# 探索实验一 简易数字音乐合成器的设计

Github项目地址：https://github.com/Wang-Yuhe/digital-synthesizer

|  |  |  |
| --- | --- | --- |
| 学号 | 姓名 | 分工 |
| 22355082 | 王宇和 | 1.整体框架设计  2.Track、NoteBlock类、DigitalSynthesizer类的实现  3.flute和piccolo音色的实现  4. timbre\_analysis分析单音谐波成分 |
| 23320143 | 唐雅妍 | 1.Note类的实现  2.adsr包络,振荡器,filter滤波器的实现  3.piano和harp音色的实现  4.在timbre\_analysis基础上增加识别一段旋律的基频，并分析每个基频的谐波成分（在timbre\_analysis\_test） |
| 23336099 | 黄钲权 | 1.整体框架设计  2. |

## 问题描述

简易数字音乐合成器的设计

## 问题分析与解决思路

数字音乐合成器是通过对不同频率产生信号的处理来模拟自然界发声，其中的方法为：

1. 选择合适的振荡器（最常用的为正弦波，也有方波，锯齿波，三角波），再根据频率合成对应的基波
2. 进行谐波成分的叠加，因为现实中的音色很大程度上是由谐波成分来决定的，具体方法是将频率整数倍的谐波在时域上相加
3. Adsr包络模拟振幅的变化，a为attack表示音量从0达到最大的时间，d为decay表示下降到sustain level的时间，r为release表示从sustain level下降到0的衰减时间，而s表示sustain，这个时间长度由演奏者决定，表示保持音量在sustain level的时间
4. 在频率上应用巴特沃斯滤波器，使音色更加和谐

## 实验代码

单音谐波分析：

用yin方法得到基频，通过傅里叶变换得到频率图，获取整数倍频率的能量

# 傅里叶变换

spectrum = np.fft.rfft(waveform)

freqs = np.fft.rfftfreq(len(waveform), d=1/sample\_rate)

magnitude = np.abs(spectrum)

# 检测基频

f0 = librosa.yin(waveform, fmin=50, fmax=2000, sr=sample\_rate)[0]

# 提取前10个谐波的幅值

harmonics = []

for n in range(1, 17):

    harmonic\_freq = f0 \* n

    idx = np.argmin(np.abs(freqs - harmonic\_freq))

    harmonics.append(magnitude[idx])

# 归一化

harmonics = np.array(harmonics)

harmonics /= harmonics[0]  # 基频幅值归一为 1.0

滤波器：

用butter巴特沃斯滤波器进行滤波，lfilter应用与waveform波形上

def lowpass\_filter(waveform,  wc, N=4, sample\_rate=44100):#wc截止频率,butter平稳的滤波器

    b,a = butter(N, wc, fs=sample\_rate)

    #scipy 的滤波器设计函数要求频率是相对于 奈奎斯特频率（Nyquist = sample\_rate / 2）的比例

    #用fs参数不需要手动除了

    return lfilter(b, a, waveform)

adsr包络：

分段建立0~1,1~sustain\_level, 保持, sustain\_level~0的线性函数，然后相乘以包络

"""增加adsr包络曲线"""

total\_samples = len(waveform)

attack\_samples = int(sample\_rate \* attack\_time)

decay\_samples = int(sample\_rate \* decay\_time)

release\_samples = int(sample\_rate \* release\_time)

sustain\_samples = total\_samples - (attack\_samples + decay\_samples + release\_samples)

sustain\_samples = max(sustain\_samples, 0)

# 各阶段的包络

attack\_env = np.linspace(0, 1, attack\_samples)

decay\_env = np.linspace(1, sustain\_level, decay\_samples)

sustain\_env = np.full(sustain\_samples, sustain\_level)#保持不变

release\_env = np.linspace(sustain\_level, 0, release\_samples)

envelope = np.concatenate([attack\_env, decay\_env, sustain\_env, release\_env])

# 截断或填补以匹配 wave 长度

if len(envelope) > total\_samples:

    envelope = envelope[:total\_samples]

else:

    envelope = np.pad(envelope, (0, total\_samples - len(envelope)))

waveform\*=envelope

音块（一小节）产生波形：

从Note获取到波形图时移后叠加

max\_len = 0

# 填充不从0开始的音符

for note in self.notes:

    if self.start\_beat[note.note\_id] > 0:

        note.generate\_waveform()

        # 在音符开始前添加休止符

        silence = np.zeros(int(self.start\_beat[note.note\_id] / self.bpm \* 60 \* self.sample\_rate))

        note.waveform = np.concatenate((silence, note.waveform))

        max\_len = max(max\_len, len(note.waveform))

    else:

        note.generate\_waveform()

        max\_len = max(max\_len, len(note.waveform))

# 填充音符长度不一致的音符

for note in self.notes:

    if len(note.waveform) < max\_len:

        silence = np.zeros(max\_len - len(note.waveform))

        note.waveform = np.concatenate((note.waveform, silence))

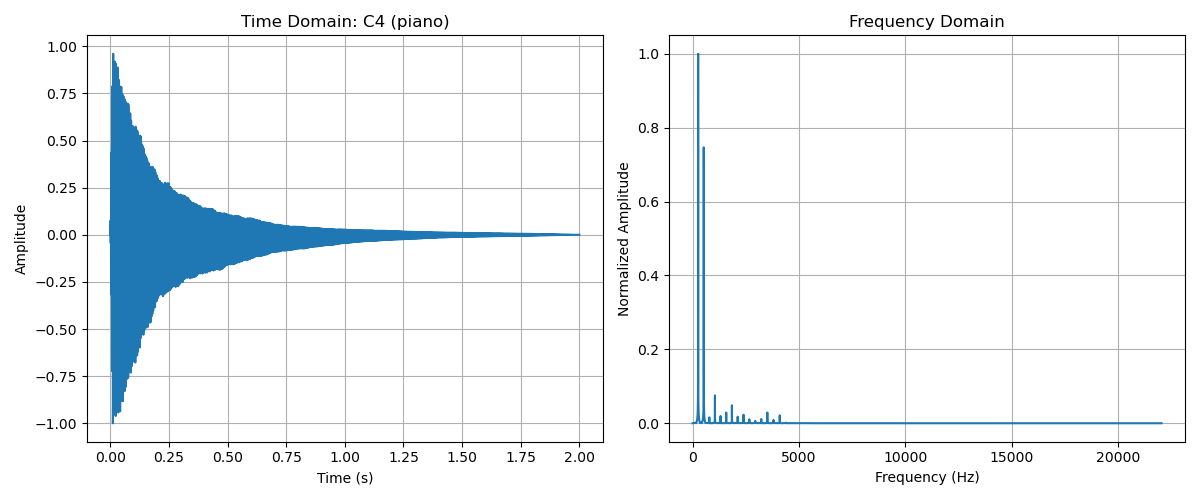
notes\_waveform = [note.waveform for note in self.notes]

self.waveform = sum(notes\_waveform) if len(notes\_waveform) > 1 else notes\_waveform[0]

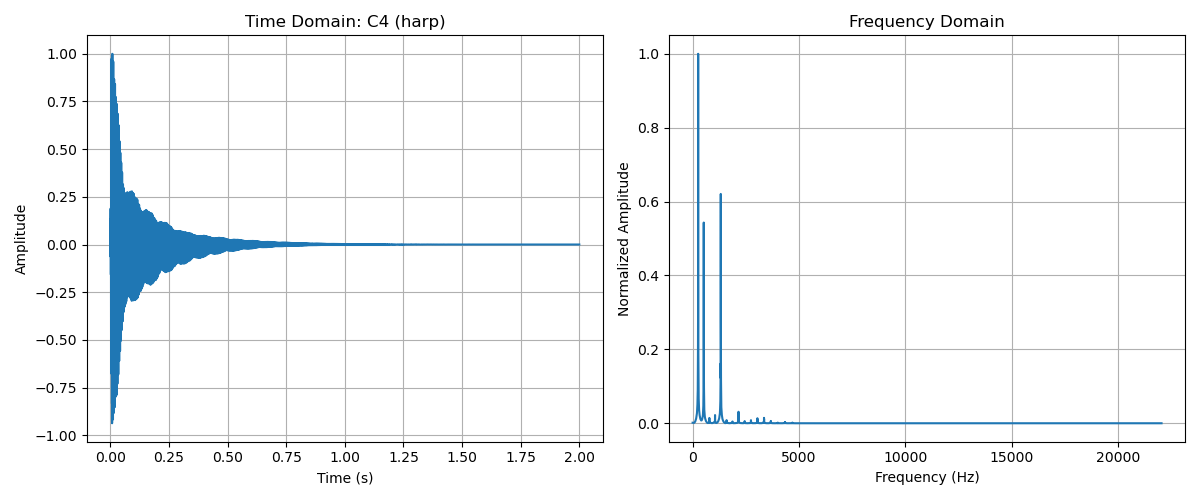
## 实验结果

在采样率44100，bpm60下，节拍数为2，音量为1的条件，C4音下

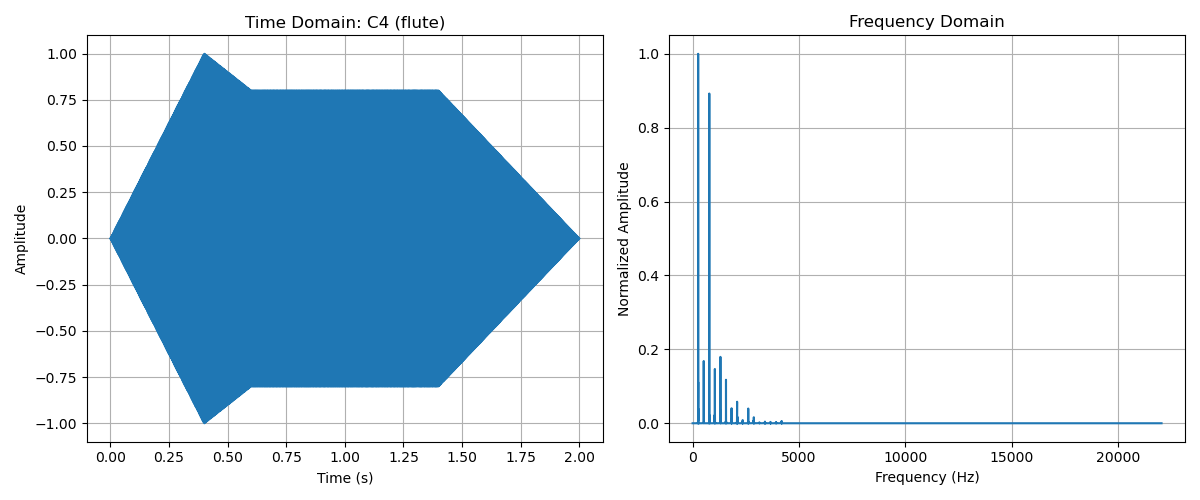
进行音色调制后得到的钢琴时域和频域如下：



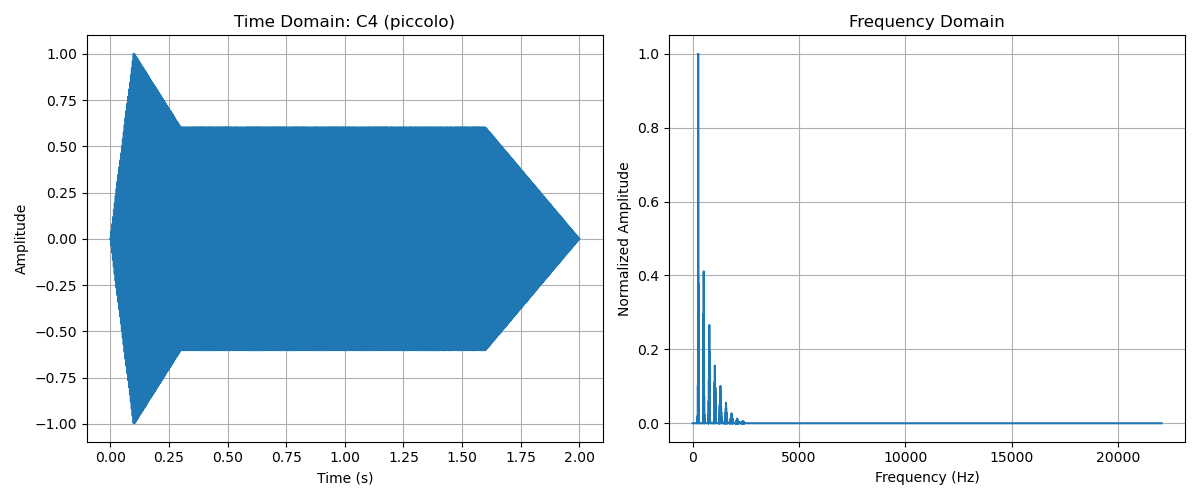
竖琴的时域和频率如下：



长笛的时域和频率如下：



短笛的时域和频率如下：



（运行Note.py即可看到）

运行main.py可以听到合成音色后的生日快乐歌：

在timbre中可以选择piano, piccolo, harp, flute选择

在volume可以在0~1中调节音量

其中.wav文件为生成的音频效果（与运行的效果是一样的）

## 结论

可以通过在时域和频率上操作用以模拟自然音色

## 收获与感想

每位同学结合自己分工写100字左右的个人收获、感想等。

## 创新点

1. 对单音的开头增加白噪音以模拟击键时的效果，对每个谐波成分都进行衰减包络（其中低频到高频动态变化，高频的衰减频率更高），对基波和谐波的频率进行微微失谐以模拟金属弦的特性

具体代码如下：

waveform=np.zeros\_like(t)

for i, amplitude in enumerate(harmonics):

    amp\_freq = freq\*(i+1) \* np.sqrt(1+B\*(i+1)\*\*2)#现实中的弦不会产生完美的谐波（非谐波性）

    waveform += amplitude\*np.exp(-decays[i]\*t) \* oscillator('sine', amp\_freq, t)

noise = np.random.randn(len(t))

attack\_time = 0.02

noise\_decay = np.linspace(1,0, int(attack\_time\*sample\_rate))

noise\_decay = np.pad(noise\_decay, (0, len(t)-len(noise\_decay)), mode='constant')\*0.2#白噪音包络线

waveform += noise \* noise\_decay\*volume#增加敲键的白噪音

1. 用python的库pyin算法获取音频的基波，yin通过在时域上每一帧对波形进行时移来获取每一帧的基频，pyin的改进为对结果进行平滑以减少半倍或两倍误差；再通过短时傅里叶变换（即对每一帧做傅里叶变换来使频率图能随着时间变化），对每一帧的频率图做谐波分析，具体方法为对于该基频，找到频率i倍，周围范围5%的能量最大的值，最后再以基频为单位1做归一化

# 基频跟踪

f0\_series, voiced\_flag, voiced\_prob = librosa.pyin(y, fmin=fmin, fmax=fmax,

                sr=sr,frame\_length=frame\_length, hop\_length=hop\_length)#pyin效果更好

S = np.abs(librosa.stft(y, n\_fft=frame\_length, hop\_length=hop\_length))#短时傅里叶变换

#S[f, t]第t帧频率f上的振幅

freqs = np.linspace(0, sr/2, num=S.shape[0])

result = []

result\_notename = []

for t, f0 in enumerate(f0\_series):

    if np.isnan(f0) or f0 <= 0 or not voiced\_flag[t] or voiced\_prob[t]<0.3:

        continue

    mags = []

    for h in range(1, n\_harmonics + 1):

        #原来的谐波成分是找频率点，而事实上有时并不完全为整数倍，这里找范围内峰值最高的

        target\_freq = f0 \* h

        search\_radius = 0.05 \* target\_freq

        mask = np.where((freqs >= target\_freq - search\_radius) & (freqs <= target\_freq + search\_radius))[0]

        if len(mask) == 0:

            mags.append(0.0)

            continue

        local\_mags = S[mask, t]

        max\_idx\_in\_window = np.argmax(local\_mags)

        mags.append(local\_mags[max\_idx\_in\_window])

    mags = np.array(mags)

    rel = (mags / mags[0]).tolist()

    result\_notename.append(librosa.midi\_to\_note(librosa.hz\_to\_midi(f0)))

    result.append(rel)