Final Project ISYE 6644 – Option 13

Michael Good

Write-up

For my final project, I chose to create a library of random variate generators from PRNs using Python. My method was to primarily use Inverse Transform Method with PRNs to generate the appropriate distribution. When Inverse Transform was trickier or not possible, I used other tricks as discussed in class, and a special trick for the Gamma Distribution.

I first used the Desert Island generator discussed in class to find lists of PRNs. Python has its own built-in generator, but I figured I should make my own for this project. That became the basis for the rest of the methods.

Bernoulli

Bernoulli is very easy! If the PRN is less than 1-p, the trial is a failure (denoted by 0), otherwise the trial is a success.

Geometric

Here, I used the Inverse Transform formula derived in class to transform a list of PRNs generated by my previous method to geometric distribution. The resulting list is geometric random variates as long as the user specifies.

Exponential

Same thing for Exponential! I used my PRN generator with the inverse transform formula derived in class.

Normal

For normal, I decided to code in the BCNN approximation discussed in class to transform PRNs from my generator into random variates of a normal distribution.

Gamma

This was a tricky one. I was having trouble figuring out how to generate gamma random variates from a PRN, so I did some research and found a method from https://www.cs.toronto.edu/~radford/csc2541.F04/gamma.html. I then coded this method into python and output a list of gamma random variates.

Weibull

Like some previous examples, I used the Weibull inverse transform formula derived in class.

Poisson

For Poisson, I stepped through the CDF calculations to determine if the condition was satisfied. We start with 0, then add the pdf of each x value until our CDF is greater than our PRN. At that point, we can append our list of random variates with the appropriate x value. Note that I also gave the user the option to enter a custom amount of time for the Poisson distribution.

Triangular

Again, this one was easy. Specify two different seeds, and add two uniform random variates from two PRN lists using python's zip function.

Negative Binomial

Back to Geometric – for this one I just added geometric random variate outputs together, taking care to change the seed for each calculation so that it didn't output the same thing for each run.

For each method, I used seaborn's distplot function to plot the distribution of 50000 random variates to verify the shape. Everything looks good! You can see these charts in "Random Variate Generation Methods.html". There is also raw python code in the "Python Files" folder, and ipynb files in the "Python Notebook Files" folder.

I have also included the python file I imported to my "Example Questions" script. This is the same as "Random Variate Generation Methods", just without commenting.

Finally, I've included some sample questions, one for each routine in the project.

User Guide

For all functions, use import functions in python. Make sure that the python file is in the same folder as your workspace.

prn_generator(num_vars=50000,seed=52545)

Generates list of PRNs using desert island approach.

Parameters:

- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

bern_generator(p,num_vars=50000,seed=52545)

Generates list of Bernoulli-distributed PRNs. Success or failure of a single event. An output of 1 indicates a success while an output of 0 indicates a failure.

Parameters:

- p: probability of success
- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

geom_generator(p,num_vars=50000,seed=52545)

Generates list of geometric-distributed PRNs. This gives the number of Bernoulli trials until a success occurs.

Parameters:

- p: probability of success
- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

negbin_generator(n,p,num_vars=50000,seed=52545)

Generates list of negative binomial-distributed PRNs. This gives the number of successes before the nth failure occurs.

- n: the nth failure
- p: probability of success

- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

poisson_generator(lam,time_int=1,num_vars=50000,seed=52545)

Generates probability of an arrival within the time interval

Parameters:

- lam: lambda rate (expected number of events in the interval)
- time_int: time interval (default is 1)
- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

exp_generator(lam,num_vars=50000,seed=52545)

Generates interarrival times from a Poisson process with rate lam=lambda.

Parameters:

- lam = lambda rate (expected number of events in the interval)
- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

gamma_generator(alpha,beta,num_vars=50000,seed=52545)

Generates gamma random variates from PRNs. Often used in engineering for skewed distributions.

Parameters:

- alpha: alpha parameter
- beta: beta parameter
- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

weibull_generator(alpha,beta,num_vars=50000,seed=52545)

Generates Weibull random variates from PRNs. Often used for life data analysis, like product life. Always positive because failures cannot occur before time 0.

Parameters:

- alpha: alpha parameter
- beta: beta parameter

- num_vars: number of random variates to generate
- seed: custom seed for start of random variate generation

triangular_generator(num_vars,seed1=52545,seed2=3079)

Generates triangular random variates by adding two uniforms. Can be transformed by multiplying the results or adding to shift the mean. Useful for estimating a distribution when it is not known, but the range and mean are known.

Parameters:

- num_vars: number of random variates to generate
- seed1: custom seed for start of first random variate generation
- seed2: custom seed for start of second random variate generation