Problem Set 2.2
MPCS 58020
Spring 2021

Due Date: Monday April 26, 2020 @ 6:00 am

Logistics and Submission – 5 pts

- You may use any programming language, but you may not use any high-level functionality specific to the homework problem. In particular, you may use basic linear algebra routines, but not specialized statistics libraries or black-box solvers (e.g. regression toolkit). You may use a uniform random number generator on $(0, 1)$ and on integers within a range, but you may not use built-in generators for more complicated distributions. If you are unsure if a feature is allowed, ask for clarification on Slack.

  Specific to homework 2: do not use built-in functions for computing the mean and variance. You can use built-in functions for summing arrays, however. For example, in Python NumPy, `np.mean()` and `np.var()` are forbidden but `np.sum()` is allowed.

- For the programming/simulation problems, your program should support running with no arguments and use enough realizations (trials) to get a good estimate of the result with a reasonably low execution time (less than 5 seconds is preferable). A command line program that prints the results is preferred, but a main script (e.g. in MATLAB) is also fine.

- All written responses, plots, and required program outputs should be included in (ideally) a **single** writeup file named `writeup.pdf`. This could be rendered from a Jupyter notebook, LaTeXsource, and/or scanned pages. At most 2x PDFs are allowed, if you are having trouble merging the rendered Jupyter notebook and a scanned document, for example.

- For non-programming homework problems involving derivations, you may use your favorite symbolic solver (Mathematica, Wolfram Alpha, MATLAB, SymPy, Maple, SageMath) to evaluate integrals.

- You must also include a plain-text `README` file with your name, assignment number, list of any references used (if any), a clear explanation of how to run your simulations, and what arguments (if any) each of the scripts takes.

- If you use a compiled language (e.g., C++), you must provide a `Makefile`.

- All source code and scripts should be labelled according to the problem number. For instance, code required for problem 6 should be labelled as `p6_d[.py, .jl, .m, ...]` and `p6_e[.py, .jl, .m, ...]` (if the parts are contained in separate files). If you have more than one source file for a given problem and part, give them more descriptive labels so we know where to start grading and what is contained in them.

- *Aside on Jupyter notebooks:* Jupyter notebooks are an increasingly popular, open-source, interactive web-based medium for computational work. Code, plots, and rich text can all live side-by-side in a single notebook. All or part of your assignment can be written in and submitted as a single Jupyter notebook. You can use any of the 100+ kernels supported Jupyter, but let us know ahead of time if you are going to use a more exotic kernel than the core three: IPython, IRkernel, IJulia. Still include a minimalist `README` if you submit a notebook containing all the code and instructions.

- Your programs should be human readable, which means it they should be well-commented, well-structured, and easy for the grader to understand. Messy and/or poorly commented code that is unreasonably difficult to follow may receive deductions even if it is logically correct.
- Your submission will include:
    1. PDF writeup
    2. README
    3. source code files and/or Jupyter notebook
    4. If necessary, a Makefile.
- The total submission should be less than 20 MB in size. Each file should be no larger than a few MB (preferably/typically much smaller).

General comments:

Many of these prompts (especially the programming exercises) are written in a way to encourage exploratory analysis. Since an objective of this course is to to help you become a practitioner of these methods for modeling real-world, stochastic phenomena, some aspects of the solutions may be open-ended.

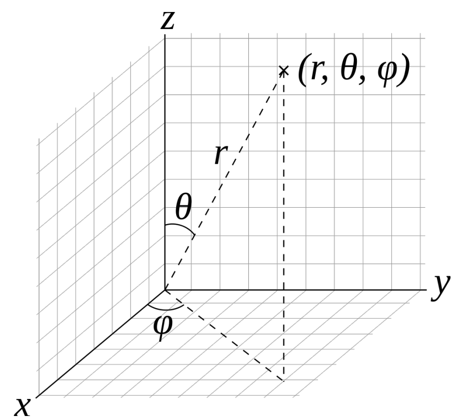
Please put some thought into making clear, high-quality visualizations when including them in a solution. Even when a plot is not explicitly required in the prompt, it might be a helpful addition. Always label your axes, use a figure legend when multiple datasets are plotted, and try to avoid cluttered plots.

A few sentences describing and interpreting results displayed in the figures and/or function output are always appreciated. Sometimes the interpretations and conclusions are not clear-cut, and your writeup can reflect that.

Problems

1. (10 points) We want to sample a random 3D direction. Any direction we sample should be equally probable. Another way of thinking about this is is to say that if we sampled the direction for a vector whose origin is at the center of a sphere, there should be an equal probability of the vector intersecting with any given location on the sphere's surface.

   We can represent the direction vector in spherical coordinates where $\theta \in [0, \pi]$ is the polar angle and $\varphi \in [0, 2\pi]$ is the azimuthal angle:

   The joint PDF of a random direction intersecting with the sphere at a given point on the sphere's surface is:

   $$f(\theta, \varphi) = \frac{1}{4\pi} \sin \theta$$

   (a) Find the marginal PDFs $f_\varphi(\varphi)$ and $f_\theta(\theta)$.

   (b) Find the cumulative distribution functions $F_\varphi(\varphi)$ and $F_\theta(\theta)$.

   (c) Use the inverse transform method to create a scheme to sample each of the two angles φ and θ with a uniform random distribution U.

   (d) Write a program that uses your sampling technique to sample directions randomly. Your program should create a visualization of where your samples intersect the unit sphere. Plot enough points so that it is clear that your sampling scheme is working correctly. To convert spherical coordinates to a Cartesian intersection location $\vec{V}$ with the unit sphere, you can use the relations:

   $$\begin{aligned} v_x &= \sin\theta \cos\phi \\ v_y &= \sin\theta \sin\phi \\ v_z &= \cos\theta \end{aligned}$$

   (e) Repeat part (d), but use a naive sampling scheme where θ and φ are sampled directly from a uniform distribution without transform such that $\theta = U(0, \pi)$ and $\varphi = U(0, 2\pi)$. Is the naive distribution biased?

2. (10 points) ***Random permutations*** *can be generated using a variation of the discrete inverse transform method, as described in Ross, Example 4b.*

   A deck of 50 cards are labeled with the numbers 1,2,. . . ,50. The cards are shuffled and then turned over one card at a time. Say that a "hit" occurs whenever card labeled when the ith card to be turned over is labeled with the number i. Let the random variable X be the total number of hits after all cards have been turned over.

(a) Without a simulation, derive the expected value of X.

(b) Compose and run a simulation to estimate the expected value of X.

3. (10 points) ***Exponential random variables*** *have the PDF* $f(x) = \lambda e^{-\lambda x}$ *and the CDF* $F(x) = 1 - e^{-\lambda x}\, dx$ *over the interval* $(0, \infty)$.

   In many applications, the exponential distribution can describe a continuous quantity that may take on any positive value, but for which larger values are increasingly unlikely. For example, the time it takes for a radioactive particle to to decay is an exponential random variable. Ross, Example 5b, describes how to use the inverse transform method to simulate exponential random variables.

   A casualty insurance company has 1000 policyholders, each of whom will independently present a claim in the next month with probability 0.05. Assuming that the amounts of the claims made are independent exponential random variables with mean \$800, use a simulation to estimate the probability that the sum of these claims will exceed \$50,000.

4. (10 points) Consider the following PDF defined within the boundaries $0 < x < 5$ (except for part b):

$$f(x) = \sqrt{\frac{2}{\pi}} x^2 e^{-x^2/2}$$

   (a) Use the rejection method to generate random numbers from $f(x)$ by generating random numbers from:

$$g(x) = 0.2 \quad 0 < x < 5$$

   using a uniform random number generator.

   (b) Use the rejection method to generate random numbers from $f(x)$ by generating random numbers from:

$$h(x) = \frac{\pi}{10} \sin \frac{\pi x}{5} \quad 0 < x \leq 4.999$$

   You will need to analytically find the CDF for $h(x)$ and then find its inverse so as to allow for sampling of $h(x)$ directly with a uniform random number generator. Since $\lim_{x \to 5} h(x) = 0$, we cannot use rejection sampling with $h(x)$ to approximate $f(x)$ over the entire open interval < 5.

   (c) Use the rejection method to generate random numbers from $f(x)$ by generating random numbers from the normal distribution:

$$j(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad 0 < x < 5$$

   where $\mu = \sqrt{2}$ and $\sigma = 2^{-1/4}$. Instead of analytically computing the CDF and inverse transform of $j(x)$, implement the Box-Muller algorithm to numerically sample directly from the given normal distribution, rejecting any samples outside of the range $0 < x < 5$. Assume that $f(x)/j(x)$ is maximal at $x = \sqrt{2}$.

   (d) Create histogram plots of your distributions generated by all three sampling methods to show they are generating the same (correct) distribution for $f(x)$.

   (e) Compare the efficiency (sampling iterations per random number accepted; and associated variance) of the three implementations ($g(x)$, $h(x)$, and $j(x)$).

   Overall, your script should test all three methods for a reasonably high number of samples, report their efficiencies, and display or output to file their sampling histograms. Include in your writeup their efficiencies and the histogram plots.