ACMS 20210 Assignment 4

Due: 11:59 PM on Sunday, April 22, 2017

Submit the C++ programs below using Sakai. If you are compiling a single file from the command line on the CRC or another Linux system, I suggest using the command

```
g++ your_program_name.cpp -std=c++1y -o your_executable_name
```

To compile a program using several files, with the main routine in the file main_program.cpp and additional functions, classes, etc., in other source files souce_1.cpp and source_2.cpp with associated headers source_1.h and source_2.h, first, compile each separately into an object file:

```
g++ -std=c++1y -c source_1.cpp
g++ -std=c++1y -c source_2.cpp
g++ -std=c++1y -c main_program.cpp
```

This will produce a .o file for each of your .cpp files. To link them together into an executable, use:

```
g++ source_1.o source_2.o main_program.o -std=c++1y -o your_executable_name
```

The procedure is similar when the program is split over other numbers of source files.

1. In this problem, we will revise the implementation of Euler's method from the previous assignment to allow for an arbitrary real-valued function $f(x, y)$ as the right hand side over the interval $[0, b]$. We wish to approximate the solution to the ordinary differential equation:

$$\begin{aligned} y' &= f(x, y) \\ y(0) &= y_0 \end{aligned}$$

   using Euler's method and N subintervals.

   Your program should take four command line arguments: a double representing y_0, a double representing the endpoint b, an integer N for the number of subintervals, and the name of an output file into which to write the pairs (x_i, y_i) (as in the previous assignment).

   The function f will be declared in the header file hw61f.h and defined in the file hw61f.cpp. You will need to #include ''hw61f.h'' in your main routine. The exact header file that you should use will appear in the folder for this assignment, and an example of what the corresponding .cpp file might look like will also appear. A different definition for the function f will be used in testing your program, and you should try alternative definitions for f to see if they yield reasonable results.

   I have provided a sample of what the output should look like for the function $f(x, y) = xy$ using 1024 subintervals in the corresponding directory on Sakai.

2. In this problem, we will approximate the integral of a function $f(x)$ over an interval using a Monte Carlo integration procedure. The form of Monte Carlo integration over the interval $[a, b]$ that we will use works in the following way:

   (a) Select N points $x_1, \ldots, x_N$ from the interval $[a, b]$, uniformly at random.

   (b) Using these N points, compute $I = \dfrac{b-a}{N} \displaystyle\sum_{i=1}^{N} f(x_i)$.

   As a consequence of the law of large numbers, the quantity I will approximate the integral of $f(x)$ over the interval $[a, b]$.

   Your program should report the value I back to the user.

   Your program should take three command line arguments: a double representing a, a double representing b, and an integer representing the number of points selected in the interval $[a, b]$. The function $f(x)$ to be integrated will be defined in a separate .cpp named hw62f.cpp, with a corresponding header file hw62f.h. You will need to include the header file with this name in the file for your main routine. The exact header file that you should use will appear in the folder for this assignment, and an example of what the corresponding .cpp file might look like will also appear. A different definition for the function f will be used in testing your program, and you should try alternative definitions for f to see if they yield reasonable results.

3. In this problem, we will use a Monte Carlo method to determine experimentally the expected distance of a randomly selected point in the unit disk from a given point (a, b). To simulate this, we will draw N points $(x_1, y_1), \ldots, (x_N, y_N)$ from the unit disk, uniformly at random. For each point, we will compute the distance of the point to (a, b). The average of the distances of the selected points to (a, b) is an approximation of the expected distance of a randomly selected point in the unit disk to (a, b).

   Your program should take three command line arguments: A double for the coordinate a, a double for the coordinate b, and an integer N representing the number of trials.

   The exact average distance of a point selected uniformly at random from the disk to the origin $(0, 0)$ is $\frac{2}{3}$. The approximate average distance of a point selected uniformly at random from the disk to the point $(1, 0)$ is 1.1318.

4. In this problem, we will simulate a random walk on the d-dimensional integer lattice $\mathbb{Z}^d$. (A point in $\mathbb{Z}^d$ can be regarded as a vector with d components, all of which are integers.) A particle will walk along the lattice in the following way:

   (a) The particle starts at the origin.

   (b) For each step, one of the d possible coordinate directions is selected uniformly at random.

   (c) The particle then moves one step forward or backward in that coordinate direction with equal probability.

   Here is an example of what a walk of 5 total steps in $\mathbb{Z}^3$ might look like: Start at $(0, 0, 0)$. For the first step, the randomly chosen coordinate direction was the first coordinate, and we randomly selected to move forward one step to arrive at $(1, 0, 0)$. For the second step, the randomly chosen coordinate direction was the third coordinate, and we randomly selected to move backward one step to arrive at $(1, 0, -1)$. For the third step, the randomly chosen coordinate direction was the first coordinate, and we randomly selected to move forward one step to arrive at $(2, 0, -1)$. For the fourth step, the randomly chosen coordinate direction was the second coordinate, and we randomly selected to move forward one step to arrive at $(2, 1, -1)$. For the fifth step, the randomly chosen coordinate direction was the second coordinate, and we randomly selected to move backward one step to arrive at $(2, 0, -1)$.

   So, for this particular walk, we arrived at $(2, 0, -1)$ after five steps of the walk, at a distance of $\sqrt{5}$ from the origin.

   Using a Monte Carlo simulation, report to the user both the average (Euclidean) distance of the location of the particle to the origin after N steps of this random walk.

   Your program should take three command line arguments: an integer d representing the number of dimensions in which to perform the walk, an integer K representing the number of steps to take on each walk, and an integer N representing the number of trials for your Monte Carlo simulation.

   As an example, for 3 dimensional random walk, the average distance to the origin after 100 steps is approximately 9.22. In one dimension, the average distance to the origin after K steps is roughly $\sqrt{2K/\pi}$.