大作业

一、项目创意

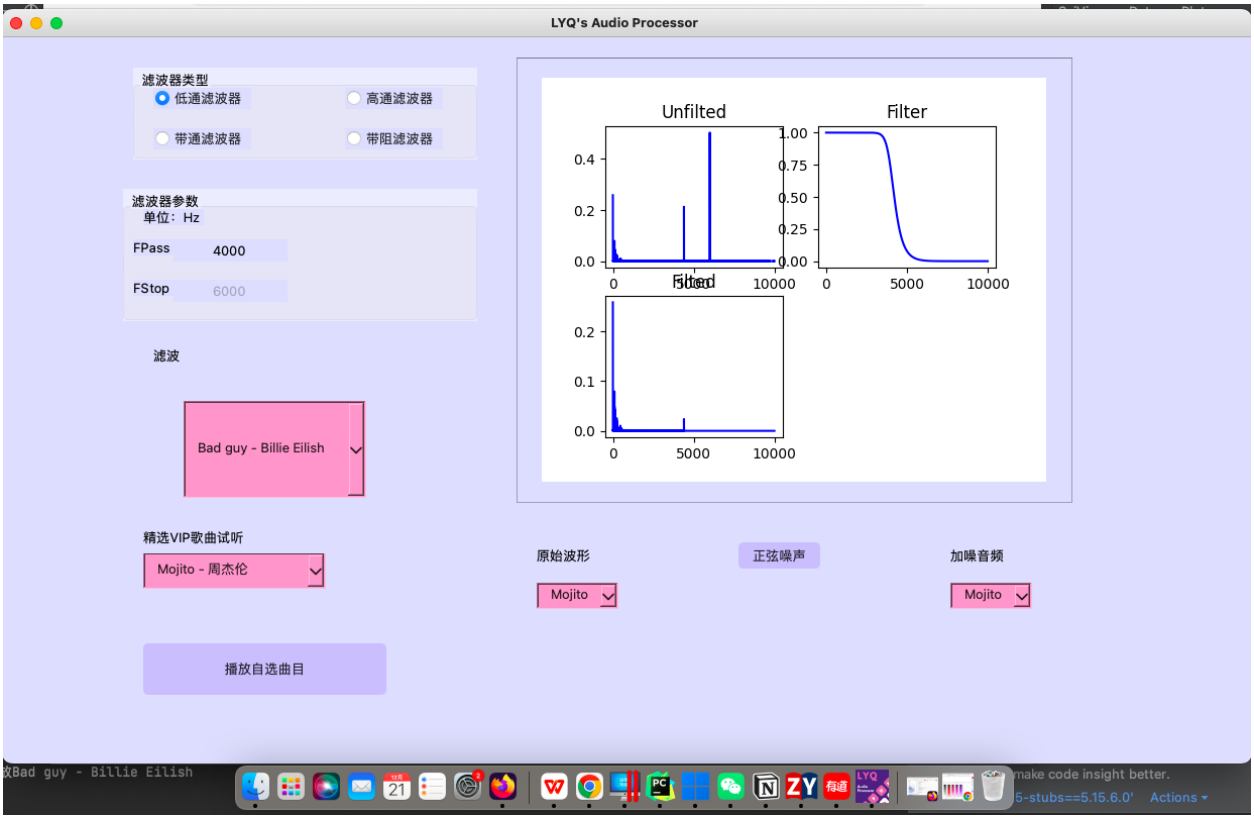
根据本学期的数字音频处理课程，设计了一个音频处理软件，实现了音频播放，音频片段的可视化呈现，音频加噪和滤波的丰富功能。

本项目最为创新的部分是将各个功能设计建立在用户个性化的选择上。也就是说，每个功能都为用户提供了多元化的、具有音乐性的音频选择，而不仅仅将功能局限于单个缺乏现实意义的事例音频文片段。这使本软件的音频处理功能拥有了更加现实化的意义。

同时本软件综合美学考虑，设计了同色系的界面，力求提高用户使用感受。

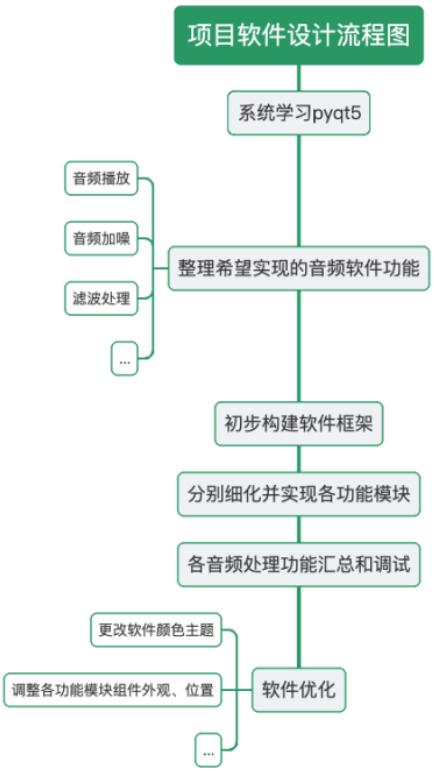
二、软件界面截图

1、软件界面



2、软件logo

三、项目软件设计流程图



四、项目主要模块的实现

1、播放歌曲

```
# 音乐播放
        # 播放歌单
        self.cb = QComboBox(self)
        self.cb.setGeometry(QtCore.QRect(140, 510, 181, 36))
        self.labelcb = QtWidgets.QLabel("精选VIP歌曲试听",self)
        self.labelcb.setGeometry(QtCore.QRect(140, 480, 241, 31))
        self.cb.addItem('Mojito - 周杰伦') # 添加下拉框内容
        self.cb.addItem('Bad guy - Billie Eilish')
        self.cb.addItem('Alright - CYN')
        self.cb.addItem('Jam Jam - IU')
```

```
        self.cb.currentIndexChanged[str].connect(self.print_value) # 打印所选歌曲名
        self.cb.currentIndexChanged[int].connect(self.play_select)  # 通过所选歌曲索引播放歌曲
        self.cb.setStyleSheet('QComboBox{background-color:#FF95CA;} ')

        # 播放本地文件
        self.pushButton_5 = QtWidgets.QPushButton("播放自选曲目", self)
        self.pushButton_5.setGeometry(QtCore.QRect(140, 600, 241, 51))
        self.pushButton_5.setObjectName("pushButton_5")
        self.pushButton_5.clicked.connect(self.onClick_Button_Wav)
        self.pushButton_5.setStyleSheet(
            'QPushButton{background:#CABEFF;border-radius:5px;}QPushButton:hover{background:HotPink;} ')  # 设置按键滤波颜色
```

a、播放提供歌单歌曲

```
def play_select(self,i):
        list=['Mojito.wav','bad guy.wav','Alright.wav','Jam Jam.wav'] # 歌单
        Func_play_audio.play_audio(list[i]) # 调用play_audio()函数播放
```

b、播放本地音频文件

```
# 点击选择音频
def onClick_Button_Wav(self):
        fname = self.onClickOpen()
        self.Myplayer(fname)
# 打开所选音频
def onClickOpen(self):
        fname, _ = QtWidgets.QFileDialog.getOpenFileName(self, '打开文件', '.', '音频文件(*.wav)')
        return fname
# 调用play_audio()函数播放
def Myplayer(self, fname):
        Func_play_audio.play_audio(fname)
```

```
# play_audio()函数
import wave
import subprocess
def play_audio(wave_path):

    chunk = 1024 #声卡设置的帧数
    wf = wave.open(wave_path, 'rb')
    p = subprocess.call(['afplay',wave_path])
```

2、可视化音频

```
# 自选歌曲可视化
        self.cb1 = QComboBox(self)
        self.cb1.setGeometry(QtCore.QRect(530, 540, 81, 26))
        self.labelcb = QtWidgets.QLabel("原始波形", self)
        self.labelcb.setGeometry(QtCore.QRect(530, 500, 81, 26))
        self.cb1.addItem('Mojito - 周杰伦')
        self.cb1.addItem('Bad guy - Billie Eilish')
        self.cb1.addItem('Alright - CYN')
        self.cb1.addItem('Jam Jam - IU')
        self.cb1.currentIndexChanged[str].connect(self.print_value)  # 打印所选歌曲名
        self.cb1.currentIndexChanged[int].connect(self.draw_select)  # 通过所选歌曲索引播放歌曲
        self.cb1.setStyleSheet('QComboBox{background-color:#FF95CA;} ')  # 设置颜色

# 自定义噪声可视化
        self.pushButton_5 = QtWidgets.QPushButton("正弦噪声",self)
        self.pushButton_5.setGeometry(QtCore.QRect(730, 500, 81, 26))
        self.pushButton_5.setObjectName("pushButton_5")
        self.pushButton_5.clicked.connect(self.onClick_Button_draw_noise)
        self.pushButton_5.setStyleSheet(
            'QPushButton{background:#CABEFF;border-radius:5px;}QPushButton:hover{background:HotPink;} ')  # 设置按键正弦噪声颜色
```

```
# 自选歌曲加噪可视化
        self.cb2 = QComboBox(self)
        self.cb2.setGeometry(QtCore.QRect(940, 540, 81, 26))
        self.labelcb = QtWidgets.QLabel("加噪音频", self)
        self.labelcb.setGeometry(QtCore.QRect(940, 500, 81, 26))
        self.cb2.addItem('Mojito - 周杰伦')
        self.cb2.addItem('Bad guy - Billie Eilish')
        self.cb2.addItem('Alright - CYN')
        self.cb2.addItem('Jam Jam - IU')
        self.cb2.currentIndexChanged[str].connect(self.print_value)  # 打印所选歌曲名
        self.cb2.currentIndexChanged[int].connect(self.draw_select_mix)  # 通过所选歌曲索引播放歌曲
        self.cb2.setStyleSheet('QComboBox{background-color:#FF95CA;} ') # 设置颜色
```

```
# 画原音频波形和频谱
def draw_select(self,i):
        list = ['Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单
        F1 = MyFigure(width=5, height=4, dpi=100)

        # 返回所选曲目采样率，波形，采样点数
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs)

        # 所选歌曲频谱
        Freq, A_fft = Func_fft_audio.fft_func(wave_data, N, fs)

        # 画图
        F1.plotWave(time, N, wave_data, Freq, A_fft)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F1)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口

# 画加噪音频波形和频谱
def draw_select_mix(self,i):
        list = ['Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单
        F3 = MyFigure(width=5, height=4, dpi=100)

        # 返回所选曲目采样率，波形，采样点数
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs)

        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理

        # 带噪音频
        mix = wave_data + noise

        # 带噪音频的频谱
        Freq_mix, A_fft_mix = Func_fft_audio.fft_func(mix, N, fs)

        # 画图
        F3.plotWave(time, N, mix, Freq_mix, A_fft_mix)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F3)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口

# 画自定义噪声，时间参数设置与示例音频文件F215.wav相同
def onClick_Button_draw_noise(self):
        F2 = MyFigure(width=5, height=4, dpi=100)

        # 示例音频文件F215.wav
        fs, wave_data, N = Func_read_audio.read_audio("F215.wav")
        time = np.arange(0, N) * (1.0 / fs)

        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理
```

```
        # 噪声的频谱
        Freq_noise, A_fft_noise = Func_fft_audio.fft_func(noise, N, fs)

        # 画图
        F2.plotWave(time,N,noise,Freq_noise, A_fft_noise)
```

```
def plotWave(self,time,N,wave_data,Freq,A_fft):
        # 画波形
        self.axes1 = self.fig.add_subplot(221)
        self.axes1.plot(time, wave_data)
        self.axes1.set_title('Waveform' )
        # 画频谱
        self.axes2 = self.fig.add_subplot(222)
        self.axes2.plot(time, wave_data, Freq[0:(N + 1) // 2], A_fft)
        self.axes2.set_title('FFT')

 def plotWave2(self,N,Freq_mix,A_fft_mix,w,h,Freq_filted,fft_dataFilted):
        # 画频谱
        self.axes1 = self.fig.add_subplot(221)
        self.axes1.plot(Freq_mix[0:(N + 1) // 2], A_fft_mix, 'b-')
        self.axes1.set_title('Unfilted')
        # 画滤波器
        self.axes2 = self.fig.add_subplot(222)
        self.axes2.plot(w, abs(h), 'b-')
        self.axes2.set_title('Filter')
        # 画滤波后频谱
        self.axes3 = self.fig.add_subplot(223)
        self.axes3.plot(Freq_filted[0:(N + 1) // 2], fft_dataFilted, 'b-')
        self.axes3.set_title('Filted')
```

3、滤波

```
# 滤波
# 滤波器输入参数设置
        self.groupBox_2 = QtWidgets.QGroupBox("滤波器参数",self)
        self.groupBox_2.setGeometry(QtCore.QRect(120, 150, 351, 131))
        self.groupBox_2.setAutoFillBackground(False)
        self.groupBox_2.setObjectName("groupBox_2")
        self.groupBox_2.setStyleSheet('QGroupBox{background-color:#ECECFF;} ')

        self.label = QtWidgets.QLabel("单位：Hz",self.groupBox_2)
        self.label.setGeometry(QtCore.QRect(20, 20, 60, 16))
        self.label.setObjectName("label")

        self.label_1 = QtWidgets.QLabel("FPass",self.groupBox_2)
        self.label_1.setGeometry(QtCore.QRect(10, 50, 60, 16))
        self.label_1.setObjectName("label_1")

        self.label_2 = QtWidgets.QLabel("FStop",self.groupBox_2)
        self.label_2.setGeometry(QtCore.QRect(10, 90, 60, 16))
        self.label_2.setObjectName("label_2")

        self.lineEditFPass = QtWidgets.QLineEdit("4000", self.groupBox_2) # 默认截止频率1为4000Hz，可自行输入
        self.lineEditFPass.setGeometry(QtCore.QRect(50, 50, 113, 22))
        self.lineEditFPass.setAlignment(Qt.AlignCenter)
        self.lineEditFPass.setFrame(False)
        self.lineEditFPass.setObjectName("lineEditFPass")

        self.lineEditFStop = QtWidgets.QLineEdit("6000",self.groupBox_2) # 默认截止频率2为6000Hz，可自行输入，在带通/带阻滤波时设置
        self.lineEditFStop.setGeometry(QtCore.QRect(50, 90, 113, 22))
        self.lineEditFStop.setAlignment(Qt.AlignCenter)
        self.lineEditFStop.setFrame(False)
        self.lineEditFStop.setObjectName("lineEditFStop")

# 滤波器类型
        self.groupBox = QtWidgets.QGroupBox("滤波器类型",self)
        self.groupBox.setGeometry(QtCore.QRect(130, 30, 341, 91))
        self.groupBox.setObjectName("groupBox")
```

```
        self.groupBox.setStyleSheet('QGroupBox{background-color:#ECECFF;} ')

        self.radioButtonLP = QtWidgets.QRadioButton("低通滤波器",self.groupBox)
        self.radioButtonLP.setGeometry(QtCore.QRect(20, 20, 97, 21))
        self.radioButtonLP.setObjectName("radioButtonLP")
        self.radioButtonLP.clicked.connect(self.onClick_radioButtonLP_Filter)

        self.radioButtonHP = QtWidgets.QRadioButton("高通滤波器",self.groupBox)
        self.radioButtonHP.setGeometry(QtCore.QRect(210, 20, 97, 21))
        self.radioButtonHP.setObjectName("radioButtonHP")
        self.radioButtonHP.clicked.connect(self.onClick_radioButtonHP_Filter)


        self.radioButtonBP = QtWidgets.QRadioButton("带通滤波器", self.groupBox)
        self.radioButtonBP.setGeometry(QtCore.QRect(20, 60, 97, 21))
        self.radioButtonBP.setObjectName("radioButtonBP")
        self.radioButtonBP.clicked.connect(self.onClick_radioButtonBP_Filter)

        self.radioButtonBS = QtWidgets.QRadioButton("带阻滤波器", self.groupBox)
        self.radioButtonBS.setGeometry(QtCore.QRect(210, 60, 97, 21))
        self.radioButtonBS.setObjectName("radioButtonBS")
        self.radioButtonBS.clicked.connect(self.onClick_radioButtonBS_Filter)

# 选择歌曲滤波
        self.cb4 = QComboBox(self)
        self.cb4.setGeometry(QtCore.QRect(180, 360, 181, 96))
        self.labelcb2 = QtWidgets.QLabel("滤波", self)
        self.labelcb2.setGeometry(QtCore.QRect(150, 300, 241, 31))
        # 添加下拉框内容
        self.cb4.addItem("F215.wav")
        self.cb4.addItem('Mojito - 周杰伦')
        self.cb4.addItem('Bad guy - Billie Eilish')
        self.cb4.addItem('Alright - CYN')
        self.cb4.addItem('Jam Jam - IU')
        self.cb4.currentIndexChanged[str].connect(self.print_value)  # 打印所选歌曲名
        self.cb4.currentIndexChanged[int].connect(self.onClick_Button_Filter)  # 通过所选歌曲索引播放歌曲
        self.cb4.setStyleSheet('QComboBox{background-color:#FF95CA;} ')
```

a、高通滤波

```
def onClick_radioButton_HP(self,i):
        F5 = MyFigure(width=5, height=4, dpi=100)

        list = ["F215.wav", 'Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单

        # 返回所选曲目采样率，波形，采样点数
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs)

        fp = int(self.lineEditFPass.text())  # 读取截止频率
        Wn = 2 * fp / fs  # 通过截止频率和采样频率计算滤波器归一化截止频率Wn
        print("您选择的截止频率是{}".format(fp))

        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理

        # 带噪音频
        mix = wave_data + noise

        # 带噪音频的频谱
        Freq_mix, A_fft_mix = Func_fft_audio.fft_func(mix, N, fs)

        b, a = signal.butter(8, Wn, 'highpass')
        # b，a: IIR滤波器的分子（b）和分母（a）多项式系数向量。
        # 配置滤波器 8 表示滤波器的阶数
        w, h = scipy.signal.freqz(b, a, fs=fs)
        # w： ndarray，计算 h 的频率，单位与 fs 相同。默认情况下，w 被归一化为范围 [0, pi)(弧度/样本)。 h： ndarray，频率响应，作为复数。

        filtedData = signal.filtfilt(b, a, mix)  # mix为要过滤的带噪信号
        Freq_filted, fft_dataFilted = Func_fft_audio.fft_func(filtedData, N, fs)  # 过滤后的音频频谱
```

```
        # 画图
        F5.plotWave2(N, Freq_mix, A_fft_mix, w, h, Freq_filted, fft_dataFilted)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F5)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口
```

b、低通滤波

```
def onClick_radioButton_LP(self,i):
        F4 = MyFigure(width=5, height=4, dpi=100)

        list = ["F215.wav", 'Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单

        # 返回所选曲目采样率，波形，采样点数
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs) # 时间

        fp = int(self.lineEditFPass.text()) # 读取截止频率
        Wn = 2*fp/fs # 通过截止频率和采样频率计算滤波器归一化截止频率Wn
        print("您选择的截止频率是{}".format(fp))

        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理

        # 带噪音频
        mix = wave_data + noise

        # 带噪音频的频谱
        Freq_mix, A_fft_mix = Func_fft_audio.fft_func(mix, N, fs)

        b, a = signal.butter(8, Wn, 'lowpass')
        # b，a: IIR滤波器的分子（b）和分母（a）多项式系数向量。
        # 配置滤波器 8 表示滤波器的阶数
        w, h = scipy.signal.freqz(b, a, fs=fs) # 输出的w,h用于作图；w： ndarray，计算 h 的频率，单位与 fs 相同。  h： ndarray，频率响应，作为复数。

        filtedData = signal.filtfilt(b, a, mix)  # mix为要过滤的带噪信号
        Freq_filted, fft_dataFilted = Func_fft_audio.fft_func(filtedData, N, fs)  # 过滤后的音频频谱

        # 画图
        F4.plotWave2(N, Freq_mix, A_fft_mix, w, h, Freq_filted, fft_dataFilted)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F4)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口
```

c、带阻滤波

```
def onClick_radioButton_BS(self,i):
        F7 = MyFigure(width=5, height=4, dpi=100)

        list = ["F215.wav",'Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单

        # F215.wav
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs)

        fp = int(self.lineEditFPass.text())  # 读取截止频率1
        fp2 = int(self.lineEditFStop.text())  # 读取截止频率2
        Wn1 = 2 * fp / fs  # 通过截止频率1和采样频率计算滤波器归一化截止频率Wn1
        Wn2 = 2 * fp2 / fs  # 通过截止频率2和采样频率计算滤波器归一化截止频率Wn2
        print("您选择的截止频率1是{0},您选择的截止频率2是{1}".format(fp, fp2))

        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理

        # 带噪音频
```

```
        mix = wave_data + noise

        # 带噪音频的频谱
        Freq_mix, A_fft_mix = Func_fft_audio.fft_func(mix, N, fs)

        b, a = signal.butter(8, [Wn1,Wn2], 'bandstop')
        # b, a: IIR滤波器的分子（b）和分母（a）多项式系数向量。
        # 配置滤波器 8 表示滤波器的阶数
        w, h = scipy.signal.freqz(b, a, fs=fs)
        # w： ndarray，计算 h 的频率，单位与 fs 相同。默认情况下，w 被归一化为范围 [0, pi)(弧度/样本)。 h： ndarray，频率响应，作为复数。

        filtedData = signal.filtfilt(b, a, mix)  # mix为要过滤的带噪信号
        Freq_filted, fft_dataFilted = Func_fft_audio.fft_func(filtedData, N, fs)  # 过滤后的音频频谱

        # 画图
        F7.plotWave2(N, Freq_mix, A_fft_mix, w, h, Freq_filted, fft_dataFilted)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F7)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口
```

d、带通滤波

```
def onClick_radioButton_BP(self,i):
        F6 = MyFigure(width=5, height=4, dpi=100)
        list = ["F215.wav", 'Mojito.wav', 'bad guy.wav', 'Alright.wav', 'Jam Jam.wav']  # 歌单

        # 返回所选曲目采样率，波形，采样点数
        fs, wave_data, N = Func_read_audio.read_audio(list[i])
        time = np.arange(0, N) * (1.0 / fs)

        fp = int(self.lineEditFPass.text())  # 读取截止频率1
        fp2 = int(self.lineEditFStop.text()) # 读取截止频率2
        Wn1 = 2 * fp / fs  # 通过截止频率1和采样频率计算滤波器归一化截止频率Wn1
        Wn2 = 2 * fp2 / fs  # 通过截止频率2和采样频率计算滤波器归一化截止频率Wn2
        print("您选择的截止频率1是{0},您选择的截止频率2是{1}".format(fp,fp2))


        # 正弦噪声
        f1 = 4400
        f2 = 6000
        noise = 1.2 + np.sin(f1 * 2 * np.pi * time) + 2.5 * np.cos(f2 * 2 * np.pi * time)  # 设置noise的波形
        noise = noise * 1.0 / (max(abs(noise)))  # 归一化处理

        # 带噪音频
        mix = wave_data + noise

        # 带噪音频的频谱
        Freq_mix, A_fft_mix = Func_fft_audio.fft_func(mix, N, fs)

        b, a = signal.butter(8, [Wn1,Wn2], 'bandpass')
        # b, a: IIR滤波器的分子（b）和分母（a）多项式系数向量。
        # 配置滤波器 8 表示滤波器的阶数
        w, h = scipy.signal.freqz(b, a, fs=fs)
        # w： ndarray，计算 h 的频率，单位与 fs 相同。默认情况下，w 被归一化为范围 [0, pi)(弧度/样本)。 h： ndarray，频率响应，作为复数。

        filtedData = signal.filtfilt(b, a, mix)  # mix为要过滤的带噪信号
        Freq_filted, fft_dataFilted = Func_fft_audio.fft_func(filtedData, N, fs)  # 过滤后的音频频谱

        # 画图
        F6.plotWave2(N, Freq_mix, A_fft_mix, w, h, Freq_filted, fft_dataFilted)

        self.scene = QGraphicsScene()  # 创建一个场景
        self.scene.addWidget(F6)  # 将图形元素添加到场景中
        self.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口f.graphicsView.setScene(self.scene)  # 将创建添加到图形视图显示窗口
```

```
# 控制参数输入，进行滤波
    def onClick_Button_Filter(self,i):
        if self.radioButtonLP.isChecked():
            self.onClick_radioButton_LP(i)
        elif self.radioButtonHP.isChecked():
```

```
            self.onClick_radioButton_HP(i)
        elif self.radioButtonBP.isChecked():
            self.onClick_radioButton_BP(i)
        elif self.radioButtonBS.isChecked():
            self.onClick_radioButton_BS(i)

    def onClick_radioButtonLP_Filter(self):
        self.lineEditFStop.setDisabled(True)
        self.lineEditFStop.setReadOnly(True)

    def onClick_radioButtonHP_Filter(self):
        self.lineEditFStop.setDisabled(True)
        self.lineEditFStop.setReadOnly(True)

    def onClick_radioButtonBP_Filter(self):
        self.lineEditFPass.setDisabled(False)
        self.lineEditFPass.setReadOnly(False)
        self.lineEditFStop.setDisabled(False)
        self.lineEditFStop.setReadOnly(False)

    def onClick_radioButtonBS_Filter(self):
        self.lineEditFPass.setDisabled(False)
        self.lineEditFPass.setReadOnly(False)
        self.lineEditFStop.setDisabled(False)
        self.lineEditFStop.setReadOnly(False)
```

五、对本课程的意见和建议

本人认为，数字音频处理课程是第一门十分优秀的音频处理相关的专业实践课程。不仅教授了相关音频处理的代码实现方法，为今后的科研项目和毕业设计打下了一定的编程基础；也使我掌握了入门级的软件设计方法，学以致用，十分收益。

网课部分的音频专业知识对本人而言有些高深，难以理解和记忆，不确定是否在今后专业领域的具体哪部分会加以应用。如果能更加结合本专业课程去学那些专业的乐理知识或许能更容易接受。

六、未来工作

1、如果你有机会参与Audition CC2023的设计，你会考虑什么新功能？

a、我会增加免费的纯净的音频素材库，供入门级新手使用。也可以考虑允许用户上传自己的音频作品，供其他用户学习参考。

b、我会考虑简化音频空间化功能，将音频片段实现指定空间坐标插入，并指定声源进行空间移动（例如：希望将音频片段A插入到听者右后方45度距离约3米处，并以听者为圆心以1m/s的速度沿圆弧移动到听者正前方，该操作过程只需输入期望声源的空间坐标/移动方程，声源的移动速度等参数即可实现）

c、优化去噪功能，在课程作业去噪时，存在部分非噪声的消除和噪声消除不彻底现象。