

You have a set of exercises on Canvas that you can go through at your leisure that are designed to test your understanding of the theory in this part of the module.

These practicals will help you get familiar with the implementation details of stochastic processes and creating simulation.

This first week will look at generating random numbers and some of their properties.

You will need to somewhat familiar with programming (preferably python). If you have any questions please ask.

Creating a random number on a computer is difficult since computers use algorithms and ‘do what they’re told’. Random number generators have to be capable of generating a long list of random numbers, that are statistically acceptable, fast and reproducible (we use the phrase independent and identically distributed, *iid*).

One of the first algorithms used was based on ‘Residue Arithmetic’ i.e. operations on the remainder after division of two numbers. The general algorithm for creating random numbers in this method is

$$x_n = Cx_{n-1}(\text{mod}N)$$

It may be surprising that we can use a method as simple as this to generate random numbers. Lets look at a simple example with $C = 5$, $N = 32$ and $x_0 = 1$.

$$\begin{aligned} x_1 &= 5x_0(\text{mod}32) = (5)(1) - (0)(32) = 5 \\ x_2 &= 5x_1(\text{mod}32) = (5)(5) - (0)(32) = 25 \\ x_3 &= 5x_2(\text{mod}32) = (5)(25) - (3)(32) = 29 \\ x_4 &= 5x_3(\text{mod}32) = (5)(29) - (4)(32) = 17 \end{aligned}$$

In fact the first 12 numbers of this sequence are

$$5, 25, 29, 17, 21, 9, 13, 1, 5, 25, 29, 17$$

which is not particularly useful since the sequence repeats itself every nine numbers, which we say has a period of 9. This is not at all useful if we require more than 9 random numbers! Choosing good constants for a random number generator is a difficult task.

Task 1: Write a short python program that will generate a large number of random numbers using the formula $I_K = (aI_{K-1} + c)\text{mod } m$, where $a = 1277$, $c = 0$, $m = 131072$ and $I_0 = 1.0$.

It is important to be able to trust the random number generator (RNG) we are using. We want a RNG that produces random numbers with equal probability and we can check

it quite easily by plotting the distribution of the generated numbers in a histogram. Since the random number generator we have just created is a uniform one we should expect each bin to contain approximately the same quantity of random numbers.

Task 2: Using the code you wrote for task 1, plot the distribution of your random numbers. Once you have generated all your random numbers (you should choose a large enough sample, at least 5,000) plot the distribution as a histogram. What does the shape of the bar chart tell you? What if ran your program again, do you get the same output? why/why not? What if you have more/fewer bins?

The previous task is a good way of visualising random numbers. Another way is to plot them and look at the scatterplot.

Task 3: Using the code you wrote for task 1, create a scatterplot of pairs of drawn numbers. For example, $x = I_K, y = I_{K+1}$. what do you notice about the plot?

In python the function `random.random()` will generate a random number uniformly distribution in $[0, 1]$.

Task 4: Use the python RNG to regenerate the plot from task 3. What do you notice when you use a good (i.e. python's) RNG and a bad one (i.e. the one in task 1)? If you plot the distribution for python's function as you did in task 2, what did you get? Is this a surprise?

Task 5: Recreate task 4 but with the quasi-random number generator `random`. Do do this you will need to import `qmc` from `scipy.stats`. To create the random number generator use the command `sampler = qmc.Sobol(d=2)` and to generate and array of 2 random numbers use the command `sample=sampler.random()`.

Consider a ball rolling freely backwards and forwards along a track between two fixed points a and b . Let the length of the track be $b - a$. The probability density, $P(x)$, is interpreted as the chance of finding the ball between x and $x + dx$.

Our problem is to find the distribution function $P(x)$?

We begin by saying that for free travel the ball has an equal chance of being at any x , i.e. that $P(x) = \text{constant} = A$ since the ball lies on the track. So how do we now

calculate A ? We know that the ball must be somewhere in $[a, b]$ so

$$\begin{aligned} \int_a^b P(x)dx &= 1 \\ A \int_a^b dx &= 1 \\ => P(x) = A &= \frac{1}{b-a} \end{aligned}$$

Now we can get random x values that lie within $[a, b]$ from a computer generated random number r in $[0, 1]$ from

$$r = \int_a^x P(x)dx = \int_a^x \frac{dx}{b-a} = \frac{x-a}{b-a}$$

or

$$x = a + (b-a)r$$

We call this a mapping of r onto x so that we can easily convert $0 < r < 1$ onto any x range.

These examples should be obvious but this method works for more complicated problems.

Task 6: Write a short python program that will sample numbers from this distribution (you can pick any value you wish for a and b (but I'd avoid 0 and 1). You should plot this distribution. Does it look correct?

Task 6.1 [Optional]: Pick a distribution you'd like to sample from (e.g. $\cos(\theta)$), derive an expression for x and create a short python program to sample from this distribution. Does it look correct?