IS 604 – Homework #3

Include the programming code (R, MATLAB, or Python) you used for each problem.

1. Starting with $X_0 = 1$, write down the entire cycle for $X_i = 11X_{i-1} \bmod (16)$.

2. Using the LCG provided below: $X_i = (X_{i-1} + 12) \bmod (13)$, plot the pairs (U_1, U_2), (U_2, U_3),…, and observe the lattice structure obtained. Discuss what you observed.

3. Implement the pseudo-random number generator:

$$X_i = 16807X_{i-1} \bmod (2^{31} - 1)$$

Using the seed $X_0 = 1234567$, run the generator for 100,000 observations. Perform a chi-square goodness-of-fit test on the resulting PRN's. Use 20 equal-probability intervals and level $\alpha = 0.05$. Now perform a runs up-and-down test with $\alpha = 0.05$ on the observations to see if they are independent.

4. Give inverse-transforms, composition, and acceptance-rejection algorithms for generating from the following density:

$$f(x) = \begin{cases} \frac{3x^2}{2}, -1 \le x \le 1 \\ 0, \quad otherwise \end{cases}$$

5. Implement, test, and compare different methods to generate from a $N(0,1)$ distribution.

a) The simplest generator is the inverse transform method. Create a function `normrandit` that:

1. Takes no input variables.
2. Generates a uniform random number $U \sim U(0,1)$.
3. Returns one output variable: $X = F^{-1}(U)$, where $F^{-1}$is the inverse normal CDF.

Also create a function `itstats` that:

1. Takes one input variable N.
2. Generates N samples from a $N(0,1)$ distribution using `normrandit`.
3. Returns two output variables: the mean and the standard deviation of the samples.

b) Next up, we want to generate samples using the Box-Müller algorithm. Create a function `normrandbm that:`

1. Takes no input variables.
2. Generates two uniform random numbers $U_1, U_2 \sim U(0,1)$.
3. Returns two output variables: $X = (-2\ln(U_1))^{\frac{1}{2}} \cos(2\pi U_2)$ and

$$Y = (-2\ln(U_1))^{\frac{1}{2}} \sin(2\pi U_2).$$

As in a), create a function `bmstats` that can produce N samples using `normrandbm` and return their mean and the standard deviation.

c) Lastly, we want to generate samples using the accept-reject approach. Create a function `normrandar` that:

1. Takes no input variable.
2. Generate two uniform random numbers $U_1, U_2 \sim U(0,1)$.
3. Convert the samples to $Exp(1)$ by calculating $X, Y = -\ln(U_i)$.
4. Accept the sample if $Y \geq \frac{(X-1)^2}{2}$ and reject otherwise.
5. If sample is accepted, randomly choose sign, and return.
6. If sample is rejected, return to 2. and try again.

As in a), create a function `arstats` that produces N samples using `normrandar` and returns their mean and standard deviation.

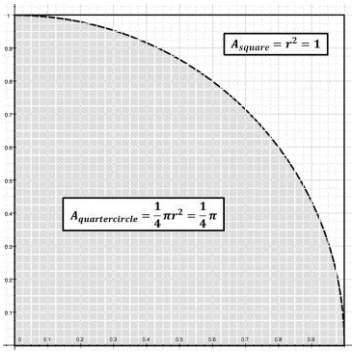
d) We now compare and evaluate the approaches you implemented in parts a) b) and c). Run 10 iterations of `itstats`, `bmstats`, and `arstats` for N = 100, 1000, 10000, and 100000, and calculate the average means and standard deviations produced by each method at each value of *N*.

In addition, measure the exact CPU time required for each iteration, and calculate the average time required for each method at each value of *N*.

For each method, plot the average means and standard deviations against the sample size. Which of the three methods appears to be the most accurate? Also, plot the average CPU time vs. *N*. Which of the three methods appears to take the least time?

e) Plot histograms of 1000000 samples from each method and compare.

6. Use Monte Carlo integration to estimate the value of π. To summarize the approach, consider the unit quarter circle illustrated in the figure below:



Generate N pairs of uniform random numbers (x, y), where $x \sim U(0,1)$ and $y \sim U(0,1)$, and each (x, y) pair represents a point in the unit square. To obtain an estimate of π, count the fraction of points that fall inside the unit quarter circle and multiply by 4. Note that the fraction of points that fall inside the quarter circle should tend to the ratio between the area of the unit quarter circle (i.e., ¼ π) as compared to area of the unit square (i.e., 1). We proceed step-by-step:

a) Create a function `insidecircle` that takes two inputs between 0 and 1 and returns 1 if these points fall within the unit circle.

b) Create a function `estimatepi` that takes a single input N, generates N pairs of uniform random numbers and uses `insidecircle` to produce an estimate of π as described above. In addition to the estimate of π, `estimatepi` should also return the standard error of this estimate, and a 95% confidence interval for the estimate.

c) Use `estimatepi` to estimate π for N = 1000 to 10000 in increments of 500 and record the estimate, its standard error and the upper and lower bounds of the 95% CI. How large must N be in order to ensure that your estimate of π is within 0.1 of the true value?

d) Using the value of N you determined in part c), run `estimatepi` 500 times and collect 500 different estimates of π. Produce a histogram of the estimates and note the shape of this distribution. Calculate the standard deviation of the estimates – does it match the standard error you obtained in part c)? What percentage of the estimates lies within the 95% CI you obtained in part c)?