
file_path = ""
fcL = 0
fcH = 0
```

Methods:

__init__ method

Create window for GUI

```
self.setMinimumSize(QSize(600, 220))
        self.setWindowTitle("BAND FILTER")
```

Get user input with QLineEdit

```
self.lineLabel = QLabel(self)
        self.lineLabel.setText('Frequency:')
        self.line = QLineEdit(self)
        self.line.move(80, 60)
        self.line.resize(200, 32)
        self.lineLabel.move(80, 20)
```

QComboBox for filter mode choice

```
combo = QComboBox(self)
        combo.addItem("BPF")
        combo.addItem("BSF")
        combo.move(90, 120)
        self.qlabel = QLabel(self)
        self.qlabel.setText('Filter Mode')
        self.qlabel.move(90,90)
        combo.activated[str].connect(self.onChanged)
```

QPushbuttons for buttons

```
pybutton = QPushButton('Send', self)
        pybutton.clicked.connect(self.clickMethod)
        pybutton.resize(100,32)
        pybutton.move(240, 120)
```

findFile method

Linked to “Browse” button. Filters only wav files and gets the path of the chosen file.

```
def findFile(self):
        filter = "wav(*.wav)" #filter for wav files

        #Search only wav files
        path = QtWidgets.QFileDialog.getOpenFileName(None, "Select Audio File", "
", "Wav files (*.wav)")

        #Get file path
        self.file_path = path[0]
```

playFile method

Linked to “Play” button. Data is read from file. With frequency inputs and appropriate filter mode, filter is applied. Messages in intermediate steps are shown with QMessageBox. The output acquired from the filter is played with sd.play

```
def playFile(self):

        #If path not chosen, raise an exception
        if self.file_path == "":
            print("Choose a file")

        data, samplerate = sf.read(self.file_path)  #read file

        #Band Stop Filter mode
        if self.mode:
            QMessageBox.about(self, "Filtering...","Band Stop Filter for fcL:{} f
cH:{} applying".format(self.fcL,self.fcH))
            self.output = BSF_helper(data, self.fcL, self.fcH)
            QMessageBox.about(self, "DONE!","Press Play to listen, Save to Save")

        #Band Pass Filter mode
        else:
            QMessageBox.about(self, "Filtering...","Band Pass Filter for fcL:{} f
cH:{} applying".format(self.fcL,self.fcH))
            self.output = BPF_helper(data, self.fcL, self.fcH)
            QMessageBox.about(self, "DONE!","Press Play to listen, Save to Save")

        #winsound.PlaySound(self.file_path, winsound.SND_FILENAME)
        sd.play(self.output, samplerate) #play audio
        #status = sd.wait()  # Wait until file is done playing
```

saveFile method

Linked to “Save” button. Split file_path with dot and adds B to end. Saves wav file with wave.

```
#Method for saving file
    def saveFile(self):
        #Create file name with B appended to end
        splitted = self.file_path.split('.') #split with dot
        path = splitted[0] + "B." + splitted[1] #add B
        save_wav(path , self.output) #save
```

clickMethod method

Linked to “Send” button. Checks the given inputs bounds and calculated fcH and fcL

```
 #Click for inputs
    def clickMethod(self):

        freq = float(self.line.text())
        bw = float(self.line_2.text())

        #Frequency and bandwidth bound check
        if (freq < 200 or freq > 7000) and (bw < 5 or bw > 50):
            QMessageBox.about(self, "Try Again!", "Frequency must be between 200
and 7000, Bandwidth must be between 5 and 50.")
        elif freq < 200 or freq > 7000:
            QMessageBox.about(self, "Try Again!", "Frequency must be between 200
and 7000")
        elif bw < 5 or bw > 50:
            QMessageBox.about(self, "Try Again!", "Bandwidth must be between 5 an
d 50")
        else:
            #High low frequency calculations
            self.fcH = freq + (freq*bw/100)
            self.fcL = freq - (freq*bw/100)
            print('Freq:{} {} '.format(self.fcH,self.fcL))
            print('Bandwidth:{} '.format(bw))
```

onChanged method

Linked to combo. self.mode is set according to filter mode. BPF for false and BSF for true.

```
def onChanged(self,text):

        #Choosing filter type
        if text == "BPF":
            self.mode = False
        else:
            self.mode = True

        self.qlabel.adjustSize()
```

USER MANUEL

Frequency box: for frequency input

Bandwidth % box: for bw percentage

Filter mode: choice for BPF and BSF

Save button: to save filtered output

Play button: to play filtered output

Browse: to find an input wav file

Send: to send inputs

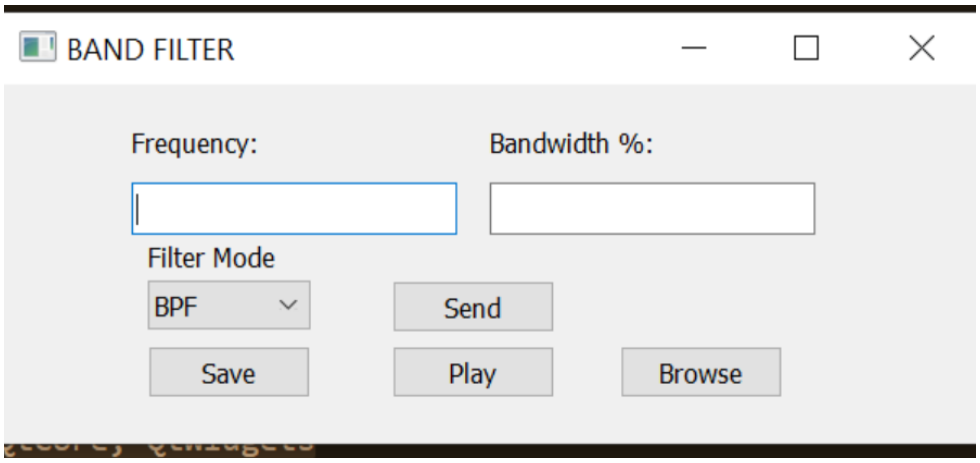
You may use it as such:

Write input

Press Send

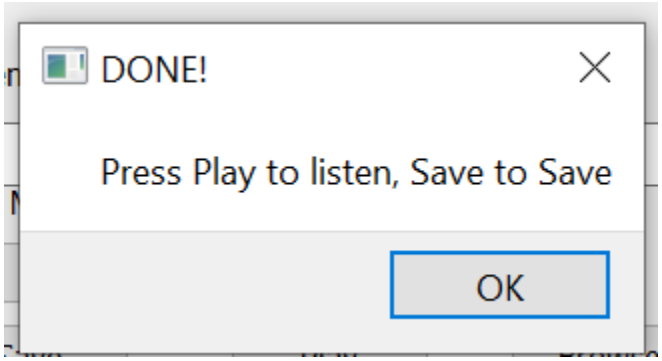
Browse Files

Press Play

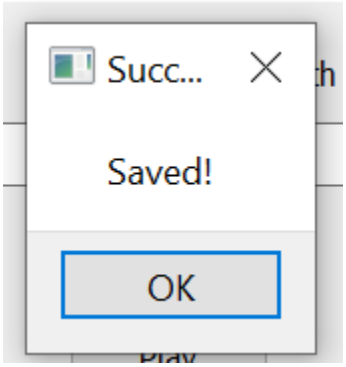


Pop ups:

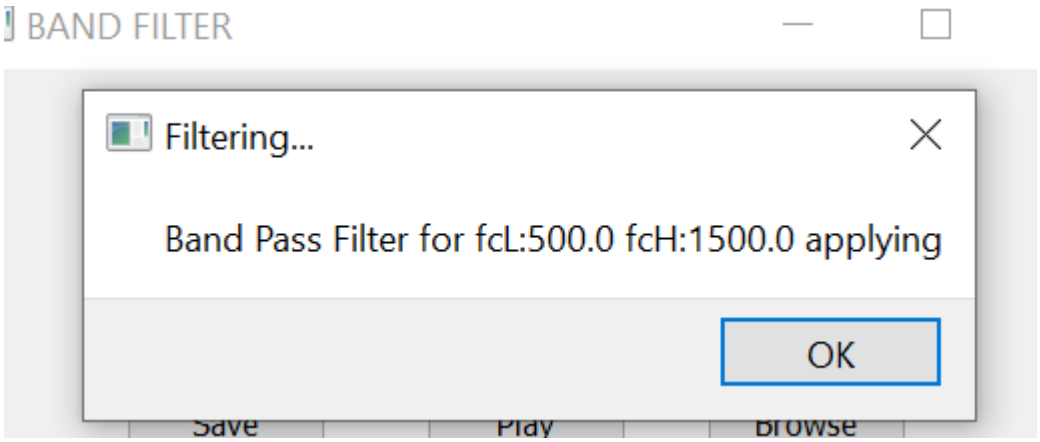
When filtering is done:



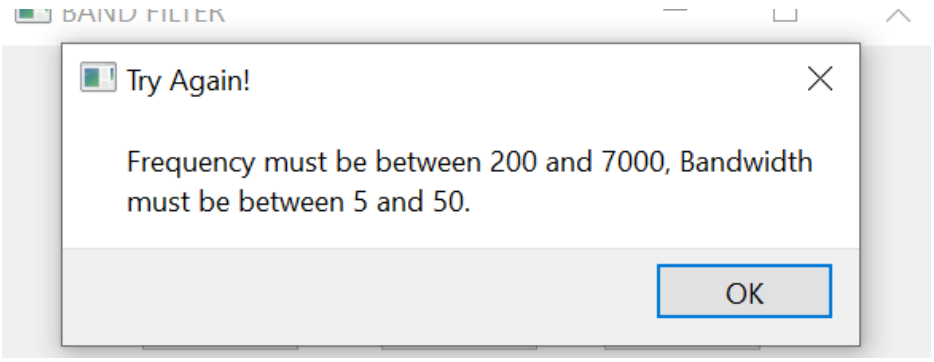
When saved:



When filtering begins:



Errors:



Requirements