Homework #1—DSC 520

Assigned: Wednesday, October 11, 2023
Due: Wednesday, October 25, 2023 (midnight)

Please use C to solve these problems. Other languages (Python, Matlab, Mathematica, gnuplot, etc) should be used only to visualize or analyze the data (ie making plots).

Report: Put all the figures and answers asked for in all the questions into a single well organized PDF document (prepared using LaTeX, a Jupyter notebook, or something else) and call it "Report.pdf". This PDF report should be just a few pages (aim for less than 4 pages).

Code: Separately, include all code used such as the C/C++ code to solve the problem, any code used to make figures or carry out analysis, and any Unix scripts. Please strive towards automation where there is one wrapper script that solves the entire problem: it will compile the code, run the code, and generate any figures or plots.

Submission: Please upload your work (Report.pdf and the codes) to Bitbucket, in a directory called "HW1", following the procedure you have been using in the labs. Be sure to do this by the due date. **For Homeworks, late submissions without prior permission is not allowed, but you get more time than the labs.**

Problem 1: (*20 points*)

Background: In class, we discussed an algorithm to numerically compute an integral. An alternative (non-deterministic) way to compute an approximate value for an integral is the Monte Carlo method: Let $f(x)$ be a real-valued function, where x is a multidimensional vector taking values in some region $R \subset \mathbb{R}^d$. Now let X_i for $i = 1, 2, \ldots$ be independent and identically distributed (iid) random variables. The random variables X_i are uniform random variables taking values in R. Then the integral

$$I = \int_R f(x)\,\mathrm{d}x\,,$$

can be approximated by

$$\hat{I}_N = V \frac{1}{N} \sum_{i=1}^{N} f(X_i)\,, \tag{1}$$

where

$$V = \int_R \mathrm{d}x\,,$$

is the volume of the region defined by R. Furthermore, in the large-N limit

$$\int_R f(x)\,\mathrm{d}x = \lim_{N\to\infty} V \frac{1}{N} \sum_{i=1}^{N} f(X_i)\,.$$

The theoretical framework for Monte Carlo integration relies on key results and concepts from probability theory, and is beyond the scope of this course. These details can be found in any standard textbook on probability or from numerous online sources, for example, http://www.math.chalmers.se/Stat/Grundutb/CTH/tms150/1112/MC.pdf.

The first problem should make this otherwise general and somewhat abstract introduction to Monte Carlo integration a bit clearer. We will work on 1D for now, and generalize to higher-dimensional integrals later in the semester.

Recall that in a recent computing lab we numerically computed

$$I_1 = \int_{-1}^{1} \frac{1}{1+x^2} dx\,, \tag{2}$$

using the trapezoidal rule. The analytic result is $I_1 = \pi/2$. The general Monte Carlo integration formula (1) applied to problem (2) is

$$I_1 \approx \hat{I}_1(N) = \frac{V}{N}\sum_{i=1}^{N} \frac{1}{1+x_i^2}\,, \tag{3}$$

where $V = 2$ and x_i are drawn from a uniform distribution on the interval $[-1, 1]$.

(a) (1 points) Write the mathematical formula (3) in your report using LaTeX. Your formula should look just like mine. For full credit, all of the mathematical symbols must be correctly displayed.

(b) (2 points) Write pseudocode which turns the formula you found in part a into a numerical algorithm, but just in words. That is, write out the algorithmic structure that you will later follow in order to implement it with C. How will you implement the summation? What is the value of V? When do the random numbers get generated and regenerated? These are some of the algorithmic design questions your pseudocode should address. A good pseudocode should be detailed enough that if you gave it to a friend they would be able to use it to write the program. We discussed an example pseudocode in class, for Lab4.

(c) (7 points) Implement your algorithm in C. Your program should accept N as input and output $\hat{I}_1(N)$ from (3), computed by the Monte Carlo method. You will need to find out how to generate uniform random variables in C.

(d) (4 points) Plot N vs absolute error, $E(N) = \left|I_1 - \hat{I}_1(N)\right|$, on a log-log plot. Use the same sequence of $N = 2^i$ for integers $i = 1, 2, 3, \ldots, 30$ that you used in lab 4. Notice that due to the random variables used to compute the integral, both $\hat{I}_1(N)$ and $E(N)$ will be random variables. And so the error plot will look "noisy".

(e) (2 points) Rerun the numerical experiment in part d 10 times; the running of these trials should be fully automated using a Unix script. For full credit, a Unix script must be used.

(f) (2 points) Compute the mean of the error $\langle E(N)\rangle$ over 10 trials like in part e, for each N and plot it. Here, $\langle E(2)\rangle$ means the average of $E(N)$ over the 10 trials, for N=2. Your job is to compute $\langle E(N)\rangle$ for the sequence of N given in part d, and plot it as a function of N on a log-log plot. You should find the error behaves like $\log\langle E(N)\rangle = B + A \times \log N$ for some constants A and B. Find a reasonable estimate for A. For comparison, add to your plot the analogous N vs error data generated with the trapezoidal rule. You should find that the trapezoidal rule's error decreases more quickly, indicating it is a higher-order method.

(g) (2 points) Time your program's runtime using the Unix program **time** (or another reliable method. If you do not use the Unix program **time**, say what you did and why it can be trusted). Plot the program's runtime vs N. Is this trend to be expected? From your data, estimate the time it would take to run your program for $N = 2^{32}$.

Good figures often contain many pieces of data. For example, in this case you can plot the results from parts d, e, and f on the same figure. Good science requires the results to be reproducible. Either provide a script to regenerate your data (best option; see above), or otherwise, make sure to document how to run the program to reproduce your figures.

Problem 2: (*5 points*)

Let's use your new code to compute a more challenging integral. Suppose you trained a probabilistic model to predict the amount of snow, measured in meters, that will fall in Dartmouth, MA over January 2024. Let the amount of snow be denoted by S, which we will take to be a continuous random variable. Let the probability density function of S be given by (this is your model!)

$$P(S) = \frac{1}{2.4496028273129427}\ \mathrm{e}^{-|S-2|^2/2} \qquad \text{if } 0 \le S \le 10 \tag{4}$$

otherwise $P(S) = 0$ (there is no chance of negative snowfall, for example). This is a correctly normalized probability density function because (you can check this with your code)

$$\int_0^{10} P(S)\, dS = 1\,. \tag{5}$$

Using your Monte Carlo integration code, find the expected average value of snowfall in January by computing

$$\int_0^{10} S\, P(S)\, dS = ???\,. \tag{6}$$

What value do you find? Provide convincing evidence that someone should trust your value (Note: this is a made-up example. The historical value of expected snowfall may not match what you find. So, just see if you can come up with a justification).

Hint: Notice that this looks just like (2), with S as the name of the random variable instead of x, and a different function $f(S) = SP(S)$ as the integrand. So, you need to figure out the corresponding approximation for (6), just like (3) was an approximation for (2).