6 February 2020

Note : Please get your own laptops to the class, pre-installed with compilers/interpreters corresponding to your coding language.

Question 1: Using a coin toss process to create an Uniform Random Number Generator

Assume that you have an unbiased coin with two outcomes. Use this coin to create a Uniform Random Number (URN) generator. Simulate the process using your favourite coding language. Note : Almost all the scripting and statistical languages have a command to simulate a coin toss. For example in Python:

```
import numpy as np
np.random.choice(['H','T'])
np.random.choice([0,1])
```

Solution:
Let χ_i represent the coin toss outcome for the i^{th} trial which is either 1 or 0. Then the uniform random number(U) is given by

$$\boxed{U = \sum_{i=1}^{N} \chi_i \left(\frac{1}{2}\right)^i}$$

Python code

```
#  Using coin toss process to generate uniform random numbers(URN)
#   Instructor : Dr. Anand Srivastava
#  Assisted by: krishnakanth and Kirtika
#  Theoretical Biophysics Laboratory, Molecular Biophysics Unit,
#  Indian Institute of Science, Bangalore - 560012
#
#  Last Modified:
#  For any queries regarding the usage/issues with the following code
#  please feel free to contact me at krishnakanth.baratam@gmail.com
#  Load the required packages

import numpy as np
import matplotlib.pylab as plt

import numpy as np

# Function to generate uniform random numbers
def munif(a):
   rvec=[]
   for j in range(a):
      r = 0
      for i in range(1,20):
         r = r+np.random.choice([0,1])*((1/2)**i)
      rvec.append(r)
   return(rvec)
```

```
munif(5) # Generates 5 uniform random numbers
```

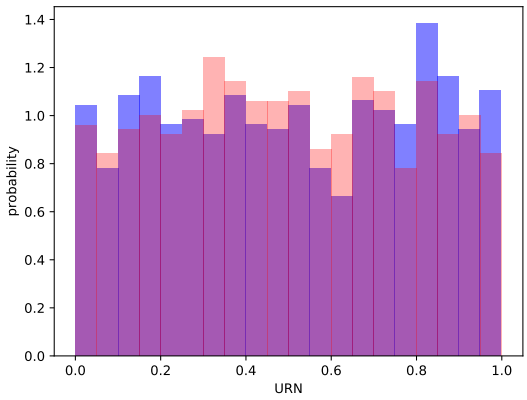
Question 2: Uniform Distribution

Generate atleast 1000 URNs and prove that the numbers indeed obey a uniform distribution. Use the URN generator in your coding language and check the distribution obtained from it with that of yours.

Solution: This is simply histogramming of URNs generated by the function created by us in the above question and the compiler's/interpreter's internal URNs

Python code

```
#  Check the distribution of uniform random numbers generated using
#  your own uniform random number generator

# import required packages
import numpy as np
import matplotlib.pylab as plt

# Function to generate uniform random numbers
def munif(a):
   rvec=[]
   for j in range(a):
      r = 0
      for i in range(1,20):
         r = r+np.random.choice([0,1])*((1/2)**i)
      rvec.append(r)
   return(rvec)

# Generated from our URN generator
my_urns = munif(1000)

# Python generated uniform random numbers
python_urns = np.random.uniform(0,1,1000)

num_bins = 20
n, bins, patches = plt.hist(my_urns, num_bins, facecolor='blue', alpha=0.5, density =
    True)
n, bins, patches = plt.hist(python_urns, num_bins, facecolor='red', alpha=0.3, density =
    True)
plt.ylabel("probability")
plt.xlabel("URN")
plt.savefig("urn_distributions.pdf")
plt.close()
```



Question 3: Calculating π value using Monte-Carlo scheme

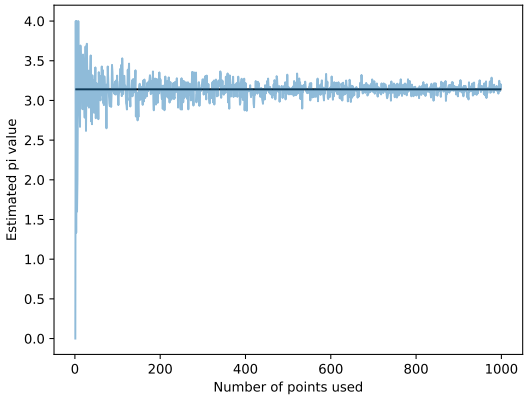
Using Monte-Carlo scheme discussed in the class and your URN generator, estimate the value of π.
a) Plot the estimate obtained with increasing number of random points used in the scheme.
b) Plot the error observed (assuming the real value of π to be 3.14159) with increasing number of random points used in the scheme. Overlay the plot $1/\sqrt{N}$ on the obtained plot.

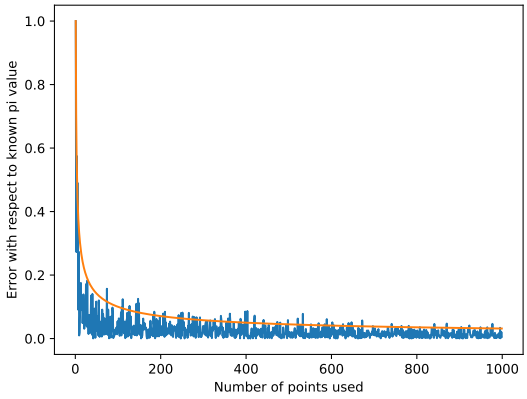
Solution:

Python code

```
#  Using uniform random number(URN) generator to calculate the value of pi

# import required modules
import numpy as np
import matplotlib.pylab as plt



def munif(a):
   rvec=[]
   for j in range(a):
      r = 0
      for i in range(1,20):
         r = r+np.random.choice([0,1])*((1/2)**i)
      rvec.append(r)
   return(rvec)


# pi_cal is a function which takes the number of random numbers
# required to be used to estimate pi value
def pi_cal(b):
   counter=0
   capture = []
   x = munif(b)
   y = munif(b)
```

```
    for i in range(b):
        if ((x[i]**2)+(y[i]**2))<=1 :
            counter = counter+1
    capture.append(4*(counter/b))
    return(capture)

# Part a
pi_vals = np.array([pi_cal(item) for item in range(1,1000)])
plt.plot(range(1,1000),pi_vals,alpha =0.5)
plt.xlabel("Number of points used")
plt.ylabel("Estimated pi value")
plt.hlines(3.14,1,1000)
plt.savefig("pi_value_estimate.pdf")
plt.close()

# Part b
plt.plot(range(1,1000),np.sqrt((np.array(pi_vals)-np.pi)**2)/np.pi)
plt.plot(range(1,1000),1/np.sqrt(np.array(range(1,1000))))
plt.xlabel("Number of points used")
plt.ylabel("Error with respect to known pi value")
plt.savefig("pi_stde.pdf")
plt.close()
```





Question 4: Sampling from a different distribution

Transform the URN generator to sample random numbers from the following probability distribution in the interval $[0, \infty)$

$$P(x) = exp(-x)$$

Generate atleast 1000 such random numbers and show that they indeed obey the above distribution.

Solution: Inversion sampling

Let the URN be ζ in the interval [0,1).

$$cdf(x) = \int_0^x e^{-t}dt = 1 - e^{-x}$$

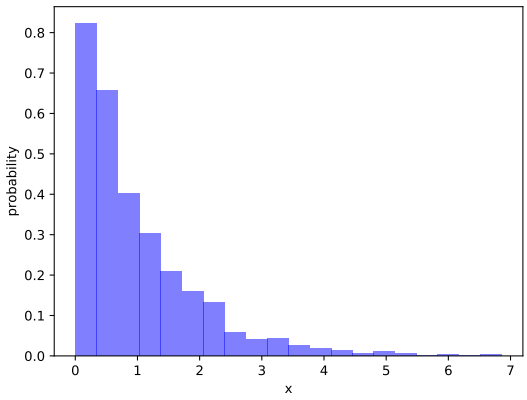
The range of cumulative distribution function also lies between [0,1). So, equating ζ and $cdf(x)$ we get

$$\zeta = 1 - e^{-x}$$

$$\boxed{x = ln\Big(\frac{1}{1-\zeta}\Big)}$$

Python code

```
#  Transform a random number sampled from uniform distribution
#  to a number sampled from exponentially decreasing probability distribution function

# import required packages
import numpy as np
import matplotlib.pylab as plt

# Function to generate uniform random numbers
def unif_to_exp(a):
   rvec=[]
   for j in range(a):
      r = 0
      for i in range(1,20):
         r = r+np.random.choice([0,1])*((1/2)**i)
      rvec.append(np.log(1/r)) # transforms URN to exp distribution
   return(rvec)

# Generated from our designed exp generator
my_urns = unif_to_exp(1000)



# Plotting routines
num_bins = 20
n, bins, patches = plt.hist(my_urns, num_bins, facecolor='blue', alpha=0.5, density =
    True)
plt.ylabel("probability")
plt.xlabel("x")
plt.savefig("exp_distribution.pdf")
plt.close()
```



Question 5: A Boy and His Atom

In 2013, IBM Research division released a 1.5 minute stop-motion animated short film depicting the story of a boy and a wayward atom who meet and become friends. The individual frames of the movie were created by carefully placing carbon monoxide molecules on a copper surface and then visualizing them under a scanning tunneling microscope.

More details of the experiment: here
Check it out: www.youtube.com/watch?v=oSCX78-8-q0

Consider the surface of copper to be composed of N sites and the chamber filled with M CO molecules where N>>M.

Calculate the possible number of configurations possible in each of the following scenarios:
This question does not require coding.

Solution:
Case 1:
a) Each site can accommodate only one molecule of CO.

$$\boxed{\Omega = \frac{N!}{M!(N-M)!}}$$

b) Each site can accommodate more than one molecule of CO.

$$\boxed{\Omega = \frac{(N+M-1)!}{M!(N-1)!}}$$

Case 2: Now fill the chamber with M different gas molecules each having different chemical identities
a) Each site can accommodate only one molecule.

$$\boxed{\Omega = \frac{N!}{(N-M)!}}$$

b) Each site can accommodate more than one molecule .

$$\boxed{\Omega = N^M}$$

Question 6: Making a Transition Matrix

A researcher performed a single molecule FRET experiment on protein-211 and noticed significant jumps in FRET efficiencies with time. He got interested and performed an all-atom molecular simulation to look at the conformational changes. From his long trajectory of simulation he noticed that the protein takes 2 states (A and B) and all the conformations fall into either of these 2 states. From the trajectory, he made a long sequence of the changes observed in the simulation which can be found here. Help him to make a transition matrix.

Solution:
This question is simply solved by counting the number of transitions from $A -- > A$, $A -- > B$, $B -- > B$ and $B -- > A$.

$$= \begin{bmatrix} P(A|A) & P(B|A) \\ P(A|B) & P(B|B) \end{bmatrix}$$

$$= \begin{bmatrix} \frac{52}{(52+214)} = 0.195 & \frac{214}{(52+214)} = 0.805 \\ \frac{214}{(214+301)} = 0.416 & \frac{301}{(214+301)} = 0.584 \end{bmatrix}$$

Python code

```
#  Create a transition state matrix using the given data
#
#  Load the required packages

import numpy as np
import matplotlib.pylab as plt

import numpy as np

# Read the file
my_seq = open("seq.txt").readlines()

# Count and print each type of transition
print("Number of A to A transitions : "+ str(str.count(k[0],"AA")))
print("Number of A to B transitions : "+ str(str.count(k[0],"AB")))
print("Number of B to A transitions : "+ str(str.count(k[0],"BA")))
print("Number of B to B transitions : "+ str(str.count(k[0],"BB")))
```