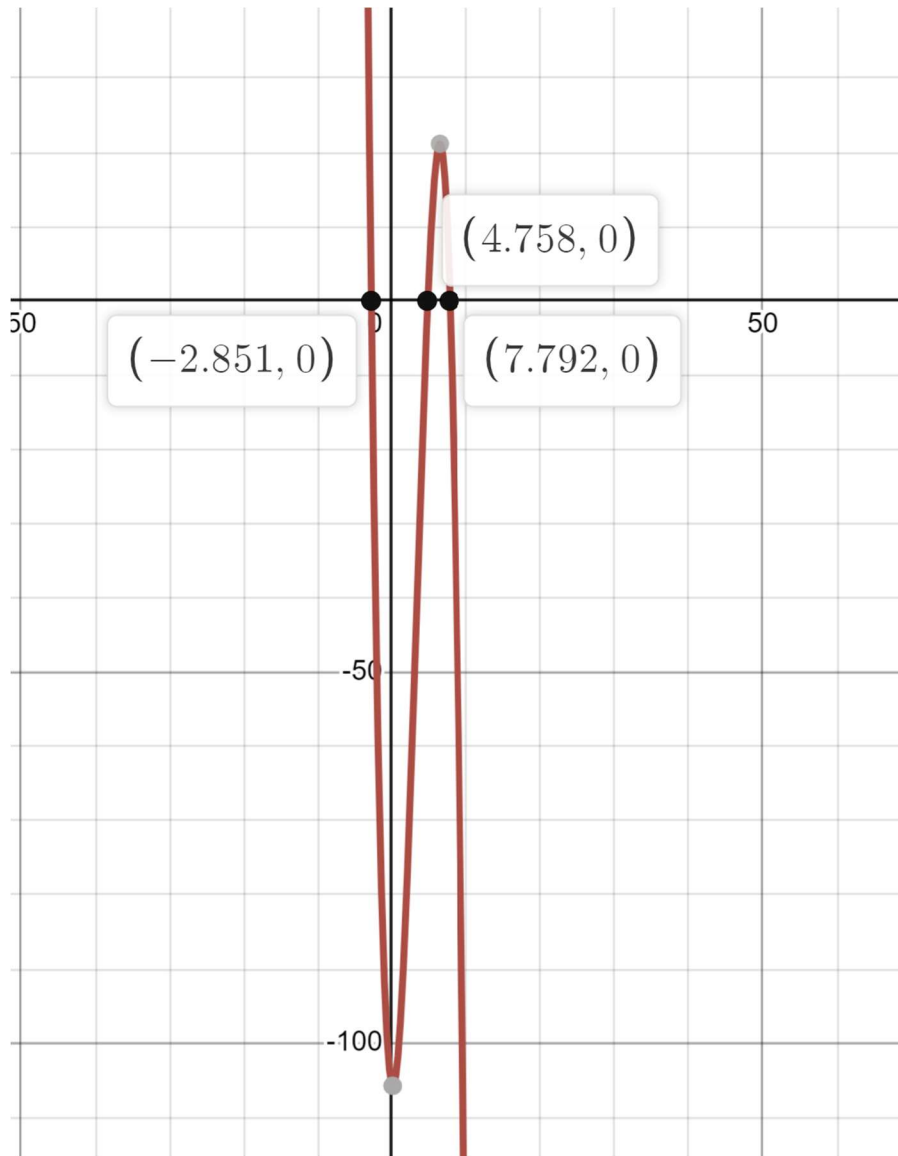


Assignment 6

Part1)

In order to plot the function $-x^3 + 9.7x^2 - 1.3x - 105.7$ I will use desmos to help visualize what this function looks like graphically you can see the graph plotted below.

As we can see there seems to be roots at $x = -2.851, 4.758, 7.792$



Now if we were to take the derivative of the function above this would be the result

$$= \frac{194x - 13}{10} - 3x^2$$

To further confirm these roots I will implement this into Symbolab to verify the roots

The solutions are

$$x \approx -2.85071..., x \approx 4.75824..., x \approx 7.79246...$$

This function will be used to calculate the roots for our Newton Raphson function

In order to plug in our function and derivative in our Newton Raphson program I will do the following

```
double f(double x)
{
    return (0-pow(x,3)+(9.7*pow(x,2))-1.2*x-105.7);
}

double df (double x)
{
    return(0-3*pow(x,2)+19.4*x-1.3);
}
```

f is our original function and df is the derivative of our function

with this modification to both of those functions we should be able to find the roots for our function.

Now I will try to find all the roots with the program I currently have

First I will try to find x=7.792 given to us by desmos

As we can see we are able to get reasonably close to the root value that desmos provided for us with an initial guess of 7 error of 0.01% and max iterations of 10 our result we were able to obtain is 7.816 so our error will turn out to be .3%

```
juan@DESKTOP-QCQU6GMF:$ ./newton

Enter x0, allowed error and maximum iterations
7 .01 10
x0=7.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 8.456000
At Iteration no. 2, x = 7.936419
At Iteration no. 3, x = 7.822309
At Iteration no. 4, x = 7.816461
After 4 iterations, root = 7.816461071438211
juan@DESKTOP-QCQU6GMF:$
```

$$\frac{(7.792 - 7.816)}{7.792} \cdot 100$$

$$= -0.3080082136$$



Unfortunately, I am not able to get an error less than 0.1% as changing the value of my initial guess, allowed error, and max iterations did not result in different outputs

Now I will attempt to get the 4.758 root with 0.01 error and 10 max iterations

As we can see our result was 4.737

```
juan@DESKTOP-QCQU6MF:~$ ./newton
Enter x0, allowed error and maximum iterations
4 .01 10
x0=4.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 4.681979
At Iteration no. 2, x = 4.737462
At Iteration no. 3, x = 4.737806
After 3 iterations, root = 4.737805850635014
juan@DESKTOP-QCQU6MF:~$
```

The error here is .4%

$$\frac{(4.758 - 4.737)}{4.758} \cdot 100 = 0.4413619168$$

Now I will try to find the -2.850 root with the Newton Raphson algorithm with a -5 guess, an error of .01 and 10 max iterations

```
juan@DESKTOP-QCQU6MF:~$ ./newton
Enter x0, allowed error and maximum iterations
-5 .01 10
x0=-5.000000, err=0.010000, max iter=10
At Iteration no. 1, x = -3.454703
At Iteration no. 2, x = -2.922208
At Iteration no. 3, x = -2.855331
At Iteration no. 4, x = -2.854235
After 4 iterations, root = -2.854234971579705
juan@DESKTOP-QCQU6MF:~$
```

The error here is .1%

Looks like the algorithm was able to get a good calculation with only a 0.14% error


$$\frac{(2.850 - 2.854)}{2.850} \cdot 100 = -0.1403508772$$


All of the roots our algorithm found are


7.816, 4.737, and -2.854

I will plug these back into our original equation to see how close to zero we get

As we can see these results are all less than 1 unit away from zero which is decently accurate

$x = 7.816$	$= 7.816$ 
$-x^3 + 9.7x^2 - 1.3x - 105.7$	$= -0.767511296$

$x = 4.737$	$= 4.737$ 
$-x^3 + 9.7x^2 - 1.3x - 105.7$	$= -0.492504253$

$x = -2.854$	$= -2.854$ 
$-x^3 + 9.7x^2 - 1.3x - 105.7$	$= 0.266497064$

Find Roots W Regula Falsi

In order to find roots using Regula Falsi I will first need to define the function in the given starter code

Here is that function

```
double f(double x)
{
    return (0-pow(x,3)+(9.7*pow(x,2))-1.2*x-105.7);
}
```

With that we can run the program then input the bound which we will then use to find the root

First I will attempt to find the 7.792 root using a bound from 5 to 9 with an error of 0.01 and maximum iteration of 10

```

juan@DESKTOP-QCQU6MF:$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
5 9 0.01 10
x0=5.000000, x1=9.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.35366
Iteration no. 2 X = 5.98195
Iteration no. 3 X = 6.74308
Iteration no. 4 X = 7.32253
Iteration no. 5 X = 7.62192
Iteration no. 6 X = 7.74524
Iteration no. 7 X = 7.79111
Iteration no. 8 X = 7.80752
Iteration no. 9 X = 7.81331
After 9 iterations, root = 7.813306798435394
juan@DESKTOP-QCQU6MF:$ 

```

The error here is .27%

$$\frac{(7.792 - 7.813)}{7.792} \cdot 100 = -0.2695071869$$

Now I will try to find the 4.758 root using a bound from 2 to 6 with a 0.01 error and maximum iterations of 10

```

juan@DESKTOP-QCQU6MF:$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
2 6 .01 10
x0=2.000000, x1=6.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.16803
Iteration no. 2 X = 4.83303
Iteration no. 3 X = 4.75519
Iteration no. 4 X = 4.74083
Iteration no. 5 X = 4.73833
After 5 iterations, root = 4.738326359046614
juan@DESKTOP-QCQU6MF:$ 

```

The Error here is .4%

$$\frac{(4.758 - 4.738)}{4.758} \cdot 100 = 0.4203446826$$

Now I will try to find the -2.580 root using a bound from -4 to 0 with a 0.01 error and maximum iterations of 10

```

juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
-4 0 .01 10
x0=-4.000000, x1=0.000000, err=0.010000, iter=10
Iteration no. 1 X = -1.88750
Iteration no. 2 X = -2.61510
Iteration no. 3 X = -2.80100
Iteration no. 4 X = -2.84268
Iteration no. 5 X = -2.85174
After 5 iterations, root = -2.851742079352846
juan@DESKTOP-QCQU6MF:~$

```

The error here is .03%

$$\frac{(2.850 - 2.851)}{2.850} \cdot 100 = -0.0350877193$$

Compare Newton Raphson to Regula Falsi

For root 7.792

Newton Raphson

Regula Falsi

7.816

7.813

```

juan@DESKTOP-QCQU6MF:~$ ./newton

Enter x0, allowed error and maximum iterations
9 .01 10
x0=9.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 8.142037
At Iteration no. 2, x = 7.853360
At Iteration no. 3, x = 7.817093
At Iteration no. 4, x = 7.816431
After 4 iterations, root = 7.816431141099257
juan@DESKTOP-QCQU6MF:~$

```

```

juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
5 9 0.01 10
x0=5.000000, x1=9.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.35366
Iteration no. 2 X = 5.98195
Iteration no. 3 X = 6.74308
Iteration no. 4 X = 7.32253
Iteration no. 5 X = 7.62192
Iteration no. 6 X = 7.74524
Iteration no. 7 X = 7.79111
Iteration no. 8 X = 7.80752
Iteration no. 9 X = 7.81331
After 9 iterations, root = 7.813306798435394
juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

```

Iterations = 4

iterations = 9

For root 4.758

Newton Raphson

Regula Falsi

4.737

4.738

```

juan@DESKTOP-QCQU6MF:~$ ./newton
Enter x0, allowed error and maximum iterations
2 0.01 10
x0=2.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 5.031373
At Iteration no. 2, x = 4.714777
At Iteration no. 3, x = 4.737802
At Iteration no. 4, x = 4.737804
After 4 iterations, root = 4.737804414861446
juan@DESKTOP-QCQU6MF:~$

```

Iterations = 4

```

juan@DESKTOP-QCQU6MF:~$ ./regulafalsi
Enter the values of x0, x1, allowed error and maximum iterations:
2 6 .01 10
x0=2.000000, x1=6.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.16803
Iteration no. 2 X = 4.83303
Iteration no. 3 X = 4.75519
Iteration no. 4 X = 4.74083
Iteration no. 5 X = 4.73833
After 5 iterations, root = 4.738326359046614

```

iterations = 5

For root -2.850

Newton Raphson

-2.854

```

juan@DESKTOP-QCQU6MF:~$ ./newton
Enter x0, allowed error and maximum iterations
-4 0.01 10
x0=-4.000000, err=0.010000, max iter=10
At Iteration no. 1, x = -3.067770
At Iteration no. 2, x = -2.864043
At Iteration no. 3, x = -2.854267
After 3 iterations, root = -2.854266984134887
juan@DESKTOP-QCQU6MF:~$

```

Iterations = 3

Regula Falsi

-2.851

```

juan@DESKTOP-QCQU6MF:~$ ./regulafalsi
Enter the values of x0, x1, allowed error and maximum iterations:
-4 0 .01 10
x0=-4.000000, x1=0.000000, err=0.010000, iter=10
Iteration no. 1 X = -1.88750
Iteration no. 2 X = -2.61510
Iteration no. 3 X = -2.80100
Iteration no. 4 X = -2.84268
Iteration no. 5 X = -2.85174
After 5 iterations, root = -2.851742079352846

```

iterations = 5

Using similar initial values in the Newton Raphson's algorithm as the first argument used in the Regula Falsi algorithm I conclude that Newton Raphson is faster in finding the roots meaning it takes less iterations

Timing

Now I will time the both of these programs and compare their performance using POSIX gettimeofday

Newton = .000072s, Regula = .000050s

```
juan@DESKTOP-QCQU6MF:~$ ./newton

Enter x0, allowed error and maximum iterations
9 0.01 10
x0=9.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 8.142037
At Iteration no. 2, x = 7.853360
At Iteration no. 3, x = 7.817093
At Iteration no. 4, x = 7.816431
After 4 iterations, root = 7.816431141099257

Total time: 0.000072 seconds
juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
5 9 .01 10
x0=5.000000, x1=9.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.35366
Iteration no. 2 X = 5.98195
Iteration no. 3 X = 6.74308
Iteration no. 4 X = 7.32253
Iteration no. 5 X = 7.62192
Iteration no. 6 X = 7.74524
Iteration no. 7 X = 7.79111
Iteration no. 8 X = 7.80752
Iteration no. 9 X = 7.81331
After 9 iterations, root = 7.813306798435394

Total time: 0.000050 seconds
juan@DESKTOP-QCQU6MF:~$
```

Newton = 0.000062s Regula = 0.000014s


```

juan@DESKTOP-QCQU6MF:~$ ./newton

Enter x0, allowed error and maximum iterations
2 0.01 10
x0=2.000000, err=0.010000, max iter=10
At Iteration no. 1, x = 5.031373
At Iteration no. 2, x = 4.714777
At Iteration no. 3, x = 4.737802
At Iteration no. 4, x = 4.737804
After 4 iterations, root = 4.737804414861446

Total time: 0.000062 seconds
juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
2 6 0.01 10
x0=2.000000, x1=6.000000, err=0.010000, iter=10
Iteration no. 1 X = 5.16803
Iteration no. 2 X = 4.83303
Iteration no. 3 X = 4.75519
Iteration no. 4 X = 4.74083
Iteration no. 5 X = 4.73833
After 5 iterations, root = 4.738326359046614

Total time: 0.000014 seconds

```

Newton = 0.000051s, Regula = 0.000031s

```

juan@DESKTOP-QCQU6MF:~$ ./newton

Enter x0, allowed error and maximum iterations
-4 0.01 10
x0=-4.000000, err=0.010000, max iter=10
At Iteration no. 1, x = -3.067770
At Iteration no. 2, x = -2.864043
At Iteration no. 3, x = -2.854267
After 3 iterations, root = -2.854266984134887

Total time: 0.000051 seconds
juan@DESKTOP-QCQU6MF:~$ ./regulafalsi

Enter the values of x0, x1, allowed error and maximum iterations:
-4 0 0.01 10
x0=-4.000000, x1=0.000000, err=0.010000, iter=10
Iteration no. 1 X = -1.88750
Iteration no. 2 X = -2.61510
Iteration no. 3 X = -2.80100
Iteration no. 4 X = -2.84268
Iteration no. 5 X = -2.85174
After 5 iterations, root = -2.851742079352846

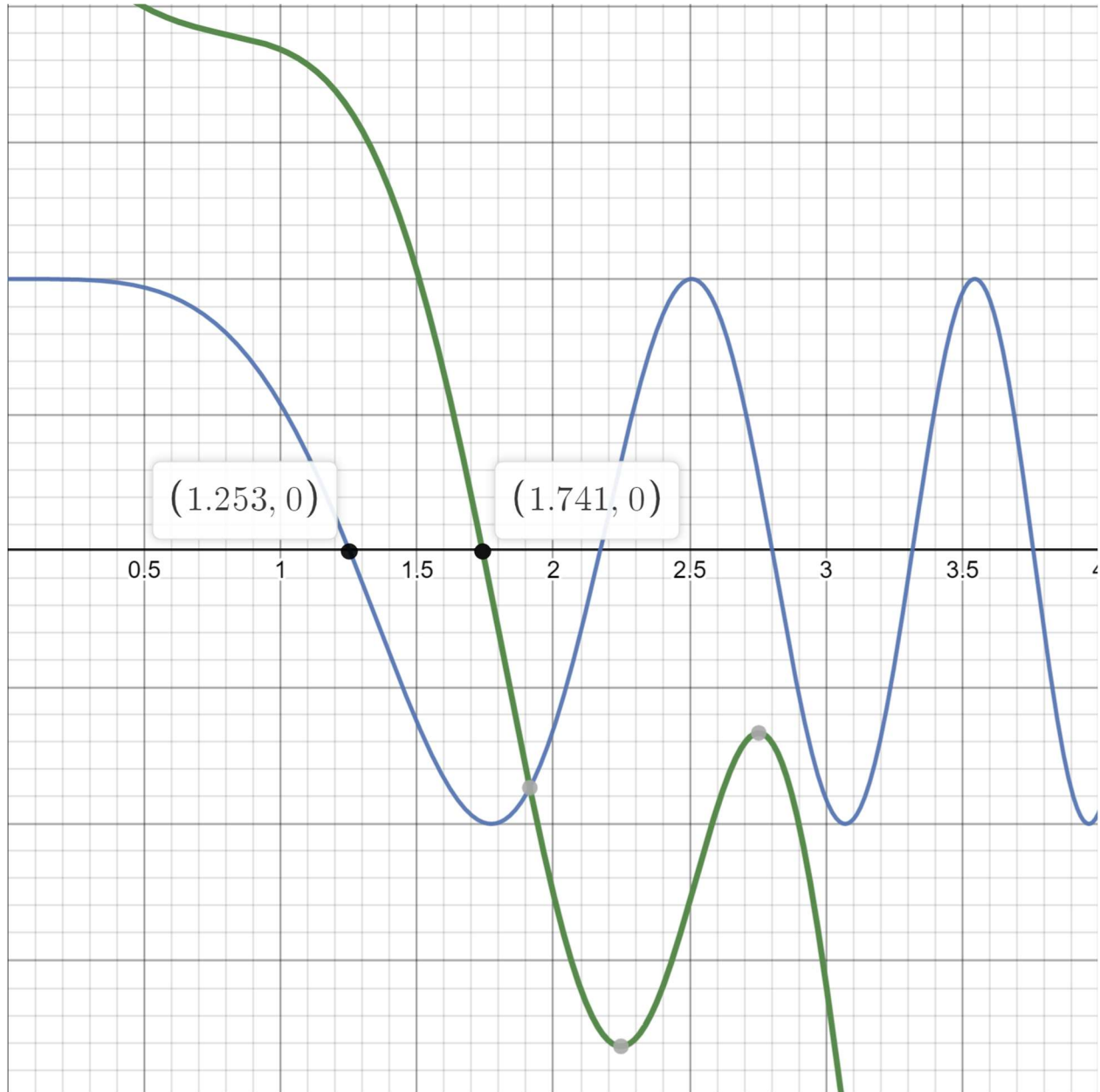
Total time: 0.000031 seconds

```

So to conclude the timing measurements it seems that Regula Falsi is definitely faster than Newton Raphson by a good margin

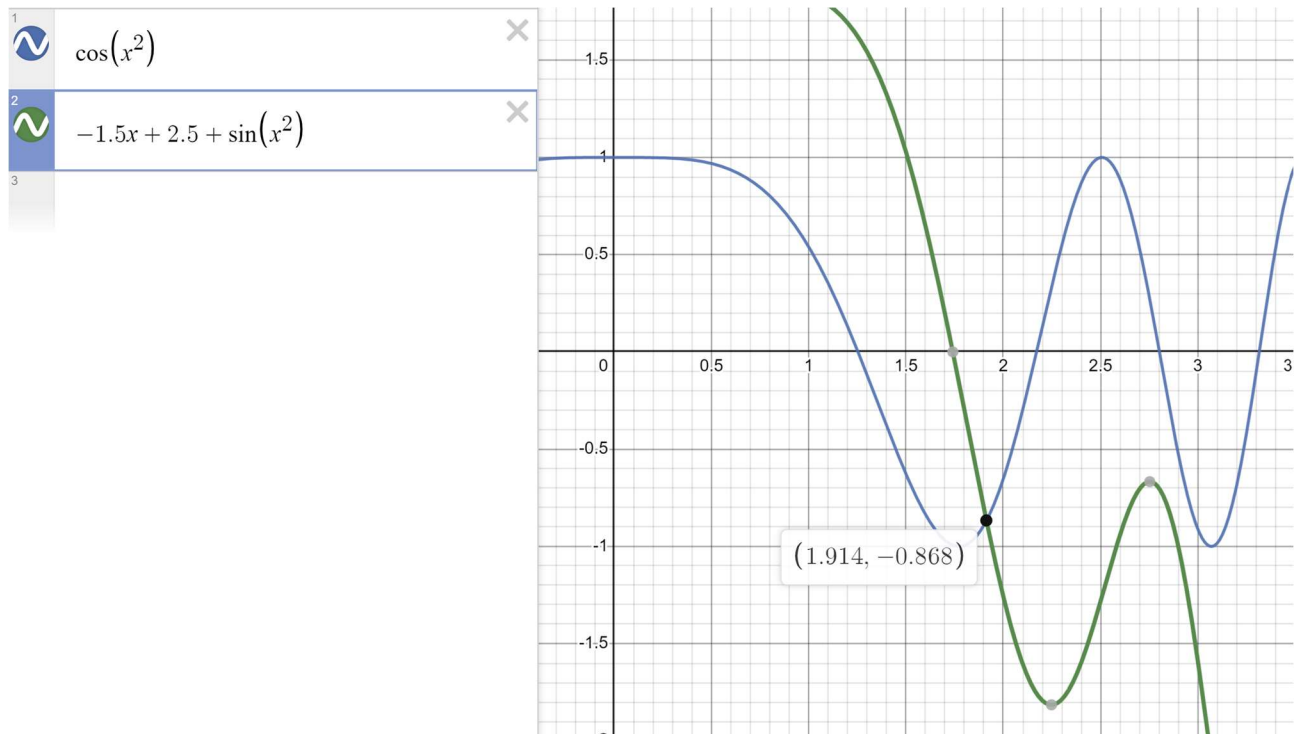
Part2)

Here we can see our first roots for both of the functions the first root coming in at $X=1.253$ for $\cos(x^2)$ and $X=1.741$ for our next root for our other function

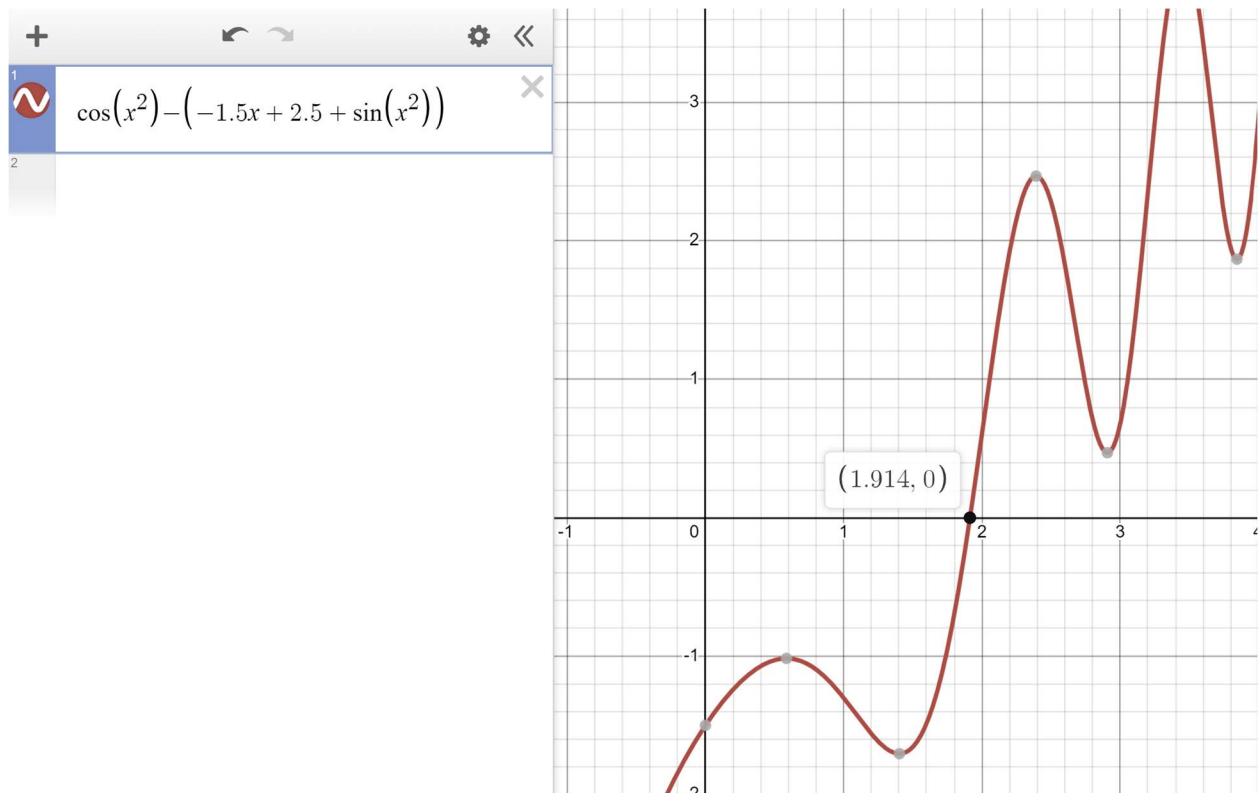


We can also accurately see exactly where the two functions intersect

We can see here that the two functions intersect at $X=1.914$



Now if we used the $F(x)=f(x)-g(x)$ convention we can see more accurately that there is a root at 1.914 meaning there is an intersection there



All roots from 0 – 10 using $\cos(x^2)$

In order to find all the roots of $\cos(x^2)$ from the interval 0-10 we will use the code provided from https://rosettacode.org/wiki/Roots_of_a_function#C++

I will change the code's f function to use the $\cos(x^2)$

```
double f(double x)
{
    return (cos(x*x));
}
```

And in order to find all the roots from 0 – 10 we will have to change the start and end values

```
double step = 0.001; // Smaller step values produce more roots
double start = 0; // Refactored to start from 0
double stop = 10; // Refactored to end at 10
double value = f(start);
double sign = (value > 0);
int count=0;
```

I had to refactor the code to keep count of the number of roots

```
if ( ( value > 0 ) != sign )
{
    count++; // Keeps count of the roots
    // We passed a root
    std::cout << "Root "<<count<<" found near " << x << std::endl;
}
else if ( 0 == value )
{
    count++; // Keeps count of the roots
    // We hit a root
    std::cout << "Root "<<count<<" found near " << x << std::endl;
}
// Update our sign
sign = ( value > 0 );
```

These are the results

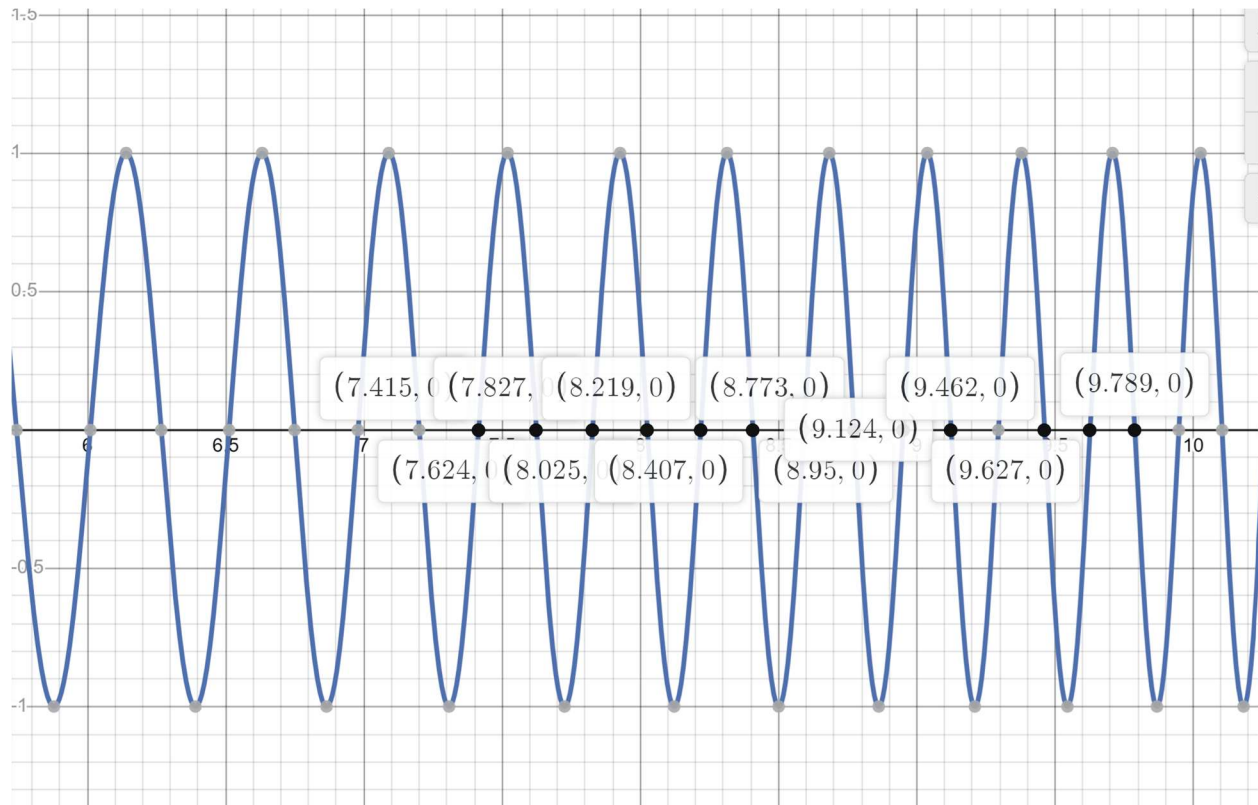
It seems that there are in total 32 roots

```
juan@DESKTOP-QCQU6MF:$ ./a.out
```

```
Root 1 found near 1.254  
Root 2 found near 2.171  
Root 3 found near 2.803  
Root 4 found near 3.316  
Root 5 found near 3.76  
Root 6 found near 4.157  
Root 7 found near 4.519  
Root 8 found near 4.855  
Root 9 found near 5.168  
Root 10 found near 5.464  
Root 11 found near 5.744  
Root 12 found near 6.011  
Root 13 found near 6.267  
Root 14 found near 6.513  
Root 15 found near 6.75  
Root 16 found near 6.979  
Root 17 found near 7.2  
Root 18 found near 7.415  
Root 19 found near 7.624  
Root 20 found near 7.827  
Root 21 found near 8.026  
Root 22 found near 8.219  
Root 23 found near 8.408  
Root 24 found near 8.593  
Root 25 found near 8.774  
Root 26 found near 8.951  
Root 27 found near 9.125  
Root 28 found near 9.295  
Root 29 found near 9.463  
Root 30 found near 9.627  
Root 31 found near 9.789  
Root 32 found near 9.948
```

```
juan@DESKTOP-QCQU6MF:$
```

Here are a few roots found by desmos



Find roots from [-10 – 10]

In order to find the roots from -10 to 10 we will need to change the code slightly just modifying the beginning value to -10

```
double step = 0.001; // Smaller step values produce
double start = -10; // Refactored to start from -10
double stop = 10; // Refactored to end at 10
double value = f(start);
double sign = (value > 0);
```

These are the results and it has found 62 total roots

```
juan@DESKTOP-QCQU6MF:~$ ./a.out
Root 1 found near -9.947
Root 2 found near -9.788
Root 3 found near -9.626
Root 4 found near -9.462
Root 5 found near -9.294
Root 6 found near -9.124
Root 7 found near -8.95
Root 8 found near -8.773
Root 9 found near -8.592
Root 10 found near -8.407
Root 11 found near -8.218
Root 12 found near -8.025
Root 13 found near -7.826
Root 14 found near -7.623
Root 15 found near -7.414
Root 16 found near -7.199
Root 17 found near -6.978
Root 18 found near -6.749
Root 19 found near -6.512
Root 20 found near -6.266
Root 21 found near -6.01
Root 22 found near -5.743
Root 23 found near -5.463
Root 24 found near -5.167
Root 25 found near -4.854
Root 26 found near -4.518
Root 27 found near -4.156
Root 28 found near -3.759
Root 29 found near -3.315
Root 30 found near -2.802
Root 31 found near -2.17
Root 32 found near -1.253
Root 33 found near 1.254
Root 34 found near 2.171
Root 35 found near 2.803
Root 36 found near 3.316
Root 37 found near 3.76
Root 38 found near 4.157
Root 39 found near 4.519
Root 40 found near 4.855
Root 41 found near 5.168
Root 42 found near 5.464
Root 43 found near 5.744
Root 44 found near 6.011
Root 45 found near 6.267
Root 46 found near 6.513
Root 47 found near 6.75
Root 48 found near 6.979
Root 49 found near 7.2
Root 50 found near 7.415
Root 51 found near 7.624
Root 52 found near 7.827
Root 53 found near 8.026
Root 54 found near 8.219
Root 55 found near 8.408
Root 56 found near 8.593
Root 57 found near 8.774
Root 58 found near 8.951
Root 59 found near 9.125
Root 60 found near 9.295
Root 61 found near 9.463
Root 62 found near 9.627
Root 63 found near 9.789
Root 64 found near 9.948
juan@DESKTOP-QCQU6MF:~$
```

Compare seq vs OpenMP

Now we can compare on a much larger interval how much faster will openmp be compared to our sequential program

Here we can see that openmp found all 6366 roots in 0.006s

```
juan@DESKTOP-QCQU6MF:$ time ./multi_root_omp
Total roots found is 6366

real    0m0.006s
user    0m0.004s
sys     0m0.000s
```

And for our sequential run it seems that we have a total time of 0.009s

```
juan@DESKTOP-QCQU6MF:$ time ./multi_root

real    0m0.009s
user    0m0.005s
sys     0m0.000s
```

Part3)

The values we know we are supposed to get are in the excel sheet

For our **non linear** train at 1800 the given final

Acceleration is -0.003889

Velocity is 0.0003497

Position is 122000.9292

1797	-0.013778633	12.5372093	0.0219085	122000.9132
1798	-0.010482768	12.54418605	0.0114257	122000.9246
1799	-0.007186393	12.55116279	0.0042393	122000.9288
1800	-0.003889668	12.55813953	0.0003497	122000.9292

2-Train-Analysis Linear-Train Non-Linear-Train +

17997	17995,	-0.00553803044196,	0.00437451859765,	122000.92694276537804
17998	17996,	-0.00520835793749,	0.00385368280390,	122000.92732813366456
17999	17997,	-0.00487868543302,	0.00336581426060,	122000.92766471508367
18000	17998,	-0.00454901292855,	0.00291091296775,	122000.92795580638631
18001	17999,	-0.00421934042408,	0.00248897892534,	122000.92820470427978

At 1800 we can see that our final values are for **Linear**

Acceleration is -0.0001232

Velocity is -8.41E-12

Position is 122001.65999

```
17997 17995, -0.00061617000000, 0.00012323399159, 122001.65998766390840
17998 17996, -0.00049293600000, 0.00007394039159, 122001.65999505795480
17999 17997, -0.00036970200000, 0.00003697019159, 122001.65999875497073
18000 17998, -0.00024646800000, 0.00001232339159, 122001.65999998731422
18001 17999, -0.00012323400000, -0.00000000000841, 122001.65999998731422
```

Using my sequential program i will try to find the intersection between the two trains

It seems to be at about 737s where we are able to see both trains at roughly the same values in terms of position

```
7371, 58754.63631592639285
7372, 58758.57859460174950
7373, 58762.51658094143932
```

This can be seen from posintersect.csv

In terms of velocity if we look at the time of 737.1 – 737.3 the velocity for the non linear train will be

```
7371 7369, -0.42978478244111, 39.55163981970228,
7372 7370, -0.42964854705413, 39.50867496499687,
7373 7371, -0.42951022036308, 39.46572394296057,
```

The velocity is at about 39.5

For the Linear train

```
7373 7371, -0.12323400000000, 124.81755689998822,
7374 7372, -0.12323400000000, 124.80523349998822,
7375 7373, -0.12323400000000, 124.79291009998822,
```

The velocity is at about 124.8

These values can be seen from the Nonlinvalues.csv file(non linear) and Linvalues.csv(linear) that my program produces