Lab Questions: Lab Session 12

Deadline: 15.11.2017 11:59pm SGT

Complete all assignments below. For the question that is marked with an asterisk *, i.e. **Questions 3** and **10**, create the files as requested. Once you are done with it, submit the file(s) via `iNTU`. Remember to put plenty of comments for all assignments, as this helps us to better understand your program (which might give you higher marks).

Important!!! Make sure your scripts work properly, as we give 0 marks otherwise. Please name the files according to the requirements, and upload each file separately and not in a Zip file or similar. Check that your files properly have been uploaded to `iNTU`. The submission system closes at the deadline. Hence after that, you will get no marks for your solution.

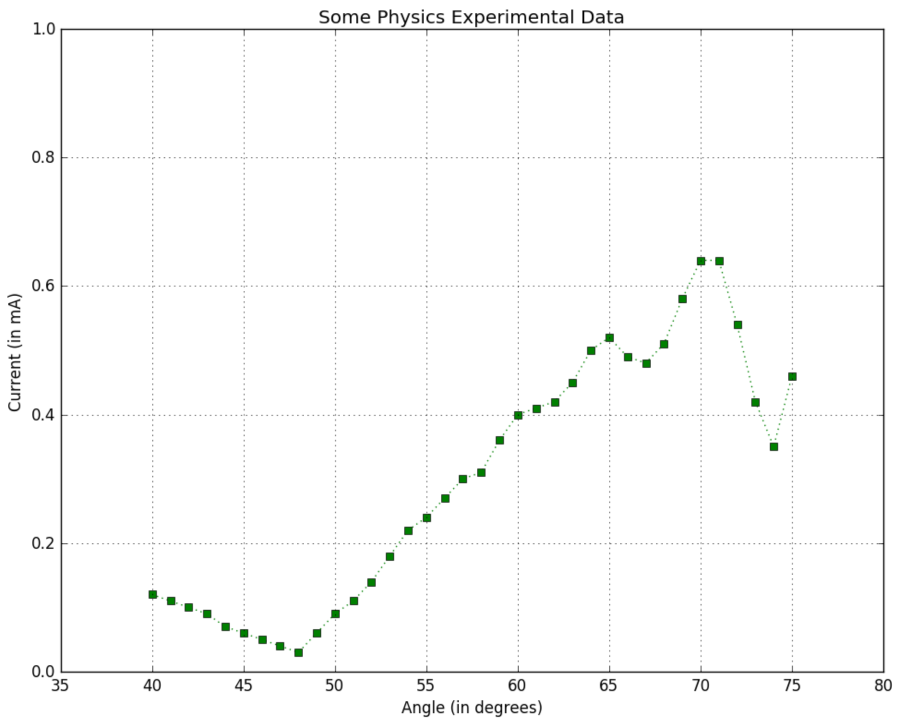
1. Write a Python script `PlotBar.py` to create a NumPy array `x` with values ranging from 7 to 77 in steps of $\sqrt{2}$. Create a NumPy array `y` which is the square root of each value in `x`. Plot these points using `plot` and `bar` (in separate figures).

   Solution:

   PlotBar.py

```python
import numpy as np
import matplotlib.pyplot as plt

# Create the vectors x,y
x = np.arange(7,77,np.sqrt(2))
y = np.sqrt(x)

# Display the plot in the first figure
plt.figure(1)
plt.plot(x,y)

# Display the bar in the second figure
plt.figure(2)
plt.bar(x,y)

plt.show()
```

2. An experiment in Physics measures the current that is flowing in dependency of the angle of a receiver (no more details available). The results have been stored in the file `ExperimentalData.txt`, which you can download from `NTU learn`. The first column represents the angle (in degrees) and the second column the current (in mA). Write a script `PlotData.py` that will load this data and plot it. Make sure your plot looks exactly like the plot below (it is green color) including range of the axes, title and axis labels.



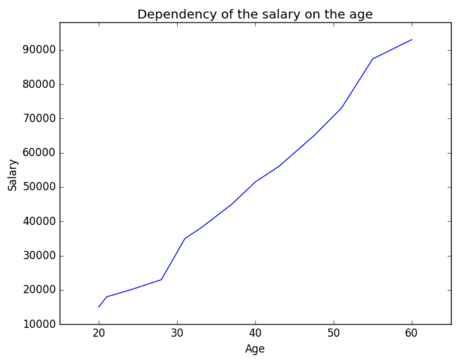
Solution:

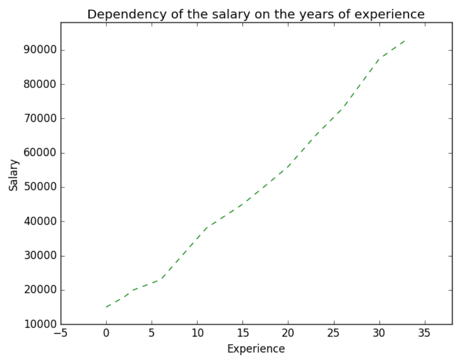
PlotData.py

```
# This scripts reads from the file ExperimentalData.txt
# and plots its data

import numpy as np
import matplotlib.pyplot as plt

# read from file
my_data = np.loadtxt('ExperimentalData.txt')

# split the matrix into two vectors
ang = my_data[:,0]
current = my_data[:,1]

# plot the vectors in green, with circles, dash-dotted line
plt.plot(ang,current,'gs:')

# turn the grid on
plt.grid()

# label axes and put title
plt.axis([35, 80, 0, 1])
plt.xlabel('Angle (in degrees)')
plt.ylabel('Current (in mA)')
plt.title('Some Physics Experimental Data')

plt.show()
```

3. * Plotting data stored in a file is trivial with Python. In one such file, called `salaries.dat` (can be downloaded from `iNTU learn`), are stored the salaries of working individuals along with their age and years of experience. The age of the person is in the first row, the experience in the second, and the salary in the third row. Write a Python script `plotsalaries.py` that will read the data from the file and will plot the dependency of the salary on the age as well as the dependency of the salary on the experience. More precisely, your script will do the following:

   - Read the file `salaries.dat` and store the data into a NumPy matrix.
   - Output the number of entries, i.e. the number of columns of the matrix, in the form:
     ```
     >>> runfile('plotsalaries.py')
     There are 13 salaries in the file.
     ```
   - Plot in two **different** figures (in two different windows!) as shown below, the two dependencies – the first with solid blue line, the second with dashed green line. In both figures, make sure you properly choose the axes, label them and give a title to the plots.





   Submit only your script file to NTU learn.
   Important!!! It should not be assumed that the number of entries in the file is known (there are three rows, but the number of entries in each of the rows is not known in advance).

4. Vectorize this code! Write one assignment statement that will accomplish exactly the same thing as the given code (assume that the variable `vec` has been initialized):

   ```python
   result = 0
   for i in range(len(vec)):
       result = result + vec[i]
   ```

   Solution:
   ```
   >>> result = np.sum(vec)
   ```

5. Create a random 3 x 5 NumPy matrix, each element being a integer in the range from -10 to 10 included. Perform each of the following using only vectorized code:

   Solution:
   ```
   >>> mat = np.random.randint(-10,11,[3,5])
   ```

(a) subtract 3 from each element

Solution:
```
>>> mat-3
```

(b) count how many are positive

Solution:
```
>>> np.sum(mat > 0)
```

(c) get the absolute value of each element

Solution:
```
>>> np.abs(mat)
```

(d) find the maximum

Solution:
```
>>> np.max(mat)
```

6. The file **measureddata.dat**, which can be downloaded from iNTU, contains the heights (in cm) of some people. Write a Python script `checkHeights.py`, that loads the measurements, finds the maximal and minimal heights, then asks the user to enter a range of height within the maximal and minimal heights and outputs how many people are in this range.
Make sure you do error checking on the user input, i.e. the range entered by the user has to be inside the found minimal and maximal heights. **Use only vectorized code (without any loops)**.
Example:

```
>>> runfile("checkHeights.py")
Please enter the range heights [low,high] within [155,190]
low : 160
high: 170
The total number of people in the range [160,170] is 3028

>>> runfile("checkHeights.py")
Please enter the range heights [low,high] within [155,190]
low : 165
high: 164
You've entered incorrect range: low > height. Exiting...

>>> runfile("checkHeights.py")
Please enter the range heights [low,high] within [155,190]
low : 160
high: 201
You've entered incorrect range: high > maximal height. Exiting...
```

Solution:

checkHeights.py

```
# Finds the number of people that have height in a certain range
import numpy as np

# Load the measurements
my_data = np.loadtxt('measureddata.dat')

# Find the minimal and maximal heights
min_height = np.min(my_data)
max_height = np.max(my_data)

# Ask the user to enter heights
print('Please enter the range heights [low,high] within [%3d,%3d]' % (
    min_height,max_height))
low  = float(input('low : '))
high = float(input('high: '))

# Error check
if low > high:
    print('You''ve entered incorrect range: low > height. Exiting...')
elif low < min_height:
    print('You''ve entered incorrect range: low < minimal height.
        Exiting...')
elif high > max_height:
    print('You''ve entered incorrect range: high > maximal height.
        Exiting...')
# No error in the input
else:
    total_people = np.sum(np.logical_and(my_data >= low, my_data <=
        high))
    print('The total number of people in the range [%3d,%3d] is %d' %
        (low,high, total_people))
```

7. A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 10 separate times; they are stored in NumPy arrays `r` and `h`.
`r = np.array([5.501, 5.5, 5.499, 5.498, 5.5, 5.5, 5.52, 5.51, 5.5, 5.48])`
`h = np.array([11.11, 11.1, 11.1, 11.12, 11.09, 11.11, 11.11, 11.1, 11.08, 11.11])`
In a Python script `calibrating.py`, use vectorized code — i.e. do not use **any** loop statement — to find the volume from each trial, which is given by the formula $vol = \Pi r^2 h$. Also use logical indexing first to make sure that all measurements were valid (i.e. all measurement values are > 0).

Solution:

```
import numpy as np

r = np.array([5.501, 5.5, 5.499, 5.498, 5.5, 5.5, 5.52, 5.51, 5.5,
    5.48])
h = np.array([11.11, 11.1, 11.1, 11.12, 11.09, 11.11, 11.11, 11.1,
    11.08, 11.11])

if not (all(r>0) & all(h>0)):
    print('error in the measurements')
else:
    vol = np.pi * r**2 * h
```

8. Write a function **harmonicSer** which will receive values of a, b and n, and will calculate and return the sum of the general harmonic series:

$$\frac{1}{b} + \frac{1}{a+b} + \frac{1}{2a+b} + \frac{1}{3a+b} + \cdots + \frac{1}{n \cdot a + b}$$

The code must be vectorized, so no loop should be used. The following examples of calls to this function illustrate what the result should be:

```
>>> print(harmonicSer(1,1,5))
   2.45
>>> print(harmonicSer(2,3,4))
   0.878210678211
```

Solution:

harmonicSer.py

```
import numpy as np

def harmonicSer(a, b, n):
# calculates the sum of the general harmonic series
# 1/b + 1/(a+b) + 1/(2a+b) + ... + 1/(n.a+b)
# Format of call: harmonicSer(a, b, n)
# Returns sum of the series
    v = np.arange(n+1)
    terms = 1/(a * v + b)
    return sum(terms)

print(harmonicSer(2,3,4))
```

9. The numbers of Fibonacci sequence $\{F_n\}$ satisfy the relation $F_{n+1} = F_n + F_{n-1}$, i.e. each consecutive number in the sequence is a sum of the previous two numbers. Usually, $F_0 = F_1 = 1$, and thus the sequence has the form:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, \ldots$$

We can define a modified Fibonacci sequence, where the first two numbers are arbitrary values, i.e. for $F_0 = 11, F_1 = 20$, the sequence will be

$$11, 20, 31, 51, 82, 133, \ldots$$

Write a function **isFibonacci** that checks if a vector contains a modified Fibonacci sequence. That is, the input to the function is a vector (implemented as a NumPy array), and the function outputs **True** if the vector is a part of some modified Fibonacci sequence, and **False** otherwise. **The code should be vectorized and no loops are allowed**.
Example:

```
>>> print(isFibonacci(np.array([ 1, 1, 2, 3, 5 ])))
   True
>>> print(isFibonacci(np.array([ 20, 31, 51, 82, 133 ])))
   True
>>> print(isFibonacci(np.array([ 7, 9, 16, 25, 41 ])))
   True
>>> print(isFibonacci(np.array([ 2, 2, 4, 7 ])))
   False
```

Solution:

isFibonacci.py

```python
import numpy as np

def isFibonacci(vec):
# Checks if the input vector forms Fibonacci sequence
# Fibonacci sequences satisfies F_{n+1} = F_{n} + F_{n-1}
# which is equvalent to F_{n+1} - F_{n} = F_{n-1}

    #Compute the vector of differences F_{n+1} - F_{n}
    #We could have written as well d_vec = vec[1:] - vec[:-1]
    d_vec = np.diff(vec)

    # Check if the difference vector equals to F_{n}
    # For n = 0,the value of F_{0-1} = F_{-1} is undefined,
    # and that is why we do not check the very first value of d_vec
    # Similarly, we should not check the last two numbers
    return all(d_vec[1:] == vec[0:-2])
```

10. * Download the file **salaries_data.txt** from iNTU Learn, which contains the salaries of the employees of a company:

    - the first column will hold the ID of the employee (an integer value)
    - the second column will hold the salary per hour of the employee in the previous year
    - the third column will hold the number of hours worked for this employee in the previous year
    - the fourth column will hold the salary per hour of the employee in the current year
    - the fifth and last column will hold the number of hours worked for this employee in the current year

    Write a script `salary_check.py` that loads the salary data, computes the total salary increase of all employees from previous to current year, and lists the IDs of the employee that got a total salary increase greater than 5% from previous to current year. **The code should be vectorized and no loops are allowed.** Example:

```
>>> runfile("salary_check.py")
The average total salary increase is 5.3918%
The following employee IDs got a total salary increase greater than 5%:
[ 670.  256.   43.]
```

Note that there can be a slight variation on the average total salary increase percentage value given in the example above, depending on what average is computed (sum over all the percentages, or sum over all salaries): both will be considered a valid solution. Submit `salary_check.py` to iNTU.