Assignment 3 by Xin Sheng and Aether Zhou
This file is assignment_3_Xin_and_Aether.pdf
Python filename is assignment_3_xin_and_aether.py

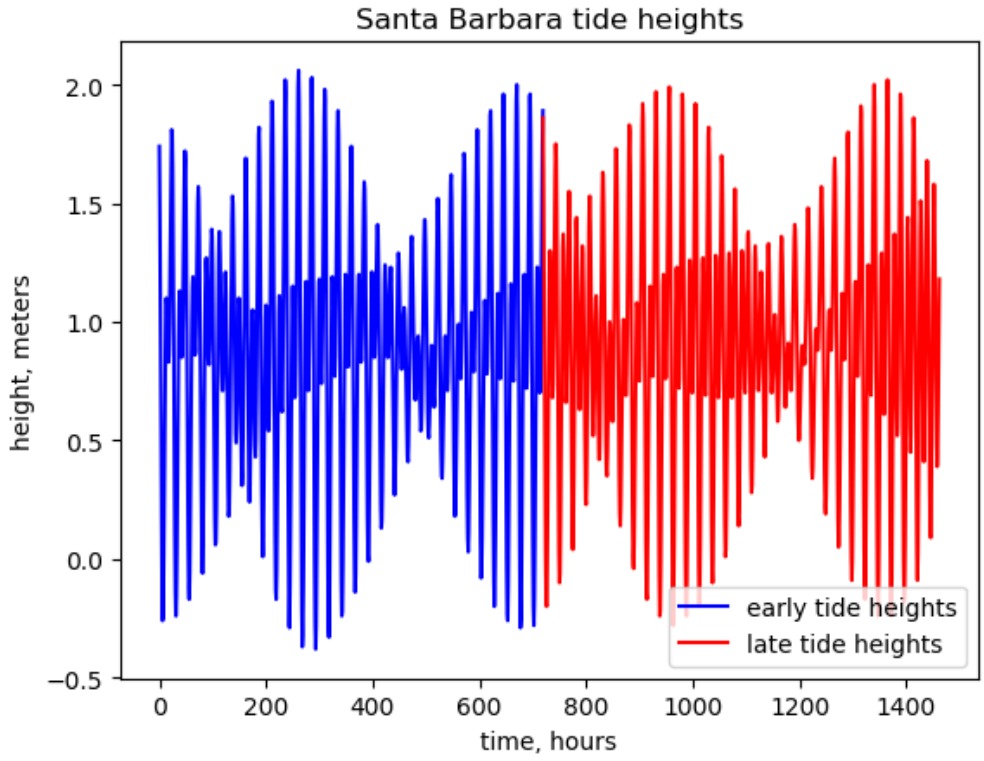
Group name: Phantom Thieves of Hearts

1) Questions
   - Why tides often have an approximate period of about 12.5 hours
     Since the moon revolves around earth while earth is spinning, a lunar day is about 24 hours and 50 minutes. Hence, the time interval between two high tides (or low tides) is about 12.5 hours (Sumich, J.L., 1996; Thurman, H.V., 1994).

   - Why tides sometimes have an approximate period of about 25 hours?
     When the moon-earth-sun angle is 90 degrees, the effect of sun's gravity will give an acceleration that is perpendicular to moon-earth direction on the ocean, and reduce the tide height significantly. Thus, the reduced tide may not qualify a tide and result in a 25 hours interval between the adjacent tides.

   - Why tides change throughout the year?
     Since the earth is revolving around the sun, season and ocean current will change throughout the year. And so does the tides.

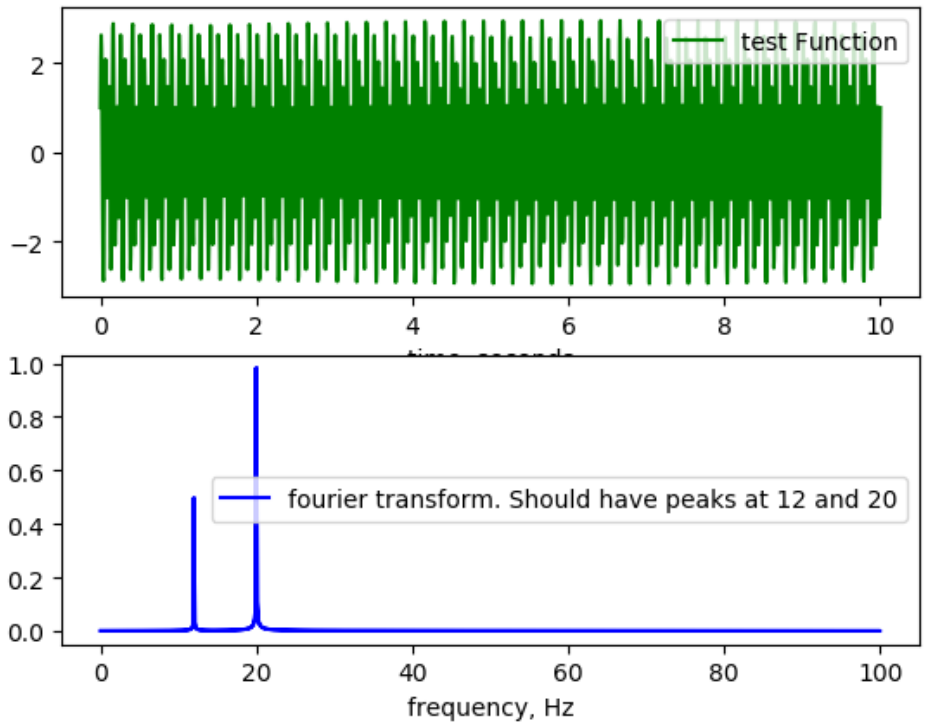
2) copy explore_tide_table.py, tidetable.txt, and tidetable2.txt into a directory that is shared by you and your partner. Run explore_tide_table.py, which will call plotTwoTides().



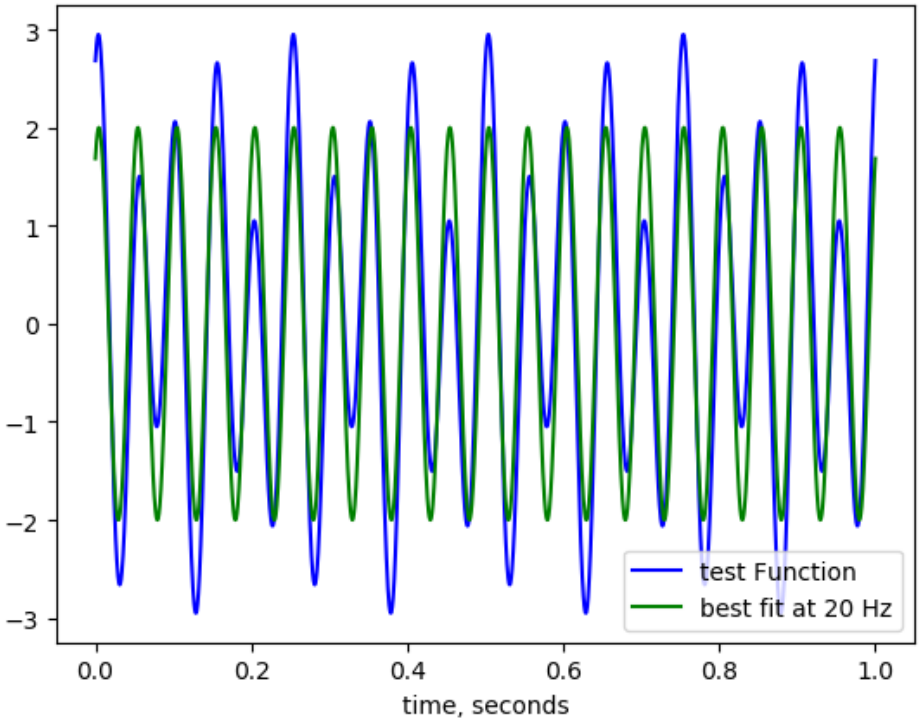
3) While I like this format, I would like you to delete this line and replace it with a for loop that does the same thing (likely in more lines).

```
8
9 def loadTides(filename = 'tidetable.txt'):
10     tideEntries = csv.reader(open(filename),delimiter='\t')
11     for i in range(14):     #what do the next two lines do? where does that 14 come from?
12         next(tideEntries)  #tideEntries is an iterable object. What does this mean?
13     tideHeights=[]
14     for row in tideEntries:
15         tideHeights.append(np.float(row[3]))
16     return tideHeights
```

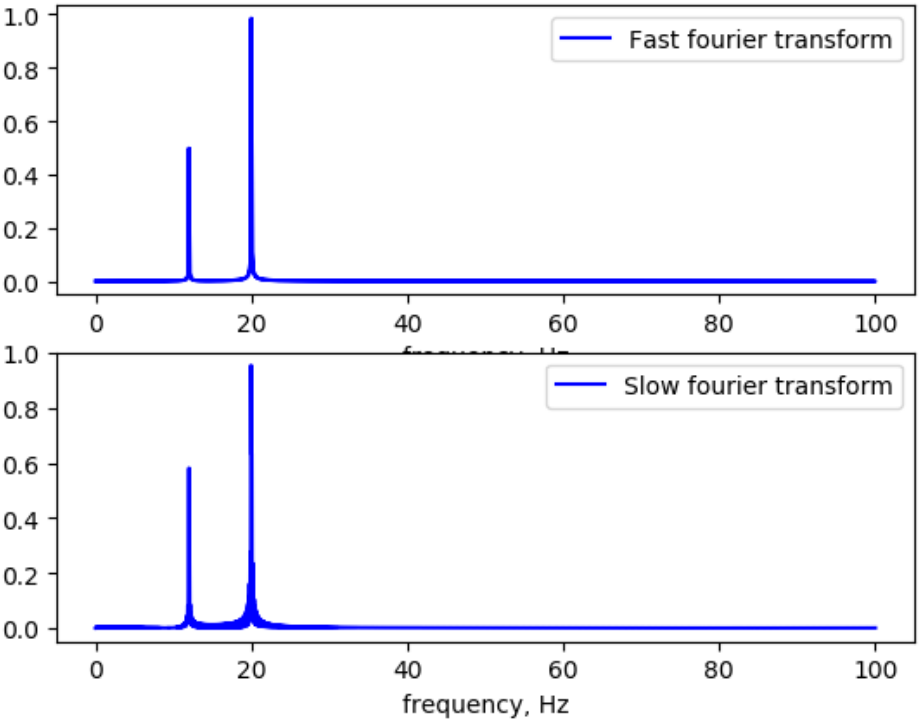
4) Write a function absFFT(times,amplitude) which takes a series of times and amplitudes and returns the corresponding frequencies and power spectrum. It will very likely call np.fft.fft() Open testFourier.py from Gauchospace and paste it into your file.



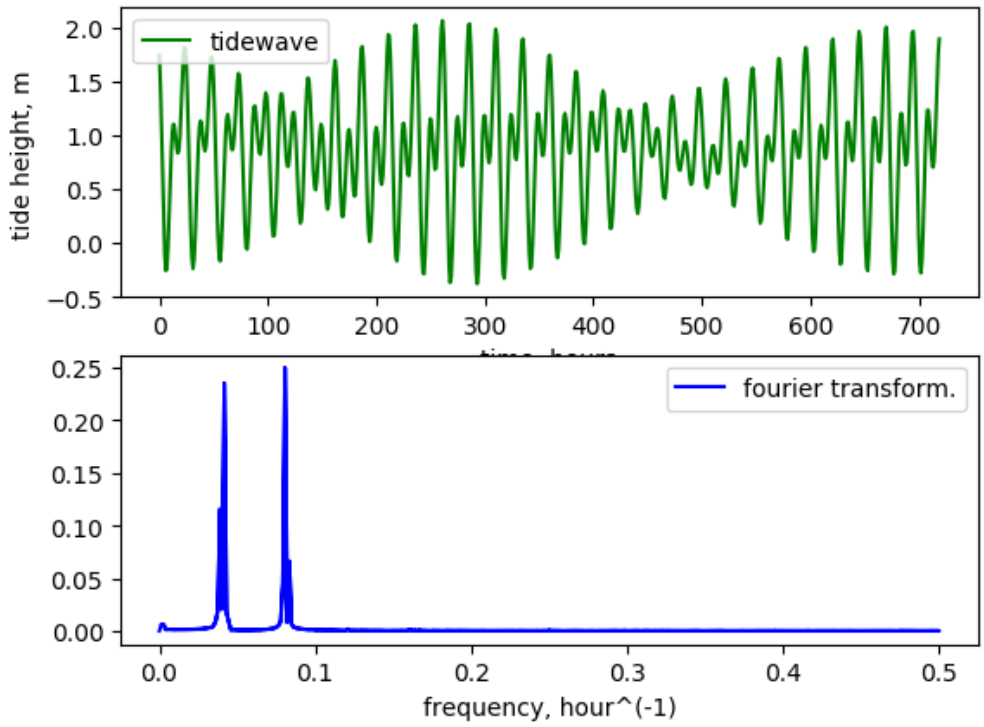
5) Write a function sinAmplitude,cosAmplitude = getFourierComponent(times,data,f) which returns a single Fourier component.



6) Use your function getFourierComponent() to write a function absSlowFT(). Make a plot which compares the output of absSlowFT() and absFFT().

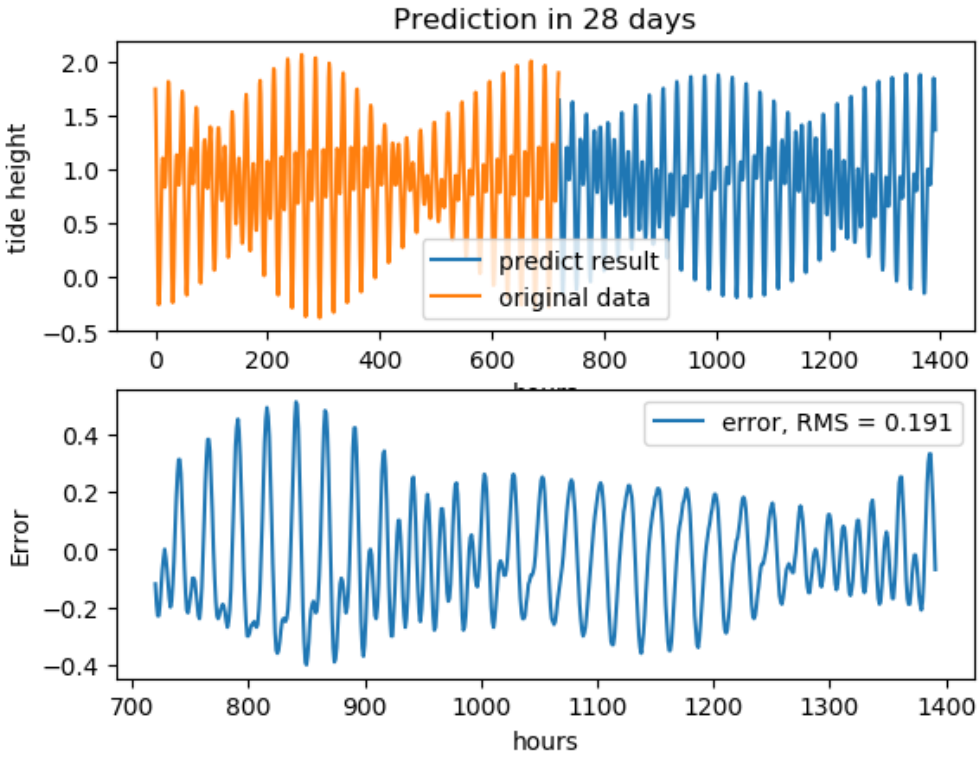


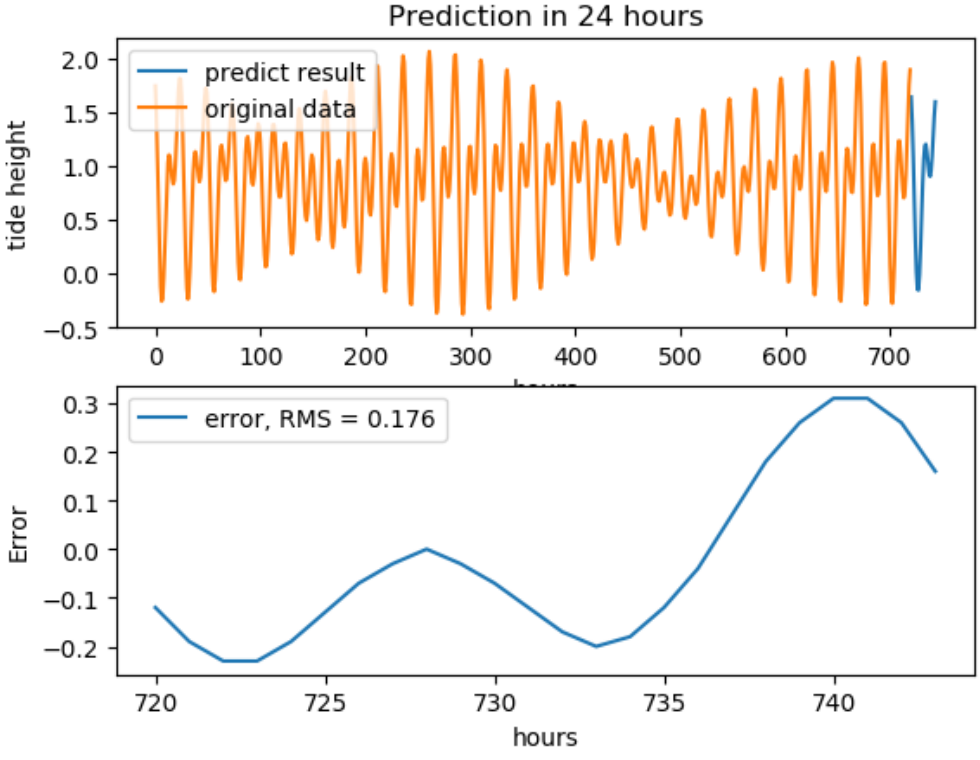
7) Write a function plotTidesFourier() that plots the fourier transform of the tide data. Label the frequency axis is some correct but reasonable units. Explain in words what this plot is telling you



The plot tells us that there are roughly two frequencies that dominant the tide wave. They are about 0.0806 and 0.0417 Hour^(-1).

8) For both 1 day and 28 days make a figure which contains two plots: one showing the plot from step 2 above, along with your prediction; and a second one showing just your errors.





Code questions answers:

```
#what do the next two lines do? where does that 14 come from?
#this two lines iterate the tidetable.txt file and move the index to line 14
#tide data starts from line 14 in the file tidetable.txt

#tideEntries is an iterable object. What does this mean?
#an iterable object means it can be iterated by indices

#QUESTION: What does that * mean?
#* allows we put multiple positional argument into a tuple of arguments
```

Full code:

```
import matplotlib.pyplot as plt
import numpy as np
import csv
from scipy.optimize import curve_fit

#load tide data from file 'tidetable.txt'
def loadTides(filename = 'tidetable.txt'):
                                        tideEntries         =
csv.reader(open(filename),delimiter='\t')
    for i in range(14):
        next(tideEntries)
    tideHeights=[]
    for row in tideEntries:
        tideHeights.append(np.float(row[3]))
    return tideHeights

#plot loaded tide data
def plotTides():
    tideHeights = loadTides()
    timeInHours = np.arange(len(tideHeights))
    plt.figure()
      plt.plot(timeInHours,tideHeights, 'b-', label='tide
Height')
    plt.legend()
    plt.show()

#plot tide data for the current month and the next month
def plotTwoTides():
    earlyTideHeights = loadTides('tidetable.txt')
    earlyTimeInHours = np.arange(len(earlyTideHeights))
    lateTideHeights = loadTides('tidetable2.txt')
      lateTimeInHours = np.arange(len(lateTideHeights)) +
earlyTimeInHours.max()
    plt.figure()
       plt.plot(earlyTimeInHours,earlyTideHeights, 'b-',
label='early tide heights')
         plt.plot(lateTimeInHours,lateTideHeights, 'r-',
label='late tide heights')
    plt.xlabel('time, hours')
    plt.ylabel('height, meters')
    plt.title('Santa Barbara tide heights')
    plt.legend()
    plt.show()

#the function for curveFit() demo
def func(x, a, b, c):
    return a * np.exp(-b * x) + c

#a demo of curve fitting
```

```
def curveFit():
    xdata = np.linspace(0, 4, 50)
    y = func(xdata, 2.5, 1.3, 0.5)
    np.random.seed(1729)
     #look at this line, and compare it to what you just
did. This is written rather pythonically
    y_noise = 0.2 * np.random.normal(size=xdata.size)
    ydata = y + y_noise
    plt.figure()
    plt.plot(xdata, ydata, 'b-', label='data')

    popt, pcov = curve_fit(func, xdata, ydata)
    print(popt)

    plt.plot(xdata, func(xdata, *popt), 'r-',
                 label='fit: a=%5.3f, b=%5.3f, c=%5.3f' %
tuple(popt))

     popt, pcov = curve_fit(func, xdata, ydata, bounds=(0,
[3., 1., 0.5]))

    plt.plot(xdata, func(xdata, *popt), 'g--',
                 label='fit: a=%5.3f, b=%5.3f, c=%5.3f' %
tuple(popt))

    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend()
    plt.show()

#returning the possible freqencies and amplitude for each
frequencies for a FFT
def absFFT(times, amplitude):
    result = np.abs(np.fft.fft(amplitude))/len(times)
                                              freq          =
len(times)/times[-1]*np.abs(np.fft.fftfreq(len(times)))
    return  freq, result

#similar to absFFT, but applying definition for fourier
transformation rather than using numPy.fft.fft()
def absSlowFT(times, amplitude):
    freq = np.linspace(0, 100, len(times))
    result = []
    for i in freq:
             sinAmp, cosAmp = getFourierComponent(times,
amplitude, i)
        result.append(np.abs(sinAmp/2 + cosAmp/2))
    return freq, result

#plotting the example function and its frourier
transformation
```

```
def testFourier():
    times = np.linspace(0,10,2000)
    f1 = 12  #in Hertz
    f2 = 20  #in Hertz
      testFunction = np.cos(times * f1 * 2 * np.pi) +
(2*np.sin(times * f2 * 2 * np.pi))
                          frequencies,powerSpectrum      =
absFFT(times,testFunction)

    plt.figure()
    plt.subplot(211)
         plt.plot(times,testFunction,  'g-',  label='test
Function')
    plt.xlabel('time, seconds')
    plt.legend()

    plt.subplot(212)
                       plt.plot(frequencies,powerSpectrum,
'b-',label='fourier  transform.  Should  have  peaks  at  12
and 20')
    plt.xlabel('frequency, Hz')
    plt.legend()

#plotting the results of absFFT(times, testFunction) and
absSlowFT(times, testFunction)
def compareByTestFourier():
    times = np.linspace(0,10,2000)
    f1 = 12  #in Hertz
    f2 = 20  #in Hertz
      testFunction = np.cos(times * f1 * 2 * np.pi) +
(2*np.sin(times * f2 * 2 * np.pi))
                          frequencies,powerSpectrum      =
absFFT(times,testFunction)
                     frequenciesnew,powerSpectrumnew     =
absSlowFT(times,testFunction)

    plt.figure()
    plt.subplot(211)
     plt.plot(frequencies,powerSpectrum, 'b-',label='Fast
fourier transform')
    plt.xlabel('frequency, Hz')
    plt.legend()

    plt.subplot(212)
                plt.plot(frequenciesnew,powerSpectrumnew,
'b-',label='Slow fourier transform')
    plt.xlabel('frequency, Hz')
    plt.legend()

#fourier  transformation  to  get  the  real  and  imaginary
parts
```

```
def getFourierComponent(times,data,f):
    sinwave = 2 * data * np.sin(2 * np.pi * f * times)  /
len(times)
    coswave = 2 * data * np.cos(2* np.pi * f *  times)  /
len(times)
    sinAmp = np.sum(sinwave)
    cosAmp = np.sum(coswave)
    return sinAmp, cosAmp

#plot the original data and the FT fitted curve
def overlayFourier():
    times = np.linspace(0,1,2000)
    f1 = 12  #in Hertz
    f2 = 20  #in Hertz
      testFunction = np.cos(times * f1 * 2 * np.pi) +
(2*np.sin((times * f2 * 2 * np.pi) + 1))
                          frequencies,powerSpectrum      =
absFFT(times,testFunction)

    f = 20
                                      sinAmp,cosAmp        =
getFourierComponent(times,testFunction,f)
     fitData = sinAmp * np.sin(f * 2 * np.pi * times) +
cosAmp * np.cos(f * 2 * np.pi * times)

    plt.figure()
         plt.plot(times,testFunction,  'b-',  label='test
Function')
     plt.plot(times,fitData, 'g-', label='best fit at %d
Hz'%f)
    plt.xlabel('time, seconds')
    plt.legend()

#plotting loaded tide data and the fourier transform
def plotTidesFourier():
    data = loadTides()
    timeInHours = np.arange(len(data))
       frequencies,  powerSpectrum  =  absFFT(timeInHours,
data)
    powerSpectrum[0] = 0

    plt.figure()
    plt.subplot(211)
    plt.plot(timeInHours, data, 'g-', label='tidewave')

    plt.xlabel('time, seconds')
    plt.legend()

    plt.subplot(212)
                  plt.plot(frequencies,    powerSpectrum,
'b-',label='fourier transform.')
```

```
    plt.xlabel('frequency, Hz')
    plt.legend()

    return 0

#pick appropriate frequencies for tide fitting
def findFrequency():
    significance = []
    data = loadTides()
    timeInHours = np.arange(len(data))
       frequencies, powerSpectrum = absFFT(timeInHours,
data)
    for i in range (1,len(powerSpectrum)):
        if powerSpectrum[i] > 0.1:
            significance.append(frequencies[i])
    return significance

#the math model for tide fitting
def fitFunction(times,A1,A2,A3,A4,A5,A6,const,phase):
      function = const + A1* np.sin(0.08067 * 2*np.pi*
times)   +A2*np.cos(0.08067   *2*np.pi*   times)   +A3   *
np.sin(0.04172* 2*np.pi * times) +A4 * np.cos(0.04172
*2*np.pi* times) +A5 * np.sin(0.03894* 2*np.pi * times)
+A6 * np.cos(0.03894 *2*np.pi* times)
    return function

#plotting prediction for next month and error
def           predictNextMonthsTides(filename           =
'tidetable.txt',filename2 = 'tidetable2.txt'):
    hours = 24 * 28
    tideHeights = loadTides(filename)
    tideHeights2 = loadTides(filename2)
    times=[]
    dayCounter = 0
    for i in range(0,len(tideHeights)):
        times.append(np.float(dayCounter))
        dayCounter = dayCounter + 1
    times = np.asarray(times)
    predictTimes = []
    for i in range (0, hours):
        predictTimes.append(times[i]+times[-1]+1)
          popt,  pcov  =  curve_fit(fitFunction,  times,
tideHeights)

    predictTimes = np.asarray(predictTimes)
    fitResult = fitFunction(predictTimes, *popt)
    error = []
    totalError = 0
    for i in range (0,hours):
        error.append(tideHeights[i]-tideHeights2[i])
                                        totalError     +=
```

```
np.abs(tideHeights[i]-tideHeights2[i])**2/hours
    RMSE = np.sqrt(totalError)

    plt.figure()
    plt.subplot(211)
         plt.plot(predictTimes,  fitResult,label='predict
result')
    plt.plot(times, tideHeights,label='original data')
    plt.title('Prediction in 28 days')
    plt.xlabel('hours')
    plt.ylabel('tide height')
    plt.legend()

    plt.subplot(212)
      plt.plot(predictTimes, error , label='error, RMS =
%.3f' % RMSE)
    plt.xlabel('hours')
    plt.ylabel('Error')
    plt.legend()
    plt.show()
    return 0

#plotting prediction for next day and error
def            predictNextDaysTides(filename            =
'tidetable.txt',filename2 = 'tidetable2.txt'):
    hours = 24
    tideHeights = loadTides(filename)
    tideHeights2 = loadTides(filename2)
    times=[]
    dayCounter = 0
    for i in range(0,len(tideHeights)):
        times.append(np.float(dayCounter))
        dayCounter = dayCounter + 1
    times = np.asarray(times)
    predictTimes = []
    for i in range (0, hours):
        predictTimes.append(times[i]+times[-1]+1)
          popt,  pcov  =  curve_fit(fitFunction,  times,
tideHeights)
    predictTimes = np.asarray(predictTimes)
    fitResult = fitFunction(predictTimes, *popt)
    error = []
    totalError = 0
    for i in range (0,hours):
        error.append(tideHeights[i]-tideHeights2[i])
                                          totalError     +=
np.abs(tideHeights[i]-tideHeights2[i])**2/hours
    RMSE = np.sqrt(totalError)

    plt.figure()
    plt.subplot(211)
```

```
          plt.plot(predictTimes,  fitResult,label='predict
result')
    plt.plot(times, tideHeights,label='original data')
    plt.title('Prediction in 24 hours')
    plt.xlabel('hours')
    plt.ylabel('tide height')
    plt.legend()

    plt.subplot(212)
      plt.plot(predictTimes, error , label='error, RMS =
%.3f' % RMSE)
    plt.xlabel('hours')
    plt.ylabel('Error')
    plt.legend()
    plt.show()
    return 0

print(findFrequency())
compareByTestFourier()
overlayFourier()
testFourier()
plotTidesFourier()
plotTwoTides()
predictNextMonthsTides()
predictNextDaysTides()
```