COMP 1012: Lab 9

August 3nd to 7th, 2020

Material covered

- Numpy

Notes

You must save all your code as a script. The program should be written using vectorization.

Grading

Total marks is 2.

If you have completed exercise 1, you can work on challenge and super challenge questions. Challenge and super challenge questions are optional. You will either receive 0, 1 or 2 mark for completion of the mandatory exercise.

Note - You do not have to successfully complete challenge and super challenge questions to get the full marks.

You will receive 2 marks for correct solution to Exercise 1. No submission or empty submission will be a direct 0. Use only what have been taught in class or described in this lab.

Exercises

Exercise 1: Random walk

Fair use

This problem is adapted from Exercise 37 of Langtangen's "A Primer on Scientific Programming with Python".

Simulating gas particles in a closed space is something you can do with a Monte Carlo type simulation – simulate a period of time (using a loop) and "move" particles around (change values for (x, y) coordinates) for each unit of time. This kind of simulation is called a "random walk".

Download the file `randomwalk.py`. This implements a Monte Carlo simulation for a random 2-dimensional walk. It **uses** `numpy` arrays, but it **does not** use them *idiomatically*.

Your task for this exercise is to change this program to use `numpy` arrays idiomatically. That is, the block

```
for i in range(np):
    direction = random.randint(1, 4)
    if direction == NORTH:
        ypositions[i] += 1
    elif direction == SOUTH:
        ypositions[i] -= 1
    elif direction == EAST:
        xpositions[i] += 1
    elif direction == WEST:
        xpositions[i] -= 1
```
should be rewritten to use `numpy` arrays of random integers.

Challenge: multidimensional arrays

Rewrite the code so that (x, y) coordinates are represented with a 2-dimensional `numpy` array. Here's how you can start:

```
coords = np.zeros(shape=(np, 2))
# xpositions = numpy.zeros(np)
# ypositions = numpy.zeros(np)
```

Remember that you can index entire *columns* of a 2-dimensional array by using the notation `[:,n]` to get the n^{th} column.

Super challenge: no loops

Rewrite the code so that it has **no** loops. You'll need to think in *3* dimensions to accomplish this.

Answer some question first:

1. *Can* you simulate time without loops?
2. Each unit of time depends on the unit of time *before* it. Can different matrices in a 3-dimensional `numpy` array *depend* on each other?

If you can answer "yes" to both of these questions, then proceed!