FE-520 Assignment 3

Dan Wang, Zhiyuan Yao

Sprint 2020

Submission Requirement:

For all the problems in this assignment you need to design and use Python 3, output and present the results in nicely format. Please submit a written report (pdf), where your results and copy your code should be readable, and 4 python files (main.py, generator.py, point.py and MCTest.py). Your grade will be evaluated by combination of report and code. You are strongly encouraged to write comment for your code, because it is a convention to have your code documented all the time. In your python file, you need contain both function and test part of function. Python script must be a '.py' script, Jupyter notebook '.ipynb is not allowed. Do NOT copy and paste from others, all homework will be firstly checked by plagiarism detection tool.

1 Class practice 30 pts

1.1 Define class (20pts)

Create a class called Rectangular, with two data attributes: length and width. They should be assigned in constructor (__init__()) It should also have two function attributes called area() and perimeter() which return area and perimeter of this rectangular respectively. Here is an example of the class structure:

```
>>> class Rectangular:
...     <Your initializer>
...     <Defination of area()>
...     <Defination of perimeter()>
>>> myRec = Rectangular(10,20)
>>> print(myRec.area())
200
>>> print(myRec.perimeter())
60
```

1.2 Numpy applying on class (10 pts)

1. define two numpy array with size 10, named with length and width.

2. test your your class Rectangular with input as np array. (Here you should have 10 output for area and perimeter).

2 Display Time (20pts)

Create a Time class and initialize it with hours, minutes and seconds.

1. Make a method addTime which should take two time object and add them. E.g. if your original initial parameter is (2 hour and 50 min and 10 seconds), then you could call this method with another input parameters (1 hr and 20 min and 5 seconds) , then output is (4 hr and 10 min and 15 seconds)
2. Make a method displayTime which should print the time (the initial parameter).
3. Make a method DisplayMinute which should display the total seconds in the Time. E.g.- (1 hr 2 min) should display 3720 seconds.

3 Uniform Distributed Random Number Generator (50pt)

You can not use any random number generators in packages like Numpy, because you are required to build your own generator in this question.

Almost all the random numbers generated by computer are pseudo random numbers, because they are generated by a formula. A good random number generator is very important for financial simulations, such as Monte-Carlo method.

The goal of this assignment is to let you understand how computer generate random numbers and create pseudo random number generators which can generate 0-1 uniform-distributed random numbers by yourselves.

The probability density function (PDF) of 0-1 uniform distribution is

$$f(x) = \begin{cases} 1 & for\ x \in [0,1] \\ 0 & otherwise \end{cases}$$

It means each point in interval $[0, 1]$ has the same probability to be chosen.

1. For implementation purpose, people use different algorithms to generate pseudo-random numbers. One of the most famous algorithms is called **Linear Congruential Generator**(LCG). This generator can yield a sequence of random numbers $\{X_i\}$. The recurrence relation of this algorithm is:

   $$X_{n+1} = (aX_n + c)\ mod\ M$$

   here a, c, M are all parameters. a is called multiplier which satisfies $0 < a < M$, c is called increment which satisfy $0 \leqslant c < M$, M is the modulus which is also

stands for the maximum range of this sequence. You can find more information about the common parameters setting in the link above. There is another parameter X_0 which is so called **seed**. You can generate totally different sequence by different seeds. In the formula above *mod* stands for the modulus operation, for example: $7 \ mod \ 3 = 1$ due to $7 = 2 \times 3 + 1$.
As you can guess, this generator can generate a sequence of integers within $[0, M)$. To create a uniform distributed sequence, you only need to divide the sequence above by M.
For example, now I have a setting: $X_0 = 1$, and $a = 1103515245,\ M = 2^{32},\ c = 12345$.

$$\begin{aligned} X_1 &= (a \cdot X_0 + c) \ mod \ M = 1103527590 \\ X_2 &= (a \cdot X_1 + c) \ mod \ M = 25248852323 \\ &... \end{aligned}$$

Then the uniform distributed sequence will be

$$\{\frac{1}{2^{32}}, \frac{1103527590}{2^{32}}, \frac{25248852323}{2^{32}}, \cdots\} \sim \{2.328 \times 10^{-10}, 0.2569, 0.5879, \cdots\}$$

There are some variants of LCG, one of them has such recurrence relation:

$$X_{n+1} = (X_n(X_n + 1)) \ mod \ M$$

The seed of this generator has to satisfy $X_0 \ mod \ 4 = 2$.
Now it's your turn to implement it in Python 3.

- (15 pt) Create a file called generator.py. In this file, create a class called **LCG** whose instance can generate random numbers by Linear Conguential Generator algorithm. It should have these data attributes: self.seed, self.multiplier, self.increment, self.modulus which are assigned in initializer (__init__()). The parameters should be passed by argument from outside of the class. This class's instance should also at least have function attributes allow me: 1) get the seed; 2) set the seed; 3) initialize the generator (start from the seed); 4) give me the next random number; 5) give me a sequence (list) of random number, the length should be passed by argument.

- (10 pt) In generator.py, create an inherited class called **SCG** from class LCG you have created. You are required to implement the variant of LCG I mentioned above in this class. It should have the same interfaces (attributes) with LCG, but different recurrence relation. Remember the seed of this generator has to satisfy $X_0 \ mod \ 4 = 2$, so you need to check this condition in initializer, raise error with customized error information if the seed doesn't satisfy this condition.
  Hint: To better reuse your code, your class should have a function attribute to specify the recurrence relation. You can only override this function in your inherited class.

- (Bonus 3pt) Specify __next__() or __iter__() attributes in your class to make the instance of your class compatible with iter() or next(), this kind of generator can yield me one random number for each call.
- A good code style like following good conventions, writing comment and doc string for your code will earn additional bonus.

2. To test the performance of your generator, one way is to use a simple Monte-Carlo method to test it. Consider a square and its inscribed circle in a 2-D rectangular coordinate. The length of square and the diameter of the circle are 2, and the center of them are both origin of this coordinate. See 2.

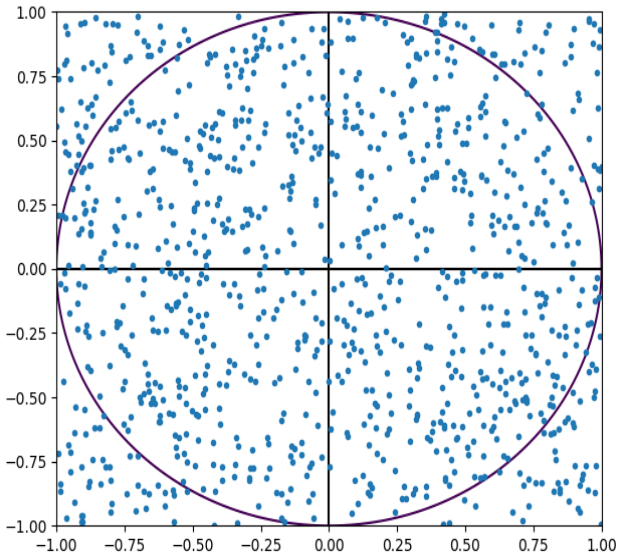

Figure 1: $x^2 + y^2 = 1$, and its circumscribed square. Blue points are uniform distributed random points

You can populated the points in this 2-D coordinate within this square, and count the number of points which fall in the circle. Then dividing this number by the number of all points will give you an estimate of ratio of area between inscribed circle and its square . Since you have already have theoretical result of this ratio:

$$ratio = \frac{\pi r^2}{4r^2} = \frac{\pi}{4} \sim 0.78539816339$$

where r is the radius of the circle, thus the length of square is $2r$. The closer your result is to this ratio, the better your random generator performs.

Now you need to use Python to do this:

- (15 pt) Create a file called point.py. Write a class to represent the points in rectangular coordinate. It should have the data attributes that store its x coordinate and y coordinate, and also a function attribute `distance` to calculate its distance from origin point.

- (10 pt) Create a file called MCTest.py. Import the classes you created above (generators and point) from Python modules. Populate 10,000,000 random points within this square. To do this, you only need to populate 2×10^7 0-1 uniform distributed random numbers firstly, then re-scale them into $[-1, 1]$, and pair them into 1×10^7 points. Test both generators.
  Determine whether points are within the circle. Hint: You can do this by computing its distance from origin, if it is smaller than 1 (the radius), it is within the circle.
  Give the estimate of above ratio, and the difference between yours and theoretical one.
- You are also encouraged to record the time consumed by your program. Hint: use `time` package.

For this question, you need to submit 3 file: generator.py, point.py and MCTest.py.