# 基于 Python 的随机拨弦与泛音生成模型

数学科学学院　PB22010407　林晓烁

2023 年 6 月 8 日

## 1　一维波动方程的物理背景

从弦振动模型抽象得到的一维波动方程形如

$$\rho\frac{\partial^2 y}{\partial t^2}=T\frac{\partial^2 y}{\partial x^2},$$

其中 $\rho$ 是弦线的线密度 (我们认为弦线质地均匀, 从而 $\rho$ 为常数), $T$ 是弦中张力, 并设弦长为 $l$. 通过引入波速 $v=\sqrt{\dfrac{T}{\rho}}$, 可以将上述方程改写为

$$\frac{\partial^2 y}{\partial t^2}=v^2\frac{\partial^2 y}{\partial x^2}.$$

不失一般性地, 可以取 $v$ 的数值为 1 将问题简化为 $y_{tt}=y_{xx}$. 再定义边界条件

$$\begin{cases}y(0,t)=\varphi(t)\\ y(l,t)=\psi(t)\end{cases}$$

以及初始条件

$$\begin{cases}y(x,0)=f(x)\\ y_t(x,0)=g(x)\end{cases}.$$

由于弦乐器弦线的两个端点均为固定端点, 以上条件中 $\varphi(t)=\psi(t)\equiv 0$, 且 $f(0)=f(l)=g(0)=g(l)\equiv 0$, 即我们处理的是两端固定的随机拨弦问题. 一般情况下拨弦时偏移量均在 $y$ 的同一方向, 故不妨要求对任意 $x\in[0,l]$ 都有 $f(x)\geqslant 0$.

## 2　基于有限差分法的随机拨弦问题求解

### 2.1　一维波动方程的离散化处理

我们用 $h$ 表示 $\mathbb{R}^3$ 中沿 $x$ 轴正方向的步长, 用 $k$ 表示沿 $t$ 轴正方向的步长, 将涉及的区域离散化如下:

$$\begin{cases}x_i=ih, & 0\leqslant i\leqslant n,\\ y_j=jk, & j\geqslant 0.\end{cases}$$

写出 $y$ 关于 $x$ 的二阶中心差分公式

$$y_{xx}(ih, jk) \approx \frac{1}{h^2}\left(y_{i+1,j} - 2y_{i,j} + y_{i-1,j}\right),$$

以及 $y$ 关于 $t$ 的二阶中心差分公式

$$y_{tt}(ih, jk) \approx \frac{1}{k^2}\left(y_{i,j+1} - 2y_{i,j} + y_{i,j-1}\right).$$

将以上两式代入一维波动方程 $y_{tt} = v^2 y_{xx}$ 整理可得有限差分近似公式

$$y_{i,j+1} = 2(1-s)y_{i,j} + s\left(y_{i+1,j} + y_{i-1,j}\right) - y_{i,j-1},$$

其中 $s = v^2\dfrac{k^2}{h^2}$. 接着对边界条件进行离散化:

$$y_{0,j} = \varphi(jk), \quad y_{n,j} = \psi(jk).$$

再对初始条件进行离散化:

$$y_{i,0} = f(ih),$$

以及利用 $y$ 关于 $t$ 的一阶中心差分公式

$$y_t(x,t) \approx \frac{y(x,t+k) - y(x,t-k)}{2k}$$

在 $t = 0$ 时的形式

$$y_t(x,0) = g(x) \approx \frac{y(x,k) - y(x,-k)}{2k}$$

得到

$$y_{i,1} - y_{i,-1} = 2kg(ih).$$

在前面得到的有限差分近似公式中令 $j = 0$, 并代入 $y_{i,0} = f(ih)$ 就得到

$$y_{i,1} + y_{i,-1} = 2f(ih) + s\left[f\left((i+1)h\right) - 2f(ih) + f\left((i-1)h\right)\right].$$

将其与前式相加化简可得

$$y_{i,1} = f(ih) + kg(ih) + \frac{s}{2}\left[f\left((i+1)h\right) - 2f(ih) + f\left((i-1)h\right)\right].$$

与此类似, 我们可以通过迭代计算出所有 $(n+1)(m+1)$ 个网格点上 $y$ 的模拟值 (如图 1 所示). 值得注意的是, 由数值分析的结论, 该算法仅在 $s \leqslant 1$ 的情形下稳定.

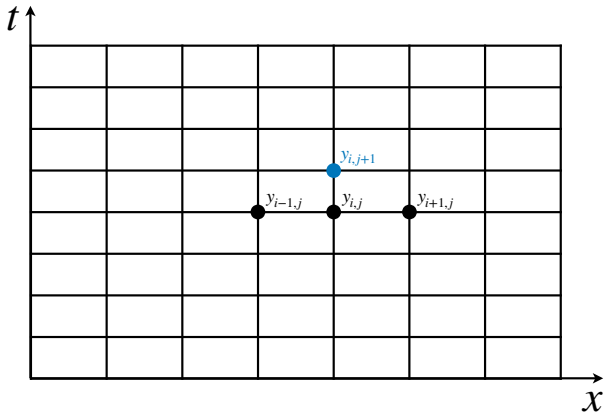


图 1: 有限差分法的网格点及计算方法

## 2.2 Python 代码实现

先将一维波动方程的有限差分解在 $\mathbb{R}^3$ 中呈现出来 (结果如图 2 所示). 其中使用命令行参数读取弦长、振动时间与波节数 (减去 1) 这三个参数, 并提供输出 PDF 或 JPG 格式图片文件的选项.

```
import numpy as np
import math
import matplotlib.pyplot as plt
import argparse
import sys

if __name__ == "__main__":
    parser = argparse.ArgumentParser(prog='Wave 3D', conflict_handler='resolve')
    parser.add_argument('-l', '--string_length', type=int,
                        help='string length: use a positive integer', default=10)
    parser.add_argument('-t', '--total_time', type=float,
                        help='define the time of vibration', default=20.0)
    parser.add_argument('-n', '--node', type=int,
                        help='number of nodes: use a positive integer', default=3)
    parser.add_argument('-p', '--pdf', type=bool,
                        help='need PDF output?', default=False)
    parser.add_argument('-j', '--jpg', type=bool,
                        help='need JPG output?', default=False)
    cmd = parser.parse_args(sys.argv[1:])
    parser.print_help()
    string_length = cmd.string_length # 弦线长度
```

```
        total_time = cmd.total_time # 总时间
        node = cmd.node # 波节数-1
        wave_velocity = 1 # 波速
        h = 0.01
        k = 0.01
        n = int(string_length / h)
        m = int(total_time / k)
        s = wave_velocity ** 2 * k ** 2 / h ** 2
        y = np.zeros([n + 1, m + 1])
        x_axis = np.linspace(0, string_length, n + 1)
        t_axis = np.linspace(0, total_time, m + 1)
        x, t = np.meshgrid(x_axis, t_axis)


        def f(x):
            return math.sin(node * math.pi * x / string_length)


        def g(x):
            return x * (string_length - x)


        def phi(t):
            return 0


        def psi(t):
            return 0


        # 对边界条件进行离散化
        for j in np.arange(0, m + 1):
            y[0, j] = phi(j * k)
            y[n, j] = psi(j * k)

        # 对初始条件进行离散化
        for i in np.arange(0, n + 1):
            y[i, 0] = f(i * h)
            y[i, 1] = f(i * h) + k * g(i * h) + s / 2.0 * (f((i + 1) * h)) - 2 * f(i
```

```
            * h) + f((i - 1) * h)

    # 求出网格点上y的值
    for j in np.arange(1, m - 1):
        for i in np.arange(1, n):
            y[i, j + 1] = 2 * (1 - s) * y[i, j] + s * y[i + 1, j] + s * y[i - 1,
                j] - y[i, j - 1]

    # 数值解结果可视化
    plt.rc('font', size=12)
    plt.rc('text', usetex=True)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    ax.set_xlabel(r'$x$')
    ax.set_ylabel(r'$t$')
    ax.set_zlabel(r'$y$')
    ax.plot_surface(x, t, y.T, cmap='rainbow')
    if cmd.pdf:
        plt.savefig('wave_3d.pdf') # 将结果保存为PDF文件
    if cmd.jpg:
        plt.savefig('wave_3d.jpg', dpi=100) # 将结果保存为JPG文件
    plt.show()
```

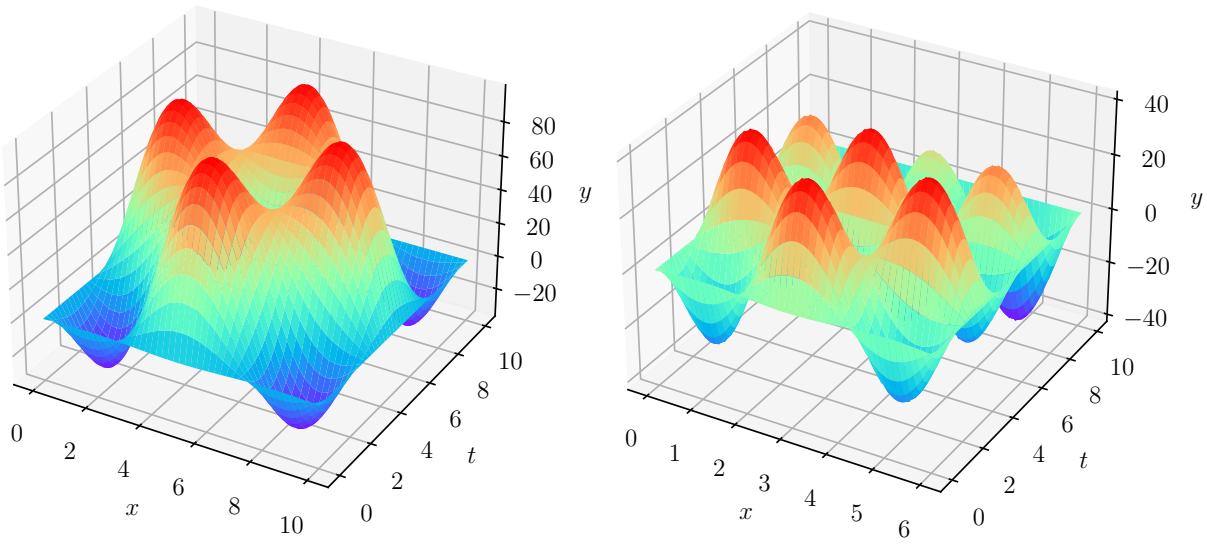


图 2: 模拟结果三维可视化 (不同弦长与时间)

再以二维平面上的动态过程展示计算结果 (如图 3 所示). 从动画中我们可以清晰地看出

对应于给定拨弦初态的弦振动过程. 其中使用命令行参数读取弦长、振动时间与波节数 (减去 1) 这三个参数, 并提供输出 GIF 或 MP4 格式动态文件的选项.

```
import numpy as np
import math
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
import argparse
import sys

if __name__ == "__main__":
    parser = argparse.ArgumentParser(prog='Wave Animation', conflict_handler='
        resolve')
    parser.add_argument('-l', '--string_length', type=int,
                        help='string length: use a positive integer', default=10)
    parser.add_argument('-t', '--total_time', type=float,
                        help='define the time of vibration', default=20.0)
    parser.add_argument('-n', '--node', type=int,
                        help='number of nodes: use a positive integer', default=3)
    parser.add_argument('-g', '--gif', type=bool,
                        help='need GIF output?', default=False)
    parser.add_argument('-m', '--mp4', type=bool,
                        help='need MP4 output?', default=False)
    cmd = parser.parse_args(sys.argv[1:])
    parser.print_help()

    string_length = cmd.string_length # 弦线长度
    total_time = cmd.total_time # 总时间
    node = cmd.node # 波节数-1
    wave_velocity = 1 # 波速
    h = 0.01
    k = 0.01
    n = int(string_length / h)
    m = int(total_time / k)
    s = wave_velocity ** 2 * k ** 2 / h ** 2
    y = np.zeros([n + 1, m + 1])
    x_axis = np.linspace(0, string_length, n + 1)
    t_axis = np.linspace(0, total_time, m + 1)
    x, t = np.meshgrid(x_axis, t_axis)
```

```
36
37
38          def f(x):
39              return math.sin(node * math.pi * x / string_length)
40
41
42          def g(x):
43              return x * (string_length - x)
44
45
46          def phi(t):
47              return 0
48
49
50          def psi(t):
51              return 0
52
53
54          # 对边界条件进行离散化
55          for j in np.arange(0, m + 1):
56              y[0, j] = phi(j * k)
57              y[n, j] = psi(j * k)
58
59          # 对初始条件进行离散化
60          for i in np.arange(0, n + 1):
61              y[i, 0] = f(i * h)
62              y[i, 1] = f(i * h) + k * g(i * h) + s / 2.0 * (f((i + 1) * h)) - 2 * f(i
                    * h) + f((i - 1) * h)
63
64          # 求出网格点上y的值
65          for j in np.arange(1, m - 1):
66              for i in np.arange(1, n):
67                  y[i, j + 1] = 2 * (1 - s) * y[i, j] + s * y[i + 1, j] + s * y[i - 1,
                        j] - y[i, j - 1]
68
69          # 初始化画布
70          plt.rc('font', size=12)
71          fig = plt.figure()
72          ax = fig.add_subplot(1, 1, 1)
```

```
73          ax.set_xlabel('Position x')
74          ax.set_ylabel('Deviation y')
75          line, = ax.plot(x_axis, x_axis, color="cornflowerblue", lw=3)
76          timer = ax.text(0.1, 0.9, '', transform=ax.transAxes)
77          amplitude = np.max(y)
78          ax.set_ylim(-1.2 * amplitude, 1.2 * amplitude)
79
80
81          # 清空当前帧
82          def init():
83              line.set_ydata([np.nan] * len(x_axis))
84              timer.set_text('')
85              return line, timer
86
87
88          # 更新下一帧的数据
89          def update(frame):
90              line.set_ydata(y[:, frame])
91              timer.set_text('t=' + str('{:.2f}'.format(frame * 0.01)) + 's')
92              return line, timer
93
94
95          # 调用FuncAnimation生成动画
96          ani = FuncAnimation(fig, update, init_func=init, frames=m, interval=1, blit=
                True)
97          plt.show()
98          if cmd.gif:
99              ani.save("animation.gif", fps=25, writer="pillow") # 将结果保存为GIF文件
100         if cmd.mp4:
101             ani.save("animation.mp4", fps=25, writer="ffmpeg") # 将结果保存为MP4文件
```

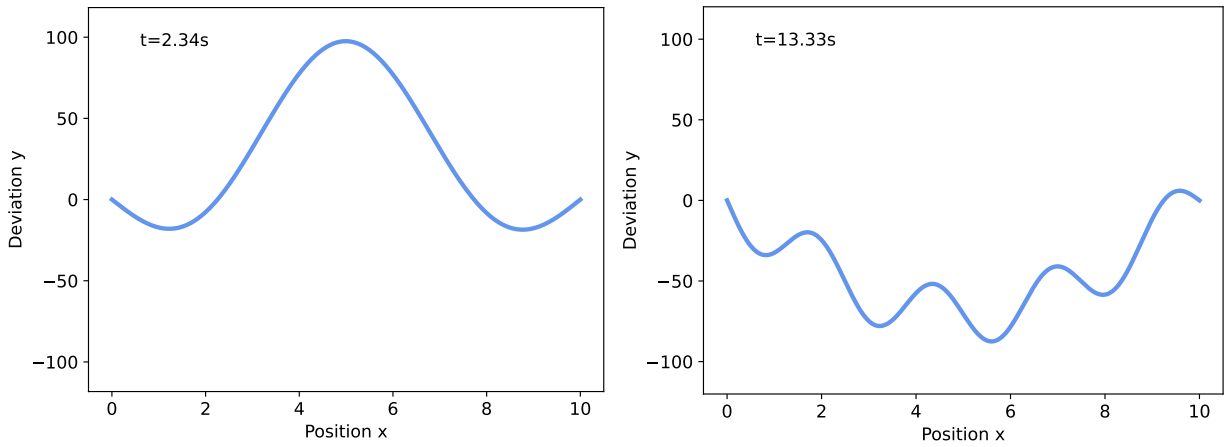


图 3: 模拟结果动态可视化 (不同时间、波节数对应的一帧画面)

# 3 音乐泛音的模拟

## 3.1 创建波形音频文件

利用 Python 中的 wave 模块可以方便地读写波形音频 (WAV) 文件. 以下是一个生成 5 秒的 220 Hz 正弦波音频片段的简单程序.

```
import math
import wave

import numpy as np

sRate = 44100 # 采样率为44100Hz(与CD所用的采样率相同)
nSamples = sRate * 5 # 生成一段5秒的音频
x = np.arange(nSamples) / float(sRate)
vals = np.sin(2.0 * math.pi * 220 * np.sqrt(2) * x) # 生成220 Hz正弦波音频片段
data = np.array(vals * 32767, 'int16').tobytes()
file = wave.open('sine220.wav', 'wb')
file.setparams((1, 2, sRate, nSamples, 'NONE', 'uncompressed')) # 使用单声道、两
    字节(16位)、无压缩格式
file.writeframes(data)
file.close()
```

### 3.2 基音与泛音

如果我们打开生成的音频文件, 会 “失望地” 发现这段音频似乎并不那么悦耳. 这里我们可以通过一段有关诺基亚手机铃声演变史的视频初探究竟.

在力学课程中我们学到过驻波的概念, 即入射波与反射波相干而形成的波形不再推进 (仅波腹上、下振动, 波节不移动) 的波. 当一个驻波只包含一个波腹时, 它被称作基音; 而当驻波包含 $n$ ($n \geqslant 2$) 个波腹时, 它就被称作 $n$ 次泛音. 事实上, 一根琴弦的振动, 不是仅由一个单调乏味的基音构成, 而是基音和多组频率与基音频率成整数倍数关系的驻波模式的组合, 这就是琴声比诸如刚才生成的 220 Hz 音频听起来更饱满的基本原理. (数学中调和级数 $\sum_{k=1}^{\infty} \frac{1}{k}$ 的名称便来源于此.)

泛音是吉他及其他弦乐器的常用技巧, 使用泛音技术可以发出更纯净、更明亮的音色. 下面我们将利用环形缓冲区实现对拨弦产生的声音的物理模拟.

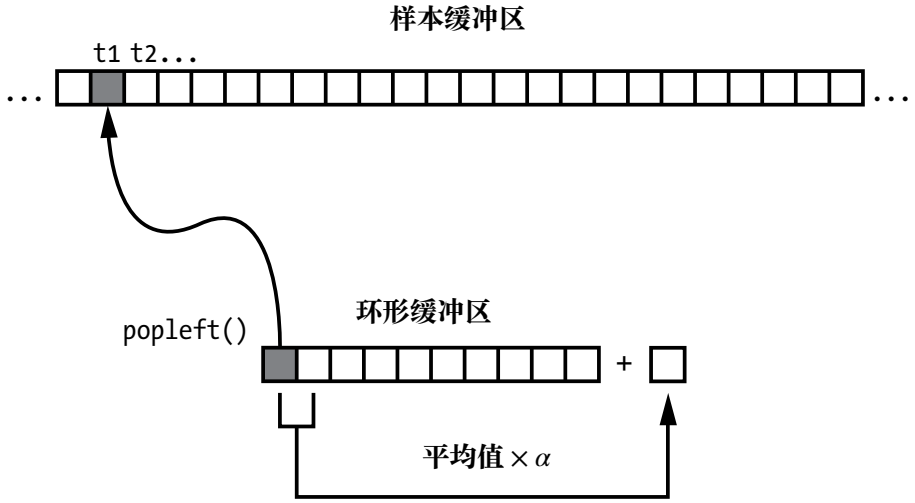


图 4: 借助环形缓冲区模拟拨弦

我们利用如图 4 所示的固定长度环形缓冲区模拟两端固定的吉他弦, 其特点是, 当到达缓冲区末端时, 下一个元素又重新回到缓冲区的头部. 我们通过向缓冲区随机填充 $[-0.5, 0.5]$ 内的数值来模拟初始状态下琴弦的偏移量, 然后循环进行以下步骤 (参考图 4):

1. 将环形缓冲区的第一个数值存入样本缓冲区.
2. 求出环形缓冲区前两个数值的平均值以达到切断较高频率的 “低通” 效果.
3. 将该平均值乘上衰减因子 $\alpha$ (选用 $\alpha = 0.995$) 来模拟波在弦上传播过程中的能量损失.
4. 将该值存入环形缓冲区末端.
5. 删除环形缓冲区的第一个数值.

### 3.3 Python 代码实现

以下代码实现了从 JSON 文件读取给定音符并模拟吉他音色演奏乐曲的过程.

```python
import json
import os
import random
import wave
from collections import deque
import time
import numpy as np
import pygame
import math

freq_dict_filename = r"freq_dict.json"
freq_list_filename = r"freq_list.json"
with open(freq_dict_filename) as f:
    freq_dict = json.load(f)
with open(freq_list_filename) as f:
    freq_list = json.load(f)


def sine(freq, time_len=1):
    sRate = 44100
    nSamples = int(np.ceil(sRate * time_len))
    x = np.arange(nSamples) / float(sRate)
    vals = np.sin(2.0 * math.pi * freq * x)
    data = np.array(vals * 32767, 'int16').tobytes()
    file = wave.open(f'notes/{freq}.wav', 'wb')
    file.setparams((1, 2, sRate, nSamples, 'NONE', 'uncompressed'))
    file.writeframes(data)
    file.close()


def writeWAVE(fname, data):
    file = wave.open(fname, 'wb')
    nChannels = 1
    sampleWidth = 2
    frameRate = 44100
    nFrames = 44100
```

```
37          file.setparams((nChannels, sampleWidth, frameRate, nFrames, 'NONE', '
                noncompressed'))
38          file.writeframes(data)
39          file.close()
40
41
42      def generateNote(freq):
43          nSamples = 44100
44          sampleRate = 44100
45          N = int(sampleRate / freq)
46          buf = deque([random.random() - 0.5 for i in range(N)])
47          samples = np.array([0] * nSamples, 'float32')
48          for i in range(nSamples):
49              samples[i] = buf[0]
50              avg = 0.995 * 0.5 * (buf[0] + buf[1])
51              buf.append(avg)
52              buf.popleft()
53          samples = np.array(samples * 32767, 'int16')
54          return samples.tobytes()
55
56
57      class NotePlayer:
58          def __init__(self):
59              pygame.mixer.pre_init(44100, -16, 1, 2048)
60              pygame.init()
61              self.notes = {}
62
63          def add(self, fileName):
64              self.notes[fileName] = pygame.mixer.Sound(fileName)
65
66          def playSong(self):
67              for freq in freq_list:
68                  if freq[0] < 0:
69                      freq[0] = 0
70                  note = self.notes[f'notes/{freq[0]}.wav']
71                  pygame.mixer.music.load(f'notes/{freq[0]}.wav')
72                  pygame.mixer.music.set_volume(freq[1]) # 设置当前音符音量
73                  pygame.mixer.music.play()
74                  time.sleep(0.25 * 1) # 1 or 2 or 4 or 8
```

```
            pygame.mixer.music.fadeout(5) # 淡出时间(ms)


nplayer = NotePlayer()
print('开始生成音符...')
for name, freq in list(freq_dict.items()):
    fileName = f'notes/{freq}.wav'
    if not os.path.exists(fileName):
        if freq != 0:
            data = generateNote(freq)
            print(f'成功创建音符{fileName}...')
            writeWAVE(fileName, data)
        elif freq == 0:
            sine(0, 1)
            print(f'成功创建音符{fileName}...')
    else:
        print(f'音符{freq}.wav已创,建忽略...')
    nplayer.add(f'notes/{freq}.wav')

nplayer.playSong()
```

## 4 创新性描述

本实验运用的模块如下:

```
argparse, collections, json, math, matplotlib, numpy, os, pygame, random, sys,
    time, wave
```

1. 一维波动方程作为力学课程中机械波的基础模型, 在采用纸质媒介的教材中难以给读者直观的认识, 在教学过程中教师多选择通过播放 flash 动画给学生进行展示, 但由于缺少参数选择等用户输入选项, 仍不能给学习者带来较好的实践体验. 本实验利用 Python 中 Matplotlib 模块提供的绘图功能, 将偏微分方程的有限差分法计算结果以 pyplot 中的三维立体及 FuncAnimation 的二维动态的模式呈现, 并向用户提供弦长、波节数、振动时间等参数选项, 有助于更好地理解并运用所学的数学、物理知识.

2. 在项目制作过程中, 我关注到不同音频 “悦耳” 程度的差异, 在与学院老师探讨后学习了这背后的数理原理. 在本实验中, 我通过实践对比验证音乐理论中有关泛音的结论, 同时利用 Python 提供的环形缓冲区进行随机拨弦模拟, 效果较为理想.

3. 本实验综合运用数学、物理、音乐乐理等多学科知识, 由浅入深逐步探索, 能极大激发对探索并运用所学知识的积极性. 同时从原理切入, 基于 Python 以视听结合的手段展现对物理过程的模拟结果, 效果清晰直观.

## 5 学习心得与收获

- **收获与感想**:

  1. 作为上学期刚体验过 C 语言学习的大一学生, 我深切地体会到 Python 各类库强大的功能, 从罗老师的讲解中学到了绘图、数据分析等实用功能. 在编程解决实际问题的过程中, Python 使我能专注于问题核心, 借助前人实现的丰富功能满足我的设计需求, 呈现出更完善的效果.
  2. 在本次大作业的选题阶段, 我曾想探索音乐节奏与音高识别的有关内容, 但在经历一个半月的 "摸爬滚打"、尝试若干深度学习模型之后仍不能实现较好的音频再现效果. 后来我在物理课上从老师展示的问题中得到了设计灵感, 最终转向与所学数学、物理知识结合更紧密的实验. 这个过程也让我意识到提出合适问题本身可能比解决问题更需要精准的判断.
  3. 横向而言, 通过综合运用偏微分方程、力学、数值计算、音乐乐理等多学科知识, 并借助功能强大的 Python 编程语言, 我较好地体会到多学科知识交融的乐趣, 也极大地锻炼了我提出并解决问题的能力.
  4. 纵向而言, 让刚入大学的我主动了解并实践了数学知识在解决实际问题中的应用. 同时也让我对偏微分方程、数值分析等未来将要学习的课程有了初步的认识和一定的兴趣.

- **对课程及讲义内容的建议**:

  1. 考虑到不少经典算法同学们在大一秋季的计算机程序设计课程中已用 C 语言实现并掌握, 在本门课程中可以适当减少对侧重基础算法的示例的细致讲解, 仅保留项目框架, 从而留出更多时间进行 Python 语法等知识的学习.
  2. 罗老师编写的讲义语言精炼、实例丰富且易于理解, 但有不少程序篇幅过长, 因此建议在展示代码时使用更小字号, 同时更多地使用代码前后注释的形式, 这样可以减少阅读篇幅较长的程序时前后翻页数过多的不便.
  3. 讲义中有部分涉及数学公式、符号变量的地方排版需要调整. 例如讲义中有部分表示变量的字母应从正体改为 LaTeX 中带 `$...$` 的代码, 以斜体字母的形式显示; 行内巨型数学算符 (如 $\sum_{i=1}^{n}$ 和 $\prod_{i=1}^{n}$ 等带上下标的运算) 应考虑使用

– `\displaystyle{}`, 或

– `\sum\limits_{}` 及 `\prod\limits_{}`

等命令呈现更好的行内公式效果. 此外, 部分程序示例中代码过长时会超出边界.

# 6 参考资料


[1] 罗奇鸣. Python 科学计算基础

[2] 杨维纮. 力学与理论力学 [M]. 2. 科学出版社, 2022.

[3] Venkitachalam M. Python Playground [M]. 1. William Pollock, 2016.

[4] M.Stein E. & Rami Shakarchi. Fourier Analysis [M]. 1. Princeton University Press, 2003.

[5] David Kincaid & Ward Cheney. Numerical Analysis [M]. 3. American Mathematical Society, 2002.

[6] https://matplotlib.org/ Matplotlib 模块文档