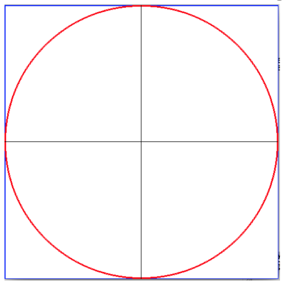
CSCI 2202 Computer Modelling for Scientists Due: 3 Feb 2021

Lab 6: Monte Carlo Simulations & Arrays

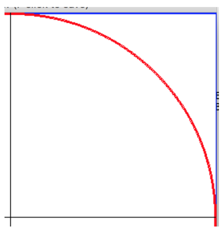
1. The idea of using a probabilistic (random) simulation to model a physical process is called a Monte Carlo simulation.

   We can *simulate* a physical process to create an estimate of π. The physical process in question is that of throwing darts at a target. Imagine a square board of side 2 units, with a circular target of radius 1 unit drawn in the centre $(0, 0)$ of the board: 

   The idea: Throw a large (known) number of darts at the board. The fraction of the darts that fall within the circle will be proportional to the area of the circle. Assume all darts thrown hit the board:

$$\frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi(1)^2}{(2)^2} = \frac{\text{Num. darts in circle}}{\text{Total num. darts}}$$

   The mechanics of doing the simulation simplify considerably if we conduct the simulation in the first quadrant of the target:



   The part of the square has corners (0, 0) (1,0) (1,1) (0,1). The quarter circle has radius 1 unit. If we throw darts at the quarter square then, following the calculation above:

$$\frac{\text{Area of 1/4 Circle}}{\text{Area of 1/4 Square}} = \frac{\pi(1)^2/4}{(1)^2} = \frac{\text{Num. darts in 1/4 circle}}{\text{Total num. darts thrown}}$$

   From this, we obtain an estimate of π:

$$\pi_{est} = \frac{4 \times \text{Num darts in 1/4 circle}}{\text{Total num. Darts thrown}}.$$

- **Throwing a dart:** A thrown dart falls (randomly) on the 1/4 board. The coordinates of the dart hits are: $0 \leq x_{throw} < 1$ and $0 \leq y_{throw} < 1$. Each throw can be simulated by a random number generator. Obtain this by importing the `random` module. Then `x_throw = random.random()` generates a random float in between 0 and 1 and similarly for `y_throw`
- **Count hits:** To count darts that hit the circle, we use the throw above, with coordinates (x_{throw}, y_{throw}) and check that it falls within a circle of radius $r = 1$.
  (*hint:* the boundary of the red circle is given by $x^2 + y^2 = r^2 = (1)^2$.)
  All dart throws: (x_i, y_i) $(1 \leq i \leq N)$ such that $x_i^2 + y_i^2 \leq r^2 = 1$, fall inside the circle.
- **Run simulations** of $N = 10, 100, 1000, 10000$ dart throws and compare the estimates of π. Use `math.pi` as a nominative "exact" value. Make a table: (Number of trials, Estimate, Relative Error).
  Note: Relative Error = (measured - exact)/exact

Storing results in a `numpy` array.
A one-dimensional `numpy` array is a list of values, *all of the same type*, indexed by nonnegative integers.

- Create an numpy array of size about $n = 6$.
- (`import numpy as np` at the top of the program).
- `np.zeros(k)` produces `[0., 0., ..., 0]` (an array of k zeros)
  (It also serves to initialize the array.)
- Store the estimates, by range, in the array.
  For example, `a[0] = 7` assigns 7 to the first array element.

Suppose most of the estimates obtained in the simulations are in the range: (3.10, 3.20), with a few outside that range.

- Store the **count** of estimates that are less than 3.10 in array `loc 0`.
- Store estimates greater than 3.10 & less than 3.12 in array `loc.1` *etc.*
- Each simulation should have 10,000 dart throws.
- Run 10,000 such simulations. Print out the number of estimates of π in each range.

2. **Making a histogram**
   This exercise makes a histogram from the data gathered in the simulation above. There you ran 10,000 simulations of 10,000 dart throws each, and stored the estimates in an array. Use the data generated by your program to construct a histogram of the 10,000 estimates in your array:
   Use *Matplotlib* for making the histogram. To use *Matplotlib*:
   First `import matplotlib.pyplot as plt`, the command `plt.hist(x, nBins)` makes the histogram, where `x` is the array containing the data and `nBins` is a variable setting the number of bins.

   Use: `plt.xlabel(' π Estimates ')`
   `plt.ylabel(' Counts ')`
   `plt.title('Histogram of π estimates')`

   `https://matplotlib.org/api/_as_gen/matplotlib.pyplot.hist.html`
   The above documentation contains examples in a gallery at the bottom of the page.

   An alternate way to make a histogtram:
   (i) Create the individual bars using:
   `fig, bars = plt.subplots()`
   (ii) Let the count data (*i.e.* the number of values in each interval) be in an array y. Create the bins using:
   `x = np.arange(len(y))` (`np.arange(stop)` acts just like `range(stop)`.)
   `https://docs.scipy.org/doc/numpy/reference/generated/numpy.arange.html#numpy.arange`)
   (iii) Then `bars.bar(x,y)` makes the bar graph
   (iv) Use `bars.set_xticks(x)` to make marks on the x-axis
   (v) Set values against the ticks:
   `bars.set_xticklabels(("3.10", ..., "3.20"))`.

   For either method, `plt.show()` should be the final command.

3. Write a script to simulate the following experiment: Mollie and Chloe agree to meet at the coffee shop between 9:30 and 10:00 am. They are pretty casual about the logistics. Each one will show up at some time during the half hour interval. If Mollie arrives first, she will wait 5 minutes and leave if Chloe does not arrive by then. Similarly, if Chloe arrives first, she will wait 7 minutes for Mollie. Neither waits past 10:00am.

- What is the probability that they meet?
- How does the answer change if Chloe reduces her waiting time to 5 to match Mollie's?
- How does the answer change if Mollie increases her waiting time to 7 to match Chloe's?