ACMS 20210 Assignment 5

Due: 11:59 PM on Thursday, April 5, 2018

Submit the C++ programs below using Sakai. If you are compiling 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
```

This tells the compiler (called g++) to compile your .cpp source file into an executable file. To run this executable file from the command line, type

```
./your_executable_name
```

You may always assume that the user enters a reasonable value (e.g. if the problem states the user enters a positive integer, you can assume that they enter a positive integer).

Use the following naming scheme for submitting your solutions on Sakai: For exercise n on homework m, the name of your submission should be hw_m_ex_n.cpp. As an example, for homework 1 exercise 3, the name of your submission should be: hw_1_ex_3.cpp.

1. Rewrite Problem 3 (the statistical analysis problem) from the previous homework so that, instead of prompting the user for input, the program takes a single command line argument containing the name of a file. The file will contain two doubles on each line, separated by a space, with the first representing an x value and the second representing a y value. Use these values to form your vectors of x and y observations. The program should report all of the same analysis as in the previous homework to the user.

   See the appropriate folder for a sample of what the input might look like.

2. Write a C++ program that takes a command line argument specifiying the name of a file. Each line of the file will consist of integers, separated by semicolons. The number of integers per line will vary. The program computes the sum of each line, and writes the totals (in order) to an output file named "hw_5_ex_2_output.dat", with one total per line. See the appropriate folder for a sample of what the input and output might look like.

3. Write a program implementing Newton's method for finding the real root of the function $f(x) = x^5 - x + 1$. (See the last page of this assignment for the details of Newton's method.) The program takes up to three optional command line arguments for the number of iterations of Newton's method to perform. If no command line argument is specified, perform 20 iterations with a starting value $x_0 = 0$ and an error tolerance of .0000001. If one command line argument is specified, assume that it represents the starting value x_0, and carry out Newton's method for a maximum of 20 iterations with an error tolerance of .0000001. If two command line arguments are specified, the first will represent the starting value for x_0, and the second the error tolerance, and the procedure will carry out a maximum of 20 iterations. If three or more command line arguments are specified, the first will represent x_0, the second the error tolerance, and the third the maximum number of iterations.

   In your code, account for the possibility that the derivative of the function will be zero at x_i and terminate the procedure early if this occurs. (You should still write output in this case.)

   Implement the function $f(x)$ and its derivative $f'(x) = 5x^4 - 1$ as appropriately named functions in your code.

   Write the initial guess and each successive iteration into an output file named "hw_5_ex_3_output.dat", with one double per line. Use 16 digits of precision for the output.

   See the appropriate directory for sample output.

4. Write a program that takes an arbitrary number of file names as command line arguments and counts the frequency of each letter in the English alphabet (A through Z) appearing among all the files. Ignore all characters that are not part of the basic English alphabet. You should regard upper and lower case of the same letter as the same. Report the count of each letter, along with its overall frequency (as a percentage of total letters) to the user. You do not have to worry about letters with diacritical marks (e.g. umlauts) or other variants from the standard alphabet. (Rather than using many conditionals or switch, I suggest using a vector or map to store counts.)

5. Write a program that takes a command line argument for the name of a file. The program then reads two matrices (whose entries are integers) from the file, with both matrices guaranteed to be of the same dimensions. The matrices will be separated from each other in the file by a blank line. The entries in each row of the matrix will be separated by a blank space, with no characters other than spaces and those forming the integers appearing on each separate line. The program then outputs the sum of the two matrices to a file named "hw_5_ex_5_solution.dat", with each row of the matrix on a separate line in the file, and the entries of each row in the matrix separated by a space. (Remember that addition of matrices is defined by adding corresponding elements.) I suggest using the type vector<vector<int>> to store each of your matrices.

   See the appropriate folder for a sample of what the input might look like.

Newton's Method

Newton's method is a procedure for finding a root of a function $f(x)$ (i.e. to find a value x^* such that $f(x^*) = 0$).

The method begins with an initial guess x_0 for a root of the function. It then constructs an iterative sequence of approximations based on the formula: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$.

This procedure might terminate early if $f'(x_i)$ is 0 for some i. Otherwise, it may produce an exact root (in which case there is no need to continue the procedure), a sequence of approximations converinging to a root, or a divergent sequence of x_i values.

In practice, the procedure is run for a maximum number of iterations, or until a stopping condition is reached (or until the derivative becomes zero). The stopping condition might take the error condition form: stop once $|f(x_i)| < \epsilon$, where ϵ is an error tolerance. There are other stopping conditions which are often better in practice.

Below is an example of using Newton's method to approximate the square root of 2 by finding a root of $f(x) = x^2 - 2$, using an initial guess of 1 and an error tolerance of .000001.

$x_0 = 1$
$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^2-2}{(2)(1)} = 1.5$
$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1 - \frac{1.5^2-2}{(2)(1.5)} = 1.416666666666667$
$x_3 = 1.41421568627451$
$x_4 = 1.41421356237469$ (the procedure stops here because $|f(1.41421356237469)| < .000001$.