SUMMER FELLOWSHIP REPORT

ON

MATHEMATICS USING PYTHON

Submitted by

ABITHA V

Under the Guidance of

Dr. ARVIND AJOY,

Assistant Professor,

Indian Institute of Technology, Palakkad.

August 9,2020

ACKNOWLEDGEMENT

I would like to thank the FOSSEE project from IIT Bombay for giving me an opportunity to do internship with Python. The internship opportunity was a great chance for me to learn and develop myself professionally. It helped me to enhance my knowledge in NumPy. I feel grateful to have met so many wonderful people and professionals who guided me through this internship period.

I would like to specially acknowledge Dr. Arvind Ajoy with my deepest gratitude who in spite of being busy with his duties, took time out to hear, guide and keep me on the correct path.

I consider this opportunity as a big milestone in my career development. I shall strive to use the acquired skills and knowledge in the best possible way, and I will continue to work on its improvement, in order to attain desired career objectives.

CONTENTS

CHAPTER 1

INTRODUCTION

Mathematics using python fellowship is provided under the FOSSEE project. FOSSEE project promotes the use of FOSS (Free and Open Source Software) tools to improve quality of education in our country. FOSSEE promotes the use of FOSS tools through various activities to ensure commercial(paid) softwares are replaced by equivalent FOSS tools.

The FOSSEE project is a part of the National Mission on Education through Infrastructure and Communication Technology (ICT), Ministry of Human Resources and Development (MHRD), Government of India.

1.1 NUMPY IN PYTHON

Python is an interpreted, high-level, general purpose programming language. It was created by Guido van Rossum and released in 1991.Python's design emphasizes code readability with its use of significant whitespace. Python is dynamically typed and garbage collected. It supports object-oriented and functional programming. Python's large standard library considered as one of its greatest strengths provides tools suited for many tasks. The combination of python with fast computation has attracted scientists and others in large numbers. NumPy and SciPy are two powerful python packages that enable the language to be used efficiently for scientific purposes.

1.2 WHY NUMPY?

The basic operations used in scientific programming include arrays, matrices, integration, differential equation solvers, statistics and much more. NumPy is the fundamental Python package for scientific computing. It adds the capabilities of N-dimensional arrays, element-by-element operations, core mathematical operations like linear algebra, and the ability to wrap C/C++/Fortran code.

Python stores data in several different ways, but the most popular methods are lists and dictionaries. But operating the elements in a list can only be done through iterative loops, which is computationally inefficient in python. The NumPy package enables users to overcome the shortcomings of the Python lists by providing a data storage object called *ndarray*.

CHAPTER 2

RANDOM WALK

2.1 GENERATING AND PLOTTING TRAJECTORIES

A Random walk is a mathematical object, known as stochastic or random process, that describes a python that consists of a succession of random steps on some mathematical spaces such as the integers.

An elementary example of a random walk is the random walk on the integer number line, which starts at 0 and at each step moves +1 or -1 with equal probability.

Other examples are the path traced by a molecule as it travels in a liquid or a gas, the price of a fluctuating stock and the financial status of a gambler: all can be approximated by random walk models, even though they may not be truly random in reality.

Adding the following code:

```
import numpy
#import pylab
import random
import matplotlib.pyplot as plt
# defining the number of steps
num_steps = 1000
# function to generate random walk
def makeRandomWalk(n):
    x = numpy.zeros(n)
    y = numpy.zeros(n)
    for i in range(1, n):
        val = random.randint(1, 4)
        if val == 1:
            x[i] = x[i - 1] + 1
            y[i] = y[i - 1]
        elif val == 2:
            x[i] = x[i - 1] - 1
            y[i] = y[i - 1]
        elif val == 3:
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1
 return x,y
# plotting stuff:
plt.figure()
x,y=makeRandomWalk(num_steps)
plt.subplot(2,2,1)
plt.axis('equal')
plt.plot(x,y)
x,y=makeRandomWalk(num_steps)
plt.subplot(2,2,2)
plt.axis('equal')
plt.plot(x,y)
x,y=makeRandomWalk(num_steps)
plt.subplot(2,2,3)
plt.axis('equal')
plt.plot(x,y)
```

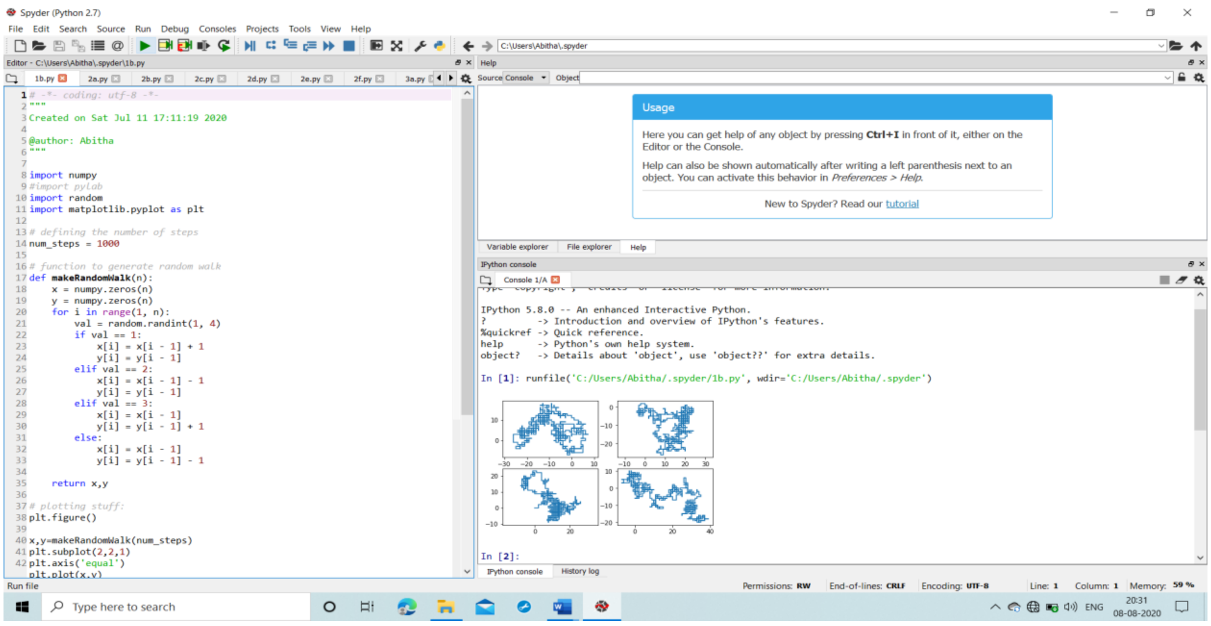
x,y=makeRandomWalk(num_steps)

plt.subplot(2,2,4)

plt.axis('equal')

plt.plot(x,y)

plt.show()



```
# -*- coding: utf-8 -*-
"""
Created on Sat Jul 11 17:11:19 2020

@author: Abitha
"""

import numpy
#import pylab
import random
import matplotlib.pyplot as plt

# defining the number of steps
num_steps = 1000

# function to generate random walk
def makeRandomWalk(n):
    x = numpy.zeros(n)
    y = numpy.zeros(n)
    for i in range(1, n):
        val = random.randint(1, 4)
        if val == 1:
            x[i] = x[i - 1] + 1
            y[i] = y[i - 1]
        elif val == 2:
            x[i] = x[i - 1] - 1
            y[i] = y[i - 1]
        elif val == 3:
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1

    return x,y

# plotting stuff:
plt.figure()

x,y=makeRandomWalk(num_steps)
plt.subplot(2,2,1)
plt.axis('equal')
plt.plot(x,y)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
IPython 5.8.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: runfile('C:/Users/Abitha/.spyder/1b.py', wdir='C:/Users/Abitha/.spyder')

In [2]:
```

Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Abitha\.spyder\1b.py
1b.py 2a.py 2b.py 2c.py 2d.py 2e.py 2f.py 3a.py
C:\Users\Abitha\.spyder
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Run file
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 1 Column: 1 Memory: 59 %
Type here to search
ENG 20:31 08-08-2020

Fig 2.1. Simulation Results for Random walk trajectories

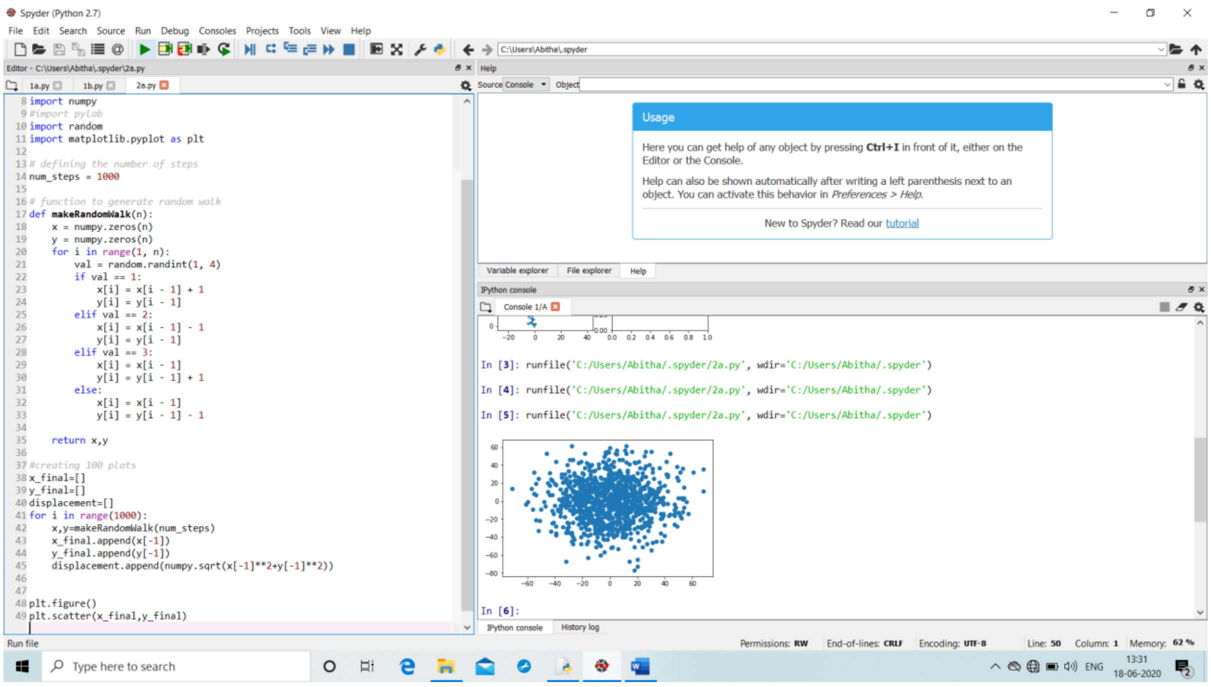
The first task is to create a random walk of 1000 steps and each trajectory will be list of 1000 *x* values and 1000 *y* values. The four such trajectories are also plotted.

2.2 PLOTTING DISPLACEMENT

For more understanding, we want to know what is the distance from the starting point (0,0) to the ending point (x_{1000}, y_{1000}). We could manually examine all plots, but it would be hard to see the common features. So, we use Python to generate all the random walks, but show us only a summary. We could create two arrays *x_final* and *y_final* to store the ending *x* and *y* positions and embedded it inside a *for* loop.

Adding this code:

```
#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
   x,y=makeRandomWalk(num_steps)
   x_final.append(x[-1])
   y_final.append(y[-1])
   displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))
plt.figure()
plt.scatter(x_final,y_final)
```



```
import numpy
#import pylab
import random
import matplotlib.pyplot as plt

# defining the number of steps
num_steps = 1000

# function to generate random walk
def makeRandomWalk(n):
    x = numpy.zeros(n)
    y = numpy.zeros(n)
    for i in range(1, n):
        val = random.randint(1, 4)
        if val == 1:
            x[i] = x[i - 1] + 1
            y[i] = y[i - 1]
        elif val == 2:
            x[i] = x[i - 1] - 1
            y[i] = y[i - 1]
        elif val == 3:
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1

    return x,y

#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
    x,y=makeRandomWalk(num_steps)
    x_final.append(x[-1])
    y_final.append(y[-1])
    displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))


plt.figure()
plt.scatter(x_final,y_final)
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [3]: runfile('C:/Users/Abitha/.spyder/2a.py', wdir='C:/Users/Abitha/.spyder')

In [4]: runfile('C:/Users/Abitha/.spyder/2a.py', wdir='C:/Users/Abitha/.spyder')

In [5]: runfile('C:/Users/Abitha/.spyder/2a.py', wdir='C:/Users/Abitha/.spyder')

In [6]:
```

Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\Abitha\.spyder
Editor - C:\Users\Abitha\.spyder\2a.py
1a.py 1b.py 2a.py
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Run file
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 50 Column: 1 Memory: 62 %
Type here to search
ENG 13:31 18-06-2020

Fig.2.2 Scatter plot of the end points

It turns out that random walks are partially predictable after all. Out of all the randomness comes systematic statistical behaviour. We can look at the distributions of various positions at various steps in making histograms of the positions of each simulation. We can also compute the mean by iterating through each time step from our simulation.

Adding this code:

import numpy

#import pylab

import random

import matplotlib.pyplot as plt

# defining the number of steps

num_steps = 1000

# function to generate random walk

def makeRandomWalk(n):

    x = numpy.zeros(n)

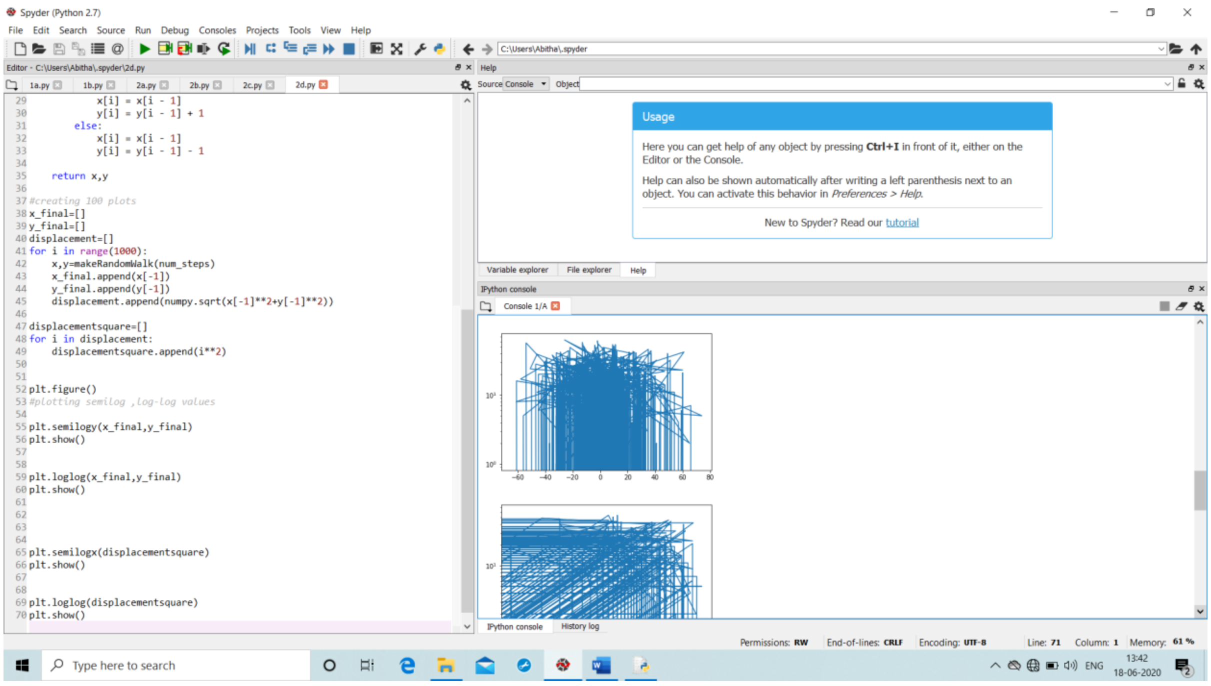
```
    y = numpy.zeros(n)
    for i in range(1, n):
        val = random.randint(1, 4)
        if val == 1:
            x[i] = x[i - 1] + 1
            y[i] = y[i - 1]
        elif val == 2:
            x[i] = x[i - 1] - 1
            y[i] = y[i - 1]
        elif val == 3:
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1
       return x,y
#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
    x,y=makeRandomWalk(num_steps)
    x_final.append(x[-1])
    y_final.append(y[-1])
    displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))


displacementsquare=[]
for i in displacement:
    displacementsquare.append(i**2)
```

plt.figure()

plt.hist(displacementsquare)

plt.show()



```
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1

    return x,y

#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
    x,y=makeRandomWalk(num_steps)
    x_final.append(x[-1])
    y_final.append(y[-1])
    displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))

displacementsquare=[]
for i in displacement:
    displacementsquare.append(i**2)


plt.figure()
#plotting semilog ,log-log values

plt.semilogy(x_final,y_final)
plt.show()


plt.loglog(x_final,y_final)
plt.show()




plt.semilogx(displacementsquare)
plt.show()


plt.loglog(displacementsquare)
plt.show()
```

Usage

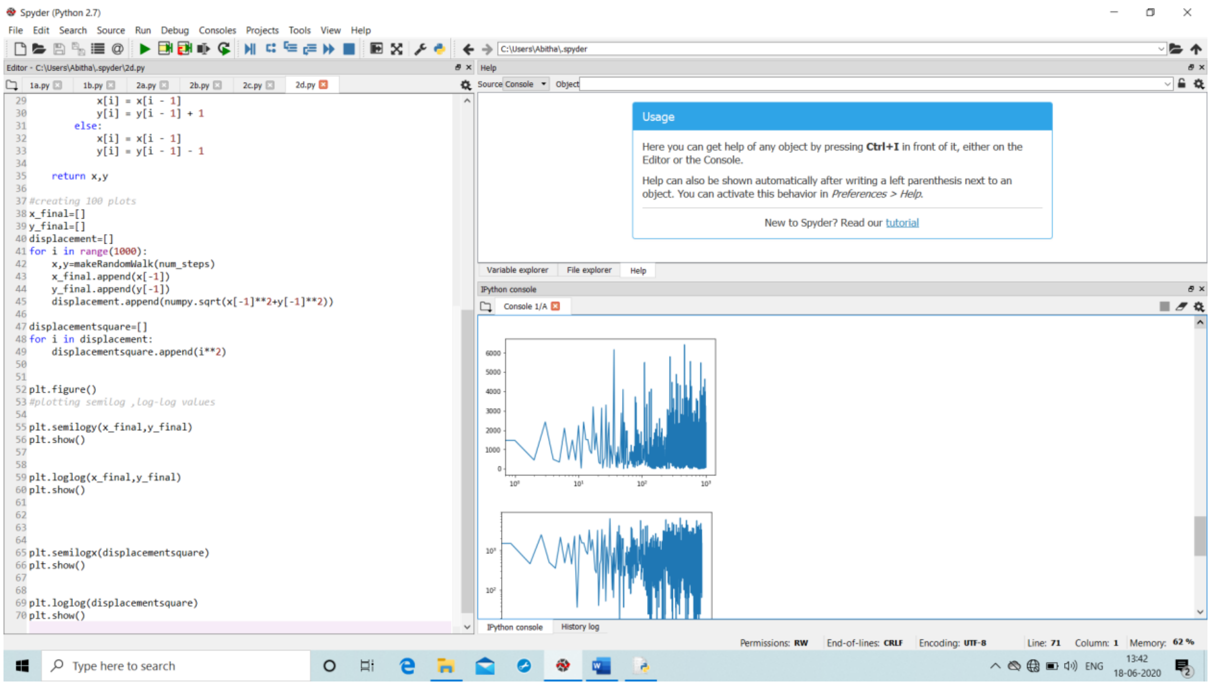
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Abitha\.spyder\2d.py
1a.py 1b.py 2a.py 2b.py 2c.py 2d.py
C:\Users\Abitha\.spyder
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 71 Column: 1 Memory: 61 %
Type here to search
ENG 13:42 18-06-2020

Fig 2.3. *Semi-log* axes of the random walk histogram



```
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1

    return x,y

#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
    x,y=makeRandomWalk(num_steps)
    x_final.append(x[-1])
    y_final.append(y[-1])
    displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))

displacementsquare=[]
for i in displacement:
    displacementsquare.append(i**2)


plt.figure()
#plotting semilog ,log-log values

plt.semilogy(x_final,y_final)
plt.show()


plt.loglog(x_final,y_final)
plt.show()




plt.semilogx(displacementsquare)
plt.show()


plt.loglog(displacementsquare)
plt.show()
```

Usage

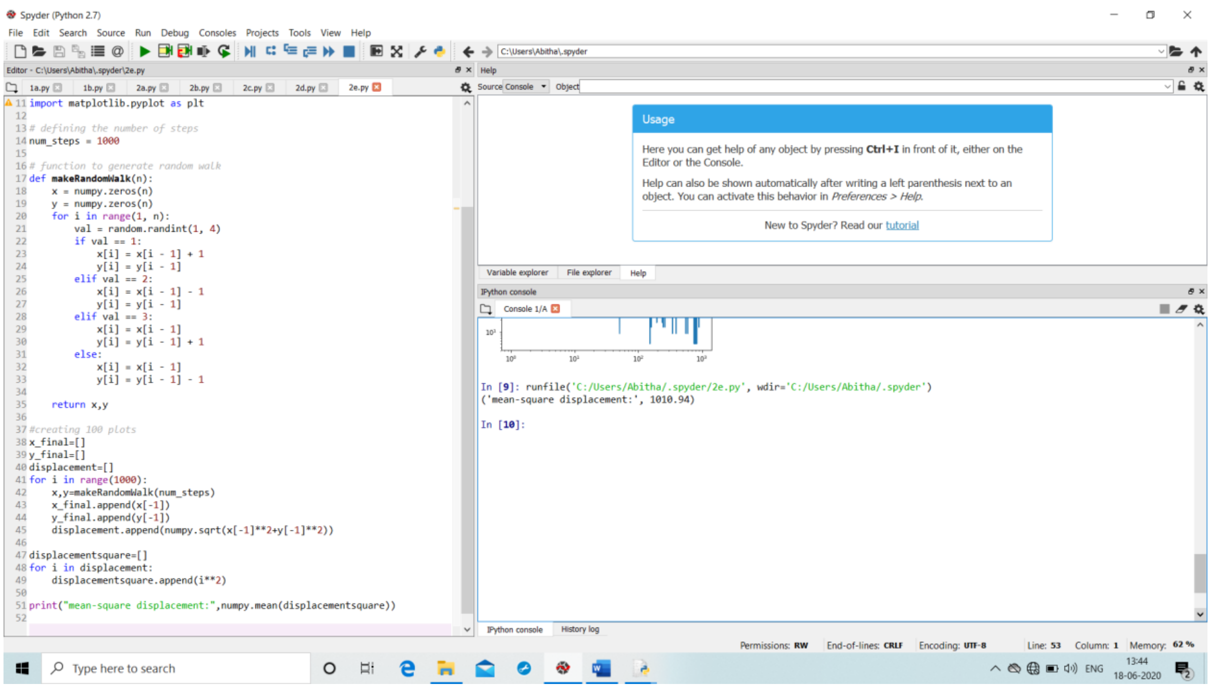
Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Abitha\.spyder\2d.py
IPython console
Console 1/A
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 71 Column: 1 Memory: 62 %
Type here to search
ENG 13:42 18-06-2020

Fig 2.4 *log-log* axes of the random walk histogram


Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Abitha\.spyder\2e.py
1a.py 1b.py 2a.py 2b.py 2c.py 2d.py 2e.py
C:\Users\Abitha\.spyder
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 53 Column: 1 Memory: 62 %
Type here to search
ENG 13:44 18-06-2020

```python
import matplotlib.pyplot as plt

# defining the number of steps
num_steps = 1000

# function to generate random walk
def makeRandomWalk(n):
    x = numpy.zeros(n)
    y = numpy.zeros(n)
    for i in range(1, n):
        val = random.randint(1, 4)
        if val == 1:
            x[i] = x[i - 1] + 1
            y[i] = y[i - 1]
        elif val == 2:
            x[i] = x[i - 1] - 1
            y[i] = y[i - 1]
        elif val == 3:
            x[i] = x[i - 1]
            y[i] = y[i - 1] + 1
        else:
            x[i] = x[i - 1]
            y[i] = y[i - 1] - 1

    return x,y

#creating 100 plots
x_final=[]
y_final=[]
displacement=[]
for i in range(1000):
    x,y=makeRandomWalk(num_steps)
    x_final.append(x[-1])
    y_final.append(y[-1])
    displacement.append(numpy.sqrt(x[-1]**2+y[-1]**2))

displacementsquare=[]
for i in displacement:
    displacementsquare.append(i**2)

print("mean-square displacement:",numpy.mean(displacementsquare))
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [9]: runfile('C:/Users/Abitha/.spyder/2e.py', wdir='C:/Users/Abitha/.spyder')
('mean-square displacement:', 1010.94)

In [10]:
```

Fig 2.5 Mean- square displacement of the random walk

2.3 RANDOM EVENTS

Rare event simulation involves extremely small but important probabilities. If we flip our imaginary coin once every second, then our string of heads and tails becomes a time series called a **Poisson process**. Flipping head is a rare event, and it raises an interesting question: What is the distribution of the waiting times from heads to next. This example is useful in demonstrating the problem of rare-event simulation and this probability maybe estimated by conducting repeated experiments using random number generator.

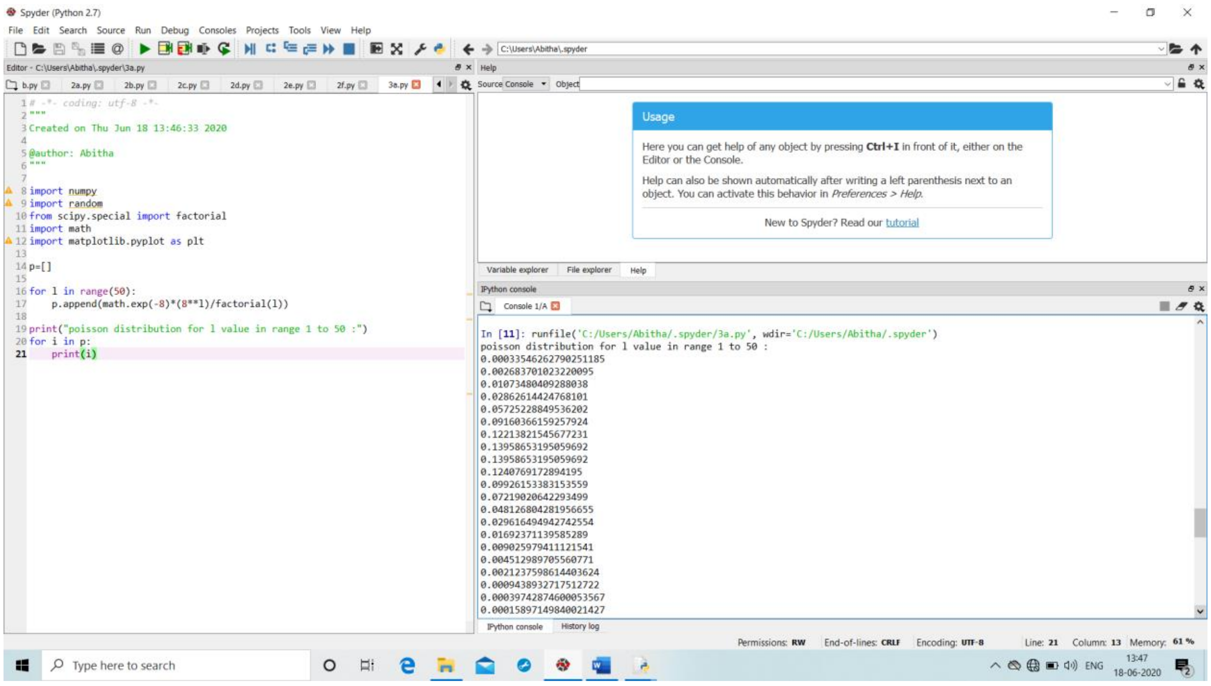
Adding this code:

import numpy

import random

from scipy.special import factorial

```
import math
import matplotlib.pyplot as plt
p=[]
for l in range(50):
    p.append(math.exp(-8)*(8**l)/factorial(l))
print("poisson distribution for l value in range 1 to 50 :")
for i in p:
    print(i)
```



```
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 18 13:46:33 2020

@author: Abitha
"""

import numpy
import random
from scipy.special import factorial
import math
import matplotlib.pyplot as plt

p=[]

for l in range(50):
    p.append(math.exp(-8)*(8**l)/factorial(l))

print("poisson distribution for l value in range 1 to 50 :")
for i in p:
    print(i)
```

```
In [11]: runfile('C:/Users/Abitha/.spyder/3a.py', wdir='C:/Users/Abitha/.spyder')
poisson distribution for l value in range 1 to 50 :
0.00033546262790251185
0.002683701023220095
0.01073480409288038
0.02862614424768101
0.05725228849536202
0.09160366159257924
0.12213821545677231
0.13958653195059692
0.13958653195059692
0.1240769172894195
0.09926153383153559
0.07219020642293499
0.048126804281956655
0.029616494942742554
0.01692371139585289
0.009025979411121541
0.004512989705560771
0.0021237598614403624
0.0009438932717512722
0.00039742874600053567
0.00015897149840021427
```

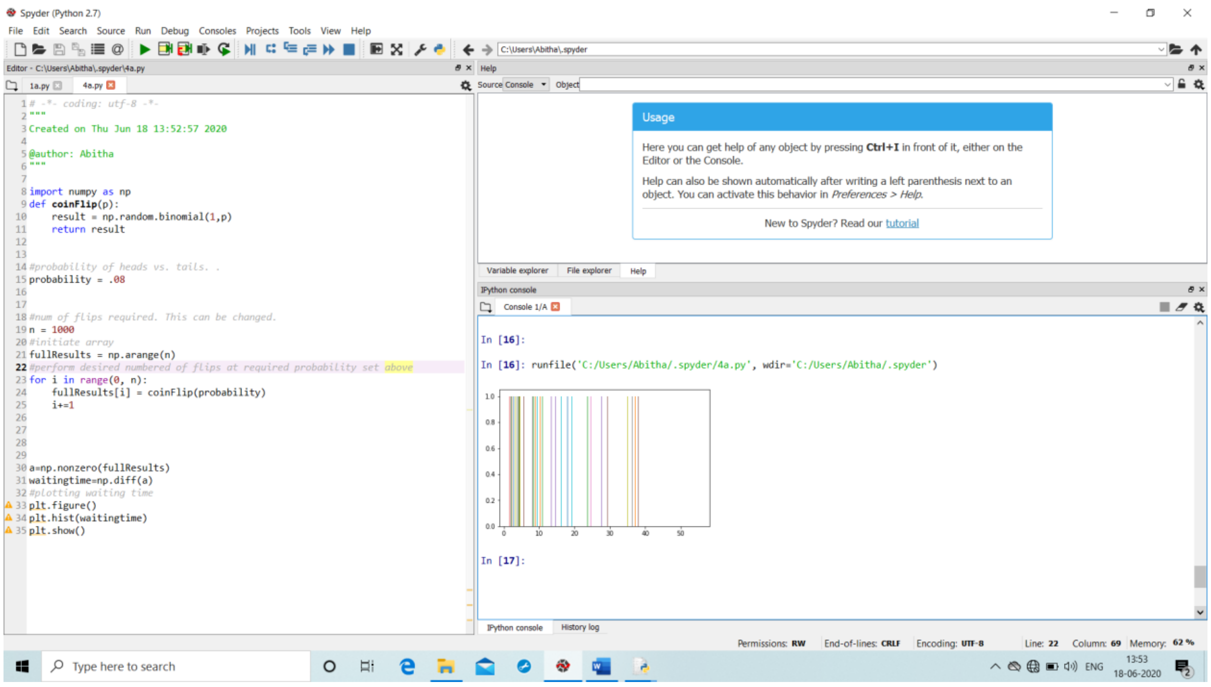
Fig 2.6 Rare event distribution

We want to know the distribution of the waiting times from one head to next. To know this, we can make a long list of ones and zeros, then search it for each occurrence of a 1 by using NumPy's *np.nonzero* function. NumPy's *np.diff* function will take the difference of the successive entries in an array.

Adding this code:

import numpy as np

```
def coinFlip(p):
    result = np.random.binomial(1,p)
    return result
#probability of heads vs. tails. .
probability = .08
#num of flips required. This can be changed.
n = 1000
#initiate array
fullResults = np.arange(n)
#perform desired numbered of flips at required probability set above
for i in range(0, n):
    fullResults[i] = coinFlip(probability)
    i+=1
a=np.nonzero(fullResults)
waitingtime=np.diff(a)
#plotting waiting time
plt.figure()
plt.hist(waitingtime)
plt.show()
```


Spyder (Python 2.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:\Users\Abitha\.spyder\4a.py
1a.py 4a.py
C:\Users\Abitha\.spyder
Help
Source Console Object
Variable explorer File explorer Help
IPython console
Console 1/A
IPython console History log
Permissions: RW End-of-lines: CRLF Encoding: UTF-8 Line: 22 Column: 69 Memory: 62 %
Type here to search
13:53
18-06-2020

```python
# -*- coding: utf-8 -*-
"""
Created on Thu Jun 18 13:52:57 2020

@author: Abitha
"""

import numpy as np
def coinFlip(p):
    result = np.random.binomial(1,p)
    return result


#probability of heads vs. tails. .
probability = .08


#num of flips required. This can be changed.
n = 1000
#initiate array
fullResults = np.arange(n)
#perform desired numbered of flips at required probability set above
for i in range(0, n):
    fullResults[i] = coinFlip(probability)
    i+=1




a=np.nonzero(fullResults)
waitingtime=np.diff(a)
#plotting waiting time
plt.figure()
plt.hist(waitingtime)
plt.show()
```

Usage

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.

Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our tutorial

```
In [16]:

In [16]: runfile('C:/Users/Abitha/.spyder/4a.py', wdir='C:/Users/Abitha/.spyder')
```

```
In [17]:
```

Fig 2.7 Waiting Times

2.4 APPLICATIONS

- ❖ In population genetics, random walk describes the statistical properties of **genetic drift**.
- ❖ In physics, random walks and some of the self-interacting walks play a role in **quantum field** theory.
- ❖ In computer science, random walks are used to estimate the **size of the web**.
- ❖ Rare event simulation is used in modern packet-switched **telecommunication** networks, in order to reduce delay in real-time video traffic.
- ❖ In insurance settings, the overall wealth of the insurance companies is modelled as a rare event process.

CHAPTER 3

CONCLUSION

On the whole, this internship was a useful experience. I have gained new knowledge and skills. I achieved several learning goals, and have moved a step further in achieving other. Fellowship has proved to be satisfactory and it has allowed as an opportunity to get an exposure of the practical implementation of theoretical fundamentals.

Here during the fellowship period I developed my new skills in following software/tools:

1. Python (how different packages and libraries works and their uses)
2. NumPy
3. SciPy
4. Random walk trajectories

I would like to once again appreciate everyone who has made my fellowship training a superb experience.

REFERENCES:

1. Jesse M. Kinder, Philip M. Nelson, “A Student’s Guide to Python for Physical Modelling”.
2. Eli bressert, “SciPy and NumPy”.

3. GeeksforGeeks
4. Github
5. NumPy tutorials