Homework #2—DSC 520

Assigned: Monday, October 23, 2023
Due: Friday, November 3, 2023 (midnight)

Reading & viewing: If you would like additional experience with C programming, please consider viewing all or part of https://www.youtube.com/watch?v=-CpG3oATGIs and consulting http://www.cprogramming.com/tutorial/c-tutorial.html

You must use C/C++, Fortran, or Julia to solve these problems. Other languages (Python, Matlab, Mathematica, gnuplot, etc) should be used only to visualize 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). This PDF report should be just a few pages (aim for less than 4 pages). **Please print out and turn in your report before midnight by the due date**. Your report should also be uploaded to Bitbucket.

Code: On every homework you turn in, be sure to upload the computer code you used to generate your solutions to Bitbucket. 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 script that solves the entire problem: it will compile the code, run the code, and generate any figures or plots.

Monte Carlo integration: Please consult homework 1 for general background on Monte Carlo integration.

Problem 1: (*20 points*)

In this problem, you will use Monte Carlo integration to compute π. In order to calculate the value of π, consider the following setup. The first thing to call attention to is we will be doing a two-dimensional integral. We will let x_1 and x_2 be the coordinates of each dimension. And let $\vec{x} = (x_1, x_2)$.

Consider a two-dimensional unit disc $D = \left\{\vec{x} \in \mathbb{R}^2 \mid x_1^2 + x_2^2 \leq 1\right\}$ and let R be the circumscribing square: $R = \left\{\vec{x} \in \mathbb{R}^2 \mid |x_1| \leq 1 \text{ and } |x_2| \leq 1\right\}$. Let $f(\vec{x}) \equiv \mathbb{1}_D(\vec{x})$ be the indicator function on D:

$$\mathbb{1}_D(\vec{x}) = \begin{cases} 1, & \vec{x} \in D, \\ 0, & \vec{x} \notin D. \end{cases}$$

Then

$$\int_{-1}^{1}\int_{-1}^{1} f(x_1, x_2) dx_1 dx_2 = \int_R \mathbb{1}_D(x_1, x_2) dx_1 dx_2 = \int_D 1 dx_1 dx_2 = \text{area}(D) = \pi\,. \tag{1}$$

Thus, if we take (x_1, x_2) as random variables uniformly distributed on the square R, the Monte Carlo integration formula (from homework 1) can be used to compute π:

$$\pi = 4 \lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_D(x_1^i, x_2^i)\,, \tag{2}$$

where we have used the fact that

$$V = \int_R \mathrm{d}x = 4\,, \tag{3}$$

since the integration region R is a square whose edge length is 2. When taking $N < \infty$ the approximation to π is

$$\pi_N = 4\frac{1}{N} \sum_{i=1}^{N} \mathbb{1}_D(X_i) \tag{4}$$

$$. \tag{5}$$

To help you get started on this problem, here's Python code for picking (x_1, x_2) from a uniform distribution on R. This code explicitly uses a for-loop to show how to make the i^{th} pick.

```
import numpy as np
import matplotlib.pyplot as plt

N=10000 # number of picks to make from the unit square
for i in range(N):
  # pick the i-th point
  x1 = 2*np.random.rand() - 1.0
  x2 = 2*np.random.rand() - 1.0
  # add this point to the figure
  plt.plot(x1,x2,'bo')
# display all N points picked uniformly from the square
plt.show()
```

(a) (10 points) Write a C, C++ or Fortran program that computes this Monte Carlo approximation to π for $N = 10^k$ for $k = 2, 3, 4, 5, 6, 7, 8, 9, 10$. Now that you know two different ways to get data out of your code (redirecting the program's output or writing to a file) you should use the method you find best suited for the job. Also, note that if $\vec{x}$ is uniformly distributed on $R = [-1, 1]^2$, then its x_1 and x_2 coordinates are uniform scalar random variables on the domain $[-1, 1]$.

(b) (8 points) Plot N (x-axis) vs absolute error $|\pi - \pi_N|$ (y-axis) on a log-log plot. Provide convincing evidence that the error is decreasing at the correct rate. As in homework 1, consider many trials at a fixed value of N and plot the mean of the error over these many trials. See homework 1 for why this is necessary.

(c) (7 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 (x-axis) vs absolute error (y-axis) on a log-log plot. Estimate how long you would need to run the program to compute π to 10^{-16} accuracy (about the level of precision one can achieve with floating-point numbers using the C data type **double**). Estimate how long you would need to run the program to compute π to an accuracy of 10^{-70030}. This would match the world record for "known digits of π" (see http://www.pi-world-ranking-list.com/index.php?page=lists&category=pi).

Problem 2: (*5 points*)

In this problem, we will compare single core performance when running the code on your laptop vs Anvil.

(a) (5 points) Lets compare your laptop to Anvil using just a single core. First, run the program on your laptop to collect timing data for $N = 10^7, 10^8, 10^9$. Next, run on Anvil using a job submission script that requests 1 core for 5 minutes. How do the runtimes compare? Do these times make sense in light of each processor's clock rate? Why or why not? Upload the output of your job script into your bitbucket project's homework 2 folder. Make sure to include the timing numbers in your report (either as a figure, screenshot, or with words) **Note**: You might wish to run the program multiple times and average the runtime – this will give more reliable numbers.