MA/CS 341 – Scientific Computing

numpy/matplotlib Mini-Project

1. First, on a Colab Jupyter Notebook, work through Running a Parallel Job (***up to "Running the Serial Job on a Compute Node"***), https://rse.shef.ac.uk/hpc-intro-tuos-citc/16-parallel/index.html . This is a nice application of the Monte Carlo Pi simulation using numpy. Note, for pedagogical purposes, the points are generated and saved in a list, and in practice our previous code would be more efficient.

2. The tutorial scaffolds the work, starting with the calculation, then adding memory and time estimates. This is a good work flow. As you do this, save your notebook often.

3. Two implementation details to consider, since the tutorial assumes it is run at the command prompt, and not in a notebook:
    a. The code in the tutorial assumes an input parameter at the command prompt for the size of the Monte Carlo simulation. To run the code as given, you can just hardcode an integer. Note that below you will be asked to modify the code to run over a number of sample sizes.
    b. The code in the tutorial defines a `main()`, so to run it you can:
        i. Run it as is, or call `main()` from a new execution group.
        ii. You could take the code out of `main()` and run it, making sure to unindent, without defining it as a procedure.

4. Update the code to loop over the number of samples, say 100 (10^2) to 1,000,000,000 (10^9), and `print` the number of samples, along with the estimate of Pi, the memory used, and the time (in seconds).

5. When you are satisfied that is working, calculate and output the relative error for the estimated and actual values of π, using the `math.pi` in python, and besides printing to the notebook, output to a `.csv` file as well.

6. Input the `.csv` file back into the notebook, and visualize the output with a variety of scatter plots, using `matplotlib`. Probably will be ***N*** vs. error, ***N*** vs. memory, and ***N*** vs. compute time. The scales are over a large range, so you should consider a log-log plot.

7. Answer the following questions in a text-based group in your Notebook:
    a. How much more accurate does the estimate of π become as ***N*** is increased? When ***N*** is increased by a factor of 10, does the error decrease by a factor of 10?
    b. How does the amount of memory required scale as ***N*** is increased? Do you expect that it would be linear? How would the slope of the trendline in the log-log plot demonstrate that?
    c. How does the amount of time to calculate scale as ***N*** is increased? Do you expect that it would be linear? How would the slope of the trendline in the log-log plot demonstrate that?

8. Some helpful commands:
    a. You can create a `numpy` array with:
    `import numpy as np`
    your-numpy-array-name `= np.array( [] )`
    b. You can then append to the `numpy` array with:
    your-numpy-array-name `= np.append(` ***your-numpy-array-name*** `,` ***your-variable-name*** `)`
    c. You can reshape a `numpy` array, to make it two-dimensional, with:
    your-numpy-array-name = ***your-numpy-array-name***`.reshape(` ***rows*** `,` ***columns*** `)`
    d. The following are two, of many, options to write to a `.csv` file:
        i. Following the above, if you have a `numpy` array that is shaped with the number of rows and columns, you can then write to the `.csv` file with:
        `np.savetxt(''`***your-file-name***`.csv',` ***your-numpy-array-name***`, delimiter=',')`
        ii. You may also use the csv.writer to write one row at a time:
        `f = open('`***your-file-name***`.csv', 'w', newline='')`
        `writer = csv.writer(f)`
        then, inside your loop, create a list of the values in that iteration and write to the file:
        `row = [`***my-var1***`,` ***my-var2***`,` and so on… `]`
        `writer.writerow(row)`
    e. Don't forget to close the file when you are done writing to it.
    f. Reading in .csv file and getting the first and third columns, where in the sample code below N is in the first column and error is in the third column:
    `import numpy as np`

```
data = np.loadtxt(fname='your-file-name.csv',
delimiter=',')
NVal=data[ : , 0 ]
ErrVal=data[ : , 2]
```

g. Creating a log-log plot (note the `pyplot.loglog` doesn't seem to work with small exponents):

```
NVal=data[ : , 0 ]
logN =[]
for n in range(len(NVal)):
   logN.append( math.log(NVal[n],10)
ErrVal=data[ : , 2]
logErr =[]
for err in ErrVal:
   logErr.append( math.log(err,10)
matplotlib.pyplot.scatter(logN,logErr)
```

9. Adding a trendline and getting the slope, https://stackoverflow.com/questions/49460596/add-trendline-with-equation-in-2d-array

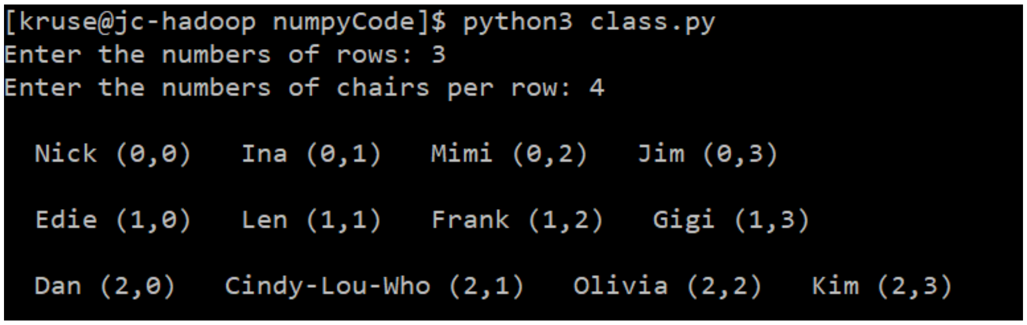
```
z = np.polyfit(logN, logErr, 1)
p = np.poly1d(z)
matplotlib.pyplot.plot(logN, p(logN))
print(z[0],z[1])
# z[0] is the slope of the trendline
# z[1] is the y-intercept of the trendline
```

10. Save your Notebook to GitHub, and then upload the `.ipynb` into Moodle.

11. Next, on `jc-hadoop`, at the ***command prompt***, and ***not*** as a Jupyter Notebook, write a Python program which generates a seating chart for a classroom.
    a. The program should prompt the user for the number of rows and chairs per row.
    b. The code should have a pre-defined list of names to use (make sure you have enough for the size of the classroom, `rows*columns`, and each name should only be used once.
    c. For this assginment, you should create a two-dimensional array of student names USING `numpy`.
    d. Import the `numpy` module and its `random` module:

```
import numpy as np
from numpy import random
```

e. Start with a 1-D `numpy` array and randomly fill names into the array.
f. Then `reshape` the array to be 2-D, rows by cols, which will make it easier to output.
g. The output should be the name of the student, and their "position" in the classroom, *(row , column)* type format. A sample run:



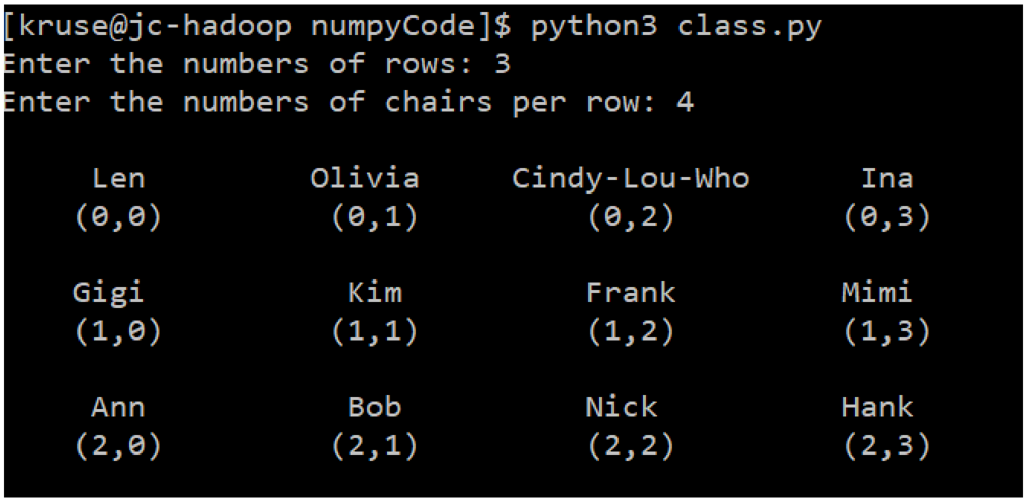
```
[kruse@jc-hadoop numpyCode]$ python3 class.py
Enter the numbers of rows: 3
Enter the numbers of chairs per row: 4

  Nick (0,0)   Ina (0,1)   Mimi (0,2)   Jim (0,3)

  Edie (1,0)   Len (1,1)   Frank (1,2)   Gigi (1,3)

  Dan (2,0)   Cindy-Lou-Who (2,1)   Olivia (2,2)   Kim (2,3)
```

h. If you are looking for a challenge, format the output so the names and locations are centered, using the longest possible name to determine column-width. A sample run:



```
[kruse@jc-hadoop numpyCode]$ python3 class.py
Enter the numbers of rows: 3
Enter the numbers of chairs per row: 4

     Len          Olivia     Cindy-Lou-Who       Ina
    (0,0)          (0,1)         (0,2)          (0,3)

    Gigi            Kim          Frank          Mimi
    (1,0)          (1,1)         (1,2)          (1,3)

     Ann            Bob          Nick           Hank
    (2,0)          (2,1)         (2,2)          (2,3)
```