Data Science 2G03 Portfolio

Kyle Drury

December 2022

Abstract

The following paper is a written report of the work that I have done in Dr. Wadsley's class at McMaster University *Data Science 2G03*. Students were required to complete weekly assignments, usually in the form of fully coded C++ programs created with Unix. The programs featured in this paper have been slightly embellished and/or streamlined from the versions assigned in class.

1 Prime Number Detector

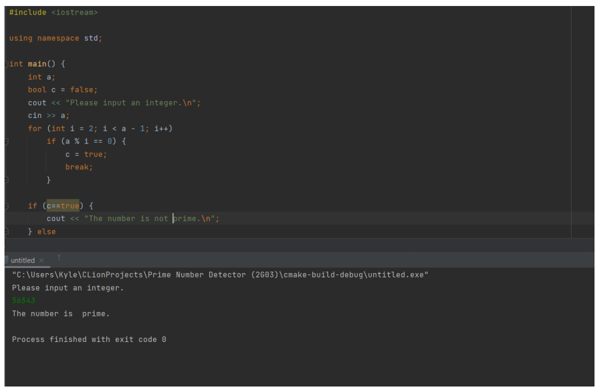
A program for determining whether a number is prime.

1.1 The Algorithm Used

In psuedo-code, the algorithm looks something like this:

```
1. Initial conditions: integer = a, i = 2
2. Begin for
3. for (a-1) > i > 2
4. if the remainder of a/i is = 0, end for
```

1.2 Testing the Program



```
#include <iostream>

using namespace std;

int main() {
    int a;
    bool c = false;
    cout << "Please input an integer.\n";
    cin >> a;
    for (int i = 2; i < a - 1; i++)
        if (a % i == 0) {
            c = true;
            break;
        }

    if (c==true) {
        cout << "The number is not prime.\n";
    } else
```

```
"C:\Users\Kyle\CLionProjects\Prime Number Detector (2G03)\cmake-build-debug\untitled.exe"
Please input an integer.
56543
The number is  prime.

Process finished with exit code 0
```

Figure 1: Testing the Prime Number Detector

2 Partial Sum Pi Estimator

A program that estimates the value of using the infinite series below within a minimum error specified by the user.

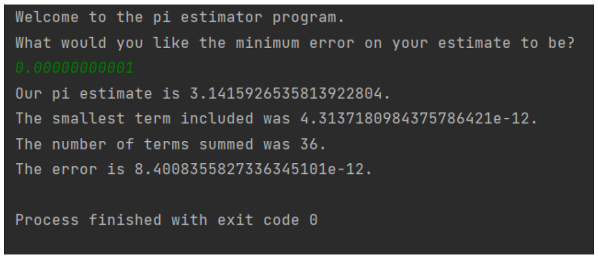
$$\sum_{2}^{\infty} \frac{c_{n-1}}{(n-1)/(2n-1)} \qquad\qquad c_1 = 1 \tag{1}$$

2.1 The Algorithm

In psuedo-code, the algorithm looks like this:

```
1. Minimum error = cutoff, n = i = 1, term = 1, sum = term
2. Begin for
3. i = i + 1, n = n + 1
4. term = term*(n-1)/(2n-1)
5. sum = sum + term
6. When (sum - pi) < cutoff, break
```

2.2 Testing the Program



```
Welcome to the pi estimator program.
What would you like the minimum error on your estimate to be?
0.00000000001
Our pi estimate is 3.1415926535813922804.
The smallest term included was 4.3137180984375786421e-12.
The number of terms summed was 36.
The error is 8.4008355827336345101e-12.

Process finished with exit code 0
```

Figure 2: Testing the pi estimator; minimum error of 0.00000000001

3 Approximating a Derivative and Integral with Finite Differences

A program to numerically calculate the derivative and integral of a function, and calculate the rms on both of these numerical calculations.

3.1 The Code

The entire program must work to carry out all of the following tasks:

1. Plot the given polynomial using an array by calculating the output of the function for small but arbitrary-sized steps *dx.*
2. Plot the approximate derivative by calculating the slope between each pair of adjacent points of the function.
3. Plot the analytical derivative (determined algebraically), and compare the y values of the approximate and exact derivatives at the same x points to get the difference (use these differences to calculate the rms error).
4. Plot the approximate integral by finding the area of small rectangles of height y and width dx for every step *dx.*

5. Plot the exact integral and use these points along the curve to determine the rms error of the integral.

The Euler Method (for approximating the derivative) in pseudo-code:

```
1. $x = x_{start} y = y_{start}, dx = \frac{t_{end}-t_{start}}{n}$, i = 0
2. begin for
3. $y_{i+1} = y_i + dx(\frac{dy}{dx}(x, y, parameters))$
4. $x_{i+1} = x_i + dx$
5. i = i + 1
6. when i = n - 1, break
```

Using a Riemann Sum to approximate the integral (let the integral of $f(x)$ be $F(x)$):

```
1. F(0) = 0, $y = y_0, x = x_0, dx = \frac{t_{end}-t_{start}}{n}$, i = 0
2. begin for
3. $F(x_i + 1) = F(x_i + dxf(x_i + \frac{dx}{2})$
4. i = i + 1
5. for $i < n$, go to step 3
```

3.2 Testing the Program

In the plots below, the green line is the polynomial, the approximate and exact derivative are red and blue, respectively, and the approximate and exact integral are orange and pink, respectively.

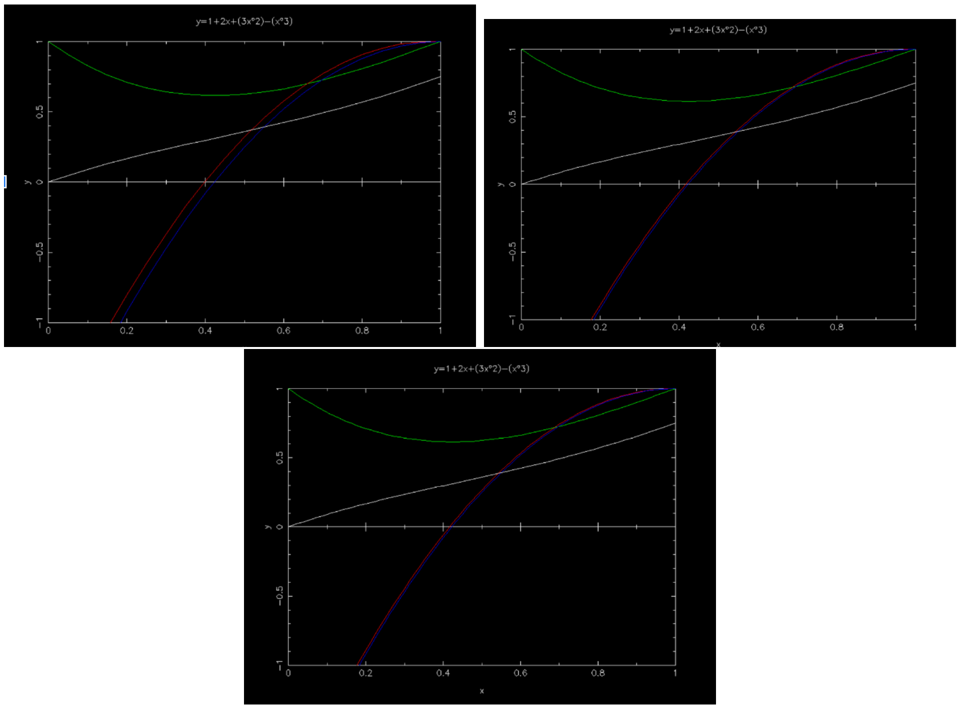


Figure 3: Number of steps = 20, 40, and 80 respectively

Number of Steps	RMS on Derivative	RMS on Integral	Max Error on Derivative	Max Error on Integral
20	0.0876606	0.000222805	0.1475	0.000311673
40	0.0435688	5.63605e-05	0.0743742	7.80225e-05
80	0.0217179	1.42473e-05	0.037343	1.97291e-05

Table 1: Error Calculations

4 Numerically Solving an Ordinary Differential Equation

A program that solves a differential equation with the Euler method and the Fourth Order Runge Kutta method.

$$\frac{dy}{dt} = a(1-y). \tag{2}$$

4.1 The Algorithms

4.1.1 The Euler Method

```
1. t = $t_{start}$, y = y(0), Δt = $\frac{(t_{end}-t_{start}}{n}$), i = 0
2. Begin while
3. t = t + dt
4. y = y + dt$\frac{dy}{dt}$(y,a)
5. i = i + 1
6. while i < n, continue at 3
```

4.1.2 The Runge Kutta Method

```
1. t = $t_{start}$, y = y(0), Δt = $\frac{(t_{end}-t_{start}}{n}$), i = 0
2. Begin while
3. $k_1$ = dt$\frac{dy}{dt}(y,a)$
4. $k_2$ = dt$\frac{dy}{dt}(y+0.5k_1,a)$
5. $k_3$ = dt$\frac{dy}{dt}(y+0.5k_2,a)$
6. $k_4$ = dt$\frac{dy}{dt}(y+k_3,a)$
7. t = t + dt
8. y = y + $(k_1+2k_2+2k_3+k_4)/6$)
9. i = i + 1
10. while i < n, go to step 3
```

4.2 Testing the Program

Below are the plots of the approximated derivative in red and the exact derivatives in green. The left plot is uses the Euler method and the right plot uses the 4th order Runge Kutta method.

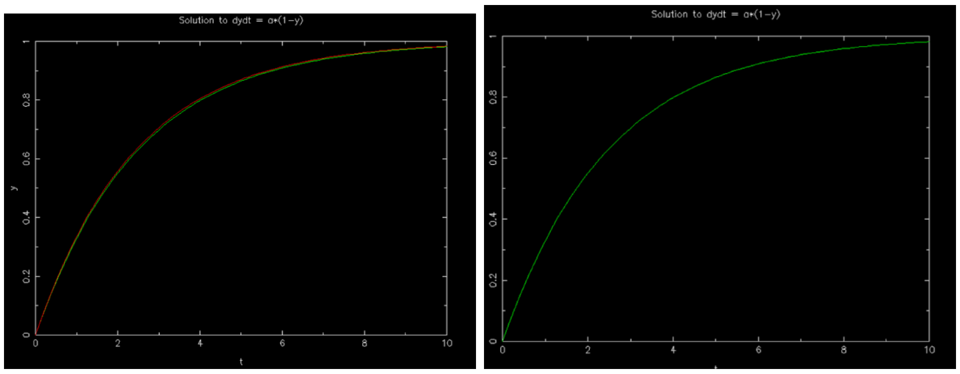


Figure 4:

5 Linear Interpolation

A program that accepts an x value and determines which two values (in a list) it falls between, and uses linear interpolation to determine the unknown y value of the inputted x value.

5.1 The Code

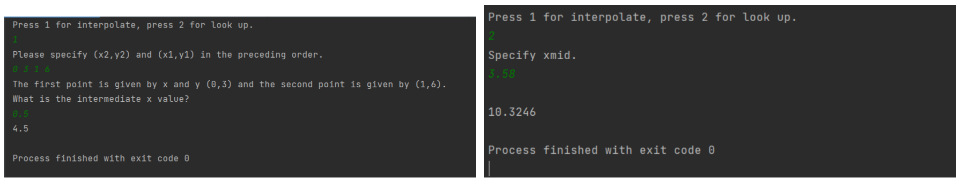
The program is should be able to do two different tasks; one, it should be able to interpolate between two points to find an unknown point; two, it also should be able to take this same input and compare it to a *list* of points, decide which two points it lies between, and interpolate between these two points.

The equation of a line can be represented as $y = mx + b$. Subbing in m = $\frac{y_2-y_1}{x_2-x_1}$ and b = y - mx = y - ($\frac{y_2-y_1}{x_2-x_1}$)x, we can find the output the function gives for the given x value that the user chose:

$$y_{mid} = mx + b = (\frac{y_2-y_1}{x_2-x_1})x_{mid} + y_2 - (\frac{y_2-y_1}{x_2-x_1})x_2 \tag{3}$$

The program has a set of points with which it can compare the user's input, determines which two points it lies between, and then interpolate between those two points. For this demonstration of the program, the points along the curve that the program will use for interpolation are the following: (0, 9.0), (1, 9.37), (2, 10.48), (3, 12.33), (4, 14.92). The program will "look up" which two points lie on either side of the input.

5.2 Testing the Program



```
Press 1 for interpolate, press 2 for look up.
1
Please specify (x2,y2) and (x1,y1) in the preceding order.
0 3 1 6
The first point is given by x and y (0,3) and the second point is given by (1,6).
What is the intermediate x value?
0.5
4.5

Process finished with exit code 0
```

```
Press 1 for interpolate, press 2 for look up.
2
Specify xmid.
3.58


10.3246


Process finished with exit code 0
```

Figure 5: Testing 'interpolate' and 'look up'

6 Sorting

A program that accepts a list of values and can order them from least to greatest using the well known Quick Sort and Partition Sort algorithms.

6.1 The Algorithms

```
1. for i = 1 to n-1 (n number of array elements)
2. key = A[i]
3. j = i - 1
4. while j $\leq 0$ and $A[j] > key$
5. A[j+1] = A[j]
6. j = j - 1
7. end while
8. A[j+1] = key
9. end for
```

Quick sort relies on partitioning the original array into two separate ones, such that the values in the array A[0...(q-1)] are all less than or equal to A[q], and A[q] is less than or equal to all the values in the array A[(q-1)...(n-1)]. These smaller arrays are sorted with recursive calls to the Quick Sort function.

The following is the algorithm for partitioning:

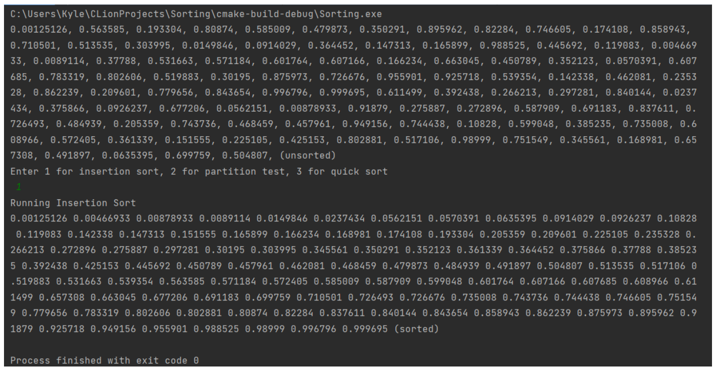
```
1. q = A[n-1], j = i = 0
2. for j $\leq n-2$, j++
3. if A[j] $\leq n-2$, switch A[j] and A[i]
4. i = i + 1
5. end if
6. end for
```

To use the partition and insertion sort functions in the quick sort algorithm, refer to the following pseudo-code:

```
 1. q = A[n-1]
 2. i = 0
 3. for j = 0 to n - 2
 4. if A[j] $\leq$ x then
 5. Swap A[i] and A[j]
 6. i = i + 1
 7. end if
 8. for
 9. Swap A[i] and A[n-1]
10. return i
```

6.2 Testing the Program

In the below figure, the program is shown generating a list of 100 random values between 0 and 1 and using the quick sort algorithm to order them from least to greatest.



```
C:\Users\Kyle\CLionProjects\Sorting\cmake-build-debug\Sorting.exe
0.00125126, 0.563585, 0.193304, 0.80874, 0.585009, 0.479873, 0.350291, 0.895962, 0.82284, 0.746605, 0.174108, 0.858943,
0.710501, 0.513535, 0.303995, 0.0149846, 0.0914029, 0.364452, 0.147313, 0.165899, 0.988525, 0.445692, 0.119083, 0.004669
33, 0.0089114, 0.37788, 0.531663, 0.571184, 0.601764, 0.607166, 0.166234, 0.663045, 0.450789, 0.352123, 0.0570391, 0.607
685, 0.783319, 0.802606, 0.519883, 0.30195, 0.875973, 0.726676, 0.955901, 0.925718, 0.539354, 0.142338, 0.462081, 0.2353
28, 0.862239, 0.209601, 0.779656, 0.843654, 0.996796, 0.999695, 0.611499, 0.392438, 0.266213, 0.297281, 0.840144, 0.0237
434, 0.375866, 0.0926237, 0.677206, 0.0562151, 0.00878933, 0.91879, 0.275887, 0.272896, 0.587909, 0.691183, 0.837611, 0.
726493, 0.484939, 0.205359, 0.743736, 0.468459, 0.457961, 0.949156, 0.744438, 0.10828, 0.599048, 0.385235, 0.735008, 0.6
08966, 0.572405, 0.361339, 0.151555, 0.225105, 0.425153, 0.802881, 0.517106, 0.98999, 0.751549, 0.345561, 0.168981, 0.65
7308, 0.491897, 0.0635395, 0.699759, 0.504807, (unsorted)
Enter 1 for insertion sort, 2 for partition test, 3 for quick sort
 1
Running Insertion Sort
0.00125126 0.00466933 0.00878933 0.0089114 0.0149846 0.0237434 0.0562151 0.0570391 0.0635395 0.0914029 0.0926237 0.10828
 0.119083 0.142338 0.147313 0.151555 0.165899 0.166234 0.168981 0.174108 0.193304 0.205359 0.209601 0.225105 0.235328 0.
266213 0.272896 0.275887 0.297281 0.30195 0.303995 0.345561 0.350291 0.352123 0.361339 0.364452 0.375866 0.37788 0.38523
5 0.392438 0.425153 0.445692 0.450789 0.457961 0.462081 0.468459 0.479873 0.484939 0.491897 0.504807 0.513535 0.517106 0
.519883 0.531663 0.539354 0.563585 0.571184 0.572405 0.585009 0.587909 0.599048 0.601764 0.607166 0.607685 0.608966 0.61
1499 0.657308 0.663045 0.677206 0.691183 0.699759 0.710501 0.726493 0.726676 0.735008 0.743736 0.744438 0.746605 0.75154
9 0.779656 0.783319 0.802606 0.802881 0.80874 0.82284 0.837611 0.840144 0.843654 0.858943 0.862239 0.875973 0.895962 0.9
1879 0.925718 0.949156 0.955901 0.988525 0.98999 0.996796 0.999695 (sorted)

Process finished with exit code 0
```

Figure 6: Sorting 100 random values between 0 and 1