计算物理第9次作业

吴远清 2018300001031

第一题

题目分析

由于题目中的弦一端固定, 因此边界条件需要修改.其中固定端的高度恒设为0, 而自由端的高度设置为最邻近的线元的高度.

由于初始情况较为复杂且多变, 我通过在类的初始化时传入一个 `function` 对象来实现不同初始情况下的复用.

代码实现

首先引入相关包:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
import numpy.fft as nf
```

生成初始波的函数:

```
def sinFunctionGenerate(x):
    return np.sin(10*np.pi*x)

def gaussPack(x):
    x0 = 0.4
    k = 1000
    return np.exp(-k*np.power((x-x0),2))
```

其中, `sinFunctionGenerate` 用于生成正弦波, `gaussPack` 用于生成高斯波包. 二者均可直接作用于 `numpy array` 上.

类的实现:

```
class waveOnString():
    def __init__(self, c = 5, dt = 0.001, dx = 0.005, waveGenerator = 
sinFunctionGenerate, l = 1, t0 = 5, waveRange = [0,1/4]):
        self.x = np.arange(0,l,dx)
        self.y = np.zeros(int(l/dx))
        self.newy = self.y.copy()
        self.y[int(waveRange[0]/dx):int(waveRange[1]/dx)] = 
waveGenerator(self.x[int(waveRange[0]/dx):int(waveRange[1]/dx)])
        self.lasty = self.y.copy()
        self.r = c*dt/dx
        self.t0 = t0
        self.l = l
        self.dt = dt
        self.dx = dx
```

```python
        self.t = 0
    def __update(self):
        self.y[0] = self.y[1]
        self.y[-1] = 0
        self.newy[1:-1] = 2*(1-self.r**2)*self.y[1:-1] - self.lasty[1:-1] + self.r**2*(self.y[2:]+self.y[:-2])
        self.lasty = self.y.copy()
        self.y = self.newy.copy()
        self.t += self.dt
    def __animaInit(self):
        self.ln.set_data(self.x,self.y)
        return self.ln,
    def __animeUpdate(self,frame):
        self.ln.set_data(self.x,self.y)
        self.__update()
        return self.ln,
    def animate(self):
        fig,ax = plt.subplots()
        self.ln, = plt.plot(self.x, self.y)
        anim = animation.FuncAnimation(fig,self.__animeUpdate, frames=np.linspace(0,self.t0,int(self.t0/self.dt)), interval = 10, init_func = self.__animaInit, blit = True)
        plt.show()
    def freqSpec(self, x = None):
        if x == None:
            x = np.mean(self.x)
        index = int(x/self.dx)
        t = [0]
        a = [self.y[index]]
        for i in np.arange(0,self.t0,self.dt):
            self.__update()
            t.append(t[-1] + self.dt)
            a.append(self.y[index])
        t, a = np.array(t), np.array(a)
        plt.subplot(121)
        plt.plot(t,a)
        plt.title("Signal at $x = {}$".format(x))
        plt.subplot(122)
        comp_arr = nf.fft(a)
        freqs = nf.fftfreq(a.size, t[1] - t[0])
        pows = np.abs(comp_arr)
        plt.plot(freqs[freqs > 0], pows[freqs > 0])
        plt.title("Frequency")
        plt.show()
```

初始化中,我们传入了`dt`,`dx`,`l`,`c`,`t0`等振动参数,并传入了`waveGenerator`和`waveRange`用于控制初始波形和分布.

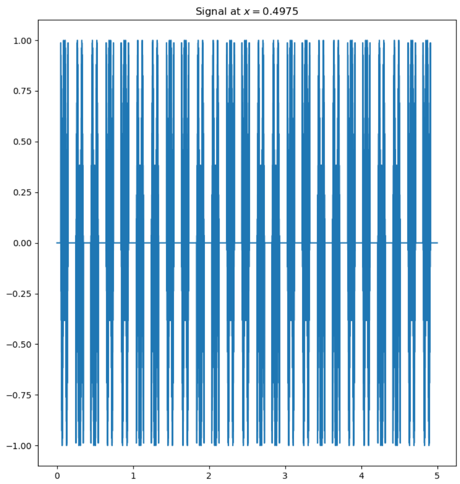
`__update`方法中,我实现了一个时间步的迭代,其中没有使用循环,而是直接使用`numpy`的数组索引, 可以大幅提升计算效率.

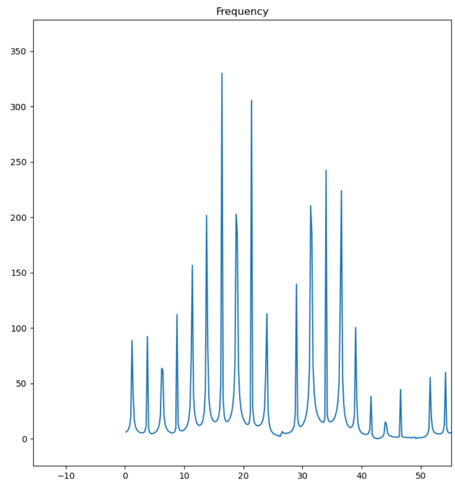
`__animaInit`和`__animeUpdate`是用于动画生成的内部方法.

`animate`可以实现弦振动的可视化,通过`matplotlib`的`funcAnimation`方法来实现绘制振动的动图.

`freqSpec`方法实现了记录一点在整个振动周期内的振幅变化,并作傅里叶变换以绘制频谱.

结果分析





可以看出,频率均是1.25Hz的倍频,即波长和弦长满足关系:

$$L = \frac{2n+1}{4}\lambda \qquad n \in \mathbb{N}^*$$

与两端均为固定的弦不同.

第二题

代码实现

产生方波的函数:

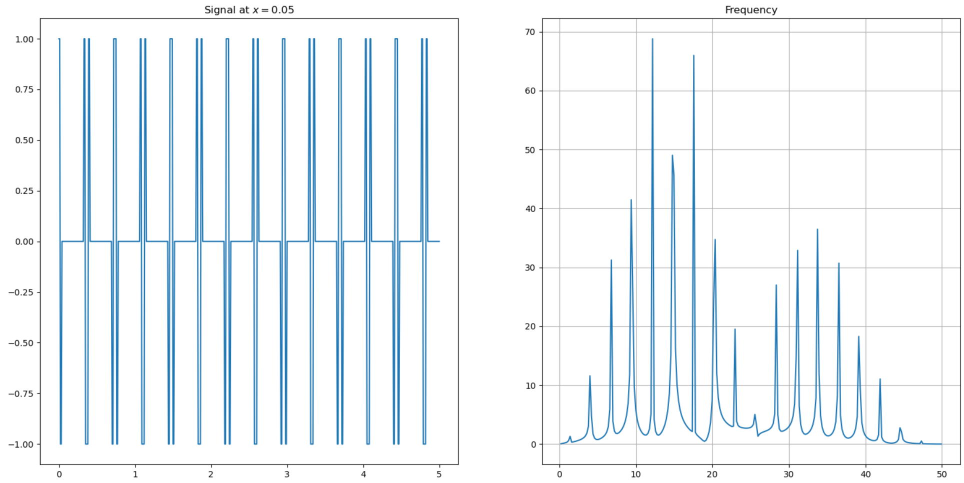
```
def squareWaveGenerate(x):
    ampl = 1.0
    wide = 0.1
    x0 = x[0]
    flag = 1
    y = np.zeros(x.__len__())
    for i in range(x.__len__()):
        if x[i] - x0 > wide:
            print('ddd')
            x0 = x[i]
            flag = flag*-1
        y[i] = flag*ampl
    return y
```

设置初始波形为方波,并设置观察位置为距离固定端5%的位置(此情况下为0.95):

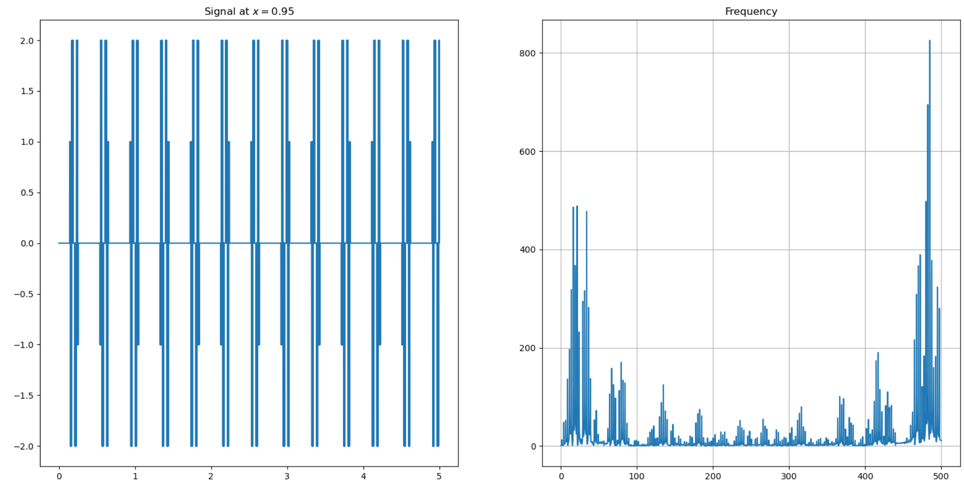
```
w = waveOnString(dt = 0.001, dx = 0.005, waveGenerator=squareWaveGenerate)
w.freqSpec(x = 0.95)
```

结果分析

当采样频率较低时,分析所得的频谱范围也较低(采样时间$dt = 0.01$,采样频率$f = 100Hz$):



而当我们提升采样频率时(降低采样间隔), 分析所得的频谱也更加广泛(采样时间$dt = 0.001$,采样频率$f = 1000Hz$)):



此结果验证了Nyquist采样定理.