

# ENGG1003 - Lab Week 5

Brenton Schulz

## 1 Pre-Reading

Read through the following textbook sections:

- Section 2.4 Random Numbers: [https://link.springer.com/chapter/10.1007/978-3-030-16877-3\\_2#Sec23](https://link.springer.com/chapter/10.1007/978-3-030-16877-3_2#Sec23)
- Section 3.3.4 Example: Random Walk in Two Dimensions: [https://link.springer.com/chapter/10.1007/978-3-030-16877-3\\_3#Sec10](https://link.springer.com/chapter/10.1007/978-3-030-16877-3_3#Sec10)

## 2 Revision - Summations and for Loops

The Week 3 lab included Exercise 3.11 from the textbook. This exercise requires an “infinite sum” to be evaluated of the form:

$$\pi = 8 \sum_{k=0}^{\infty} \frac{1}{(4k+1)(4k+3)} \quad (1)$$

Breaking down this notation:

- $\Sigma$  means “sum”
- $k = 0$  below the  $\Sigma$  means that  $k$  is a *variable* which takes integer values starting at zero
- $\infty$  above the  $\Sigma$  means that there is no limit to how large  $k$  can become
- Each value of  $k$  gets substituted into the  $\frac{1}{(4k+1)(4k+3)}$  expression and all results added together
- “ $\pi =$ ” implies that the sum will approach  $\pi$  as  $k$  approaches  $\infty$

Putting it all together, the first few terms of Equation 1 can be written out as:

$$\frac{1}{(4 \times 0 + 1)(4 \times 0 + 3)} + \frac{1}{(4 \times 1 + 1)(4 \times 1 + 3)} + \frac{1}{(4 \times 2 + 1)(4 \times 2 + 3)} + \dots \quad (2)$$

It turns out that expressions of this form map quite cleanly to `for` loops when programming:

- An expression is evaluated multiple times for different values of one variable
- A sum variable can “accumulate” calculations inside the loop in the form `sum = sum + <something>`
- $k$  as the “summation variable” becomes the “loop variable” `k`
  - Pay close attention to the starting value of `k=0`
  - Computer’s can’t count to  $\infty$ , so choose a “reasonable” value of, say, 100 000 for the final value of `k`
- The  $\frac{1}{(4k+1)(4k+3)}$  expression becomes the Python expression `1 / ((4*k+1) * (4*k+3))` (note double level nesting of `( ) ( )` in the denominator - without the outer `( )` the expression becomes  $\frac{4k+3}{4k+1}$ ).

### Task 1: Python Implementation of Equation 1

Given all the above, write a Python script which implements Equation 1 for `k=0` to `k=99` (ie: with `range(0,100)`).

Remember to multiply the result of the summation by 8 to get an approximation of  $\pi$ !

The final value should be 3.13659.

If you get 0.39207 the multiply by 8 was omitted.

Once your code is correct repeat the calculation with `range(0,100000)` to get the more accurate result of 3.141587.

### 3 Random Numbers

#### Task 2: Simplified Random Walk

Section 3.3.4 of the textbook shows a “random walk” program which randomly moves a point N, S, E, or W 1000 times.

In this task you will repeat this problem with floating point steps. ie: the x and y movements will be drawn from a uniform distribution using `np.random.uniform()`. This will make the simulation roughly analogous to brownian motion - the movement of particles in a liquid or gas.

The program is to fill in two numpy arrays: one of x locations and one of y. The array index will then become a “time” variable. This will allow the simulation results to be plotted with `plot(x, y)`.

Write a Python script which implements the follow pseudocode to move a point around randomly and plot the result:

```
BEGIN
  N = 1000
  x = an array of N zeros
  y = an array of N zeros
  FOR each element, n, in x and y from 1 to N:
    x[n] = x[n-1] + a random float from [-1,1)
    y[n] = y[n-1] + a random float from [-1,1)
  ENDFOR
  plot the set of x-y points, drawing lines between points
END
```

You may call `np.random.uniform()` within the loop or create a pair of additional arrays containing random numbers before entering the loop.

## 4 The CSV Format

As seen in lectures, we will be studying the use of the pandas library for reading CSV (comma separated value/variable) files.

The CSV format allows for “spreadsheet” data to be stored as “human-readable” ASCII text. Column headings and numerical values are written as plain text and columns are separated by commas<sup>1</sup>. Rows are separated by new lines.

### Task 3: Opening CSV Files

This lab will make use of a real scientific dataset from the Bureau of Meteorology. The Bureau of Meteorology makes a vast amount of climate data available online here: <http://www.bom.gov.au/climate/data/index.shtml>

You are welcome to browse data as it suits your interest but this lab will make use of the monthly rainfall data for the Nobbys Signal Station weather station on the coast of Newcastle.

The data used in this lab (see `rainfall-1.csv` on Blackboard) has been extracted from the zip containing the Nobbys rainfall csv data from here:

[http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p\\_display\\_type=monthlyZippedDataFile&p\\_stn\\_num=061055&p\\_c=-745551072&p\\_nccObsCode=139&p\\_startYear=](http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p_display_type=monthlyZippedDataFile&p_stn_num=061055&p_c=-745551072&p_nccObsCode=139&p_startYear=)

Extract the zip and open the three included files in a **text editor**. The lab computers have Notepad++ installed but you could also use Notepad or, at worst, open them in PyCharm.

Observe the “human readable” nature of the csv format. You may wish to keep this file open for debugging purposes.

Observe that the same rainfall data is provided in two layouts:

- `IDCJAC0001_061055_Data1.csv` contains one row per month with the Year and Month columns indicating which month (from January 1862 to February 2021) the rainfall figure is from
- `IDCJAC0001_061055_Data12.csv` contains a 2D grid where the row indicates a year and column indicates a month.

Note that there are several null values in the file ending `_Data12.csv`. These are missing data points. Missing data in `_Data1.csv` is indicated with missing rows where adjacent year/month combinations are discontinuous (eg: It jumps from 1864, 08 to 1867, 01).

For use in the next task download the gap-free subset of this data, `rainfall-1.csv`, from Blackboard.

In this course we will use the pandas library to read csv files. Before it can be used you must run `pip install pandas` to install it.

---

<sup>1</sup>Some variations allow for space or tab separation while still being classed as “CSV” files.

## Task 4: Reading and Analysing Rainfall Data

This task is presented in multiple parts and is presented in a more “real” style compared to previous lab tasks. You are welcome to discuss problem solving strategies with your lab peers as well as demonstrators.

Background lecture material:

See Slide 18f of the Thursday Week 4 lecture slides for a pandas example. The recording timestamp is: <https://www.youtube.com/watch?v=GZmJTGqh5pw#t=50m45s>

1. **Download `rainfall-1.csv` from Blackboard.**
2. Write a Python script which uses pandas to open `rainfall-1.csv` and use the `head()` function to print the first few lines. Ensure that they match what you saw in the text editor during the previous task.
3. Extend your script to load rainfall data from `rainfall-1.csv` into a single numpy array called `rainfall`.

Your array should be 1584 elements. Data is provided from January 1867 to December 1998.

Observe the first line of the file to work out the correct column heading.

4. Plot the timeseries data. A simple `plot(rainfall)` will suffice here as we are visualising the data for development purposes - not producing a plot for publication.
5. Summary statistics for Nobbys Signal Station are provided by the BOM at the bottom of this page: [http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p\\_nccObsCode=139&p\\_display\\_type=dataFile&p\\_startYear=&p\\_c=&p\\_stn\\_num=061055](http://www.bom.gov.au/jsp/ncc/cdio/weatherData/av?p_nccObsCode=139&p_display_type=dataFile&p_startYear=&p_c=&p_stn_num=061055)

From data in the `rainfall` array calculate the mean, lowest, and highest rainfall figures for each month and compare them with the BOM's calculations. Use one (or more) `for` loops to perform this analysis.

Note that the `%` (modulus) operator may be useful when converting an array index to a month. ie: the index modulo 12 will provide a month number where 0 is January and 11 is December.

It is simpler to complete this task with twelve variables (ie: one running sum for each month) but an attempt should be made to solve this problem with two *nested* `for` loops: an outer loop which counts months and an inner loop which tests every index for inclusion in that month. The nested loop method allows for the use of an array to store the monthly means and greatly reduces the code complexity.

6. **Extension:** Plot a *histogram* of the monthly rainfall data. To do this you can use the `plt.hist()` function from `matplotlib.pyplot`. If the data is stored in the numpy array `rainfall` then the histogram can be plotted with:

```
plt.hist(rainfall)
plt.show()
```