**Assignment - 2**

**Ashutosh Jha**

**(18312911011)**

Write a Matlab Code for Secant Method and Regula Falsi. Compare these two methods for function

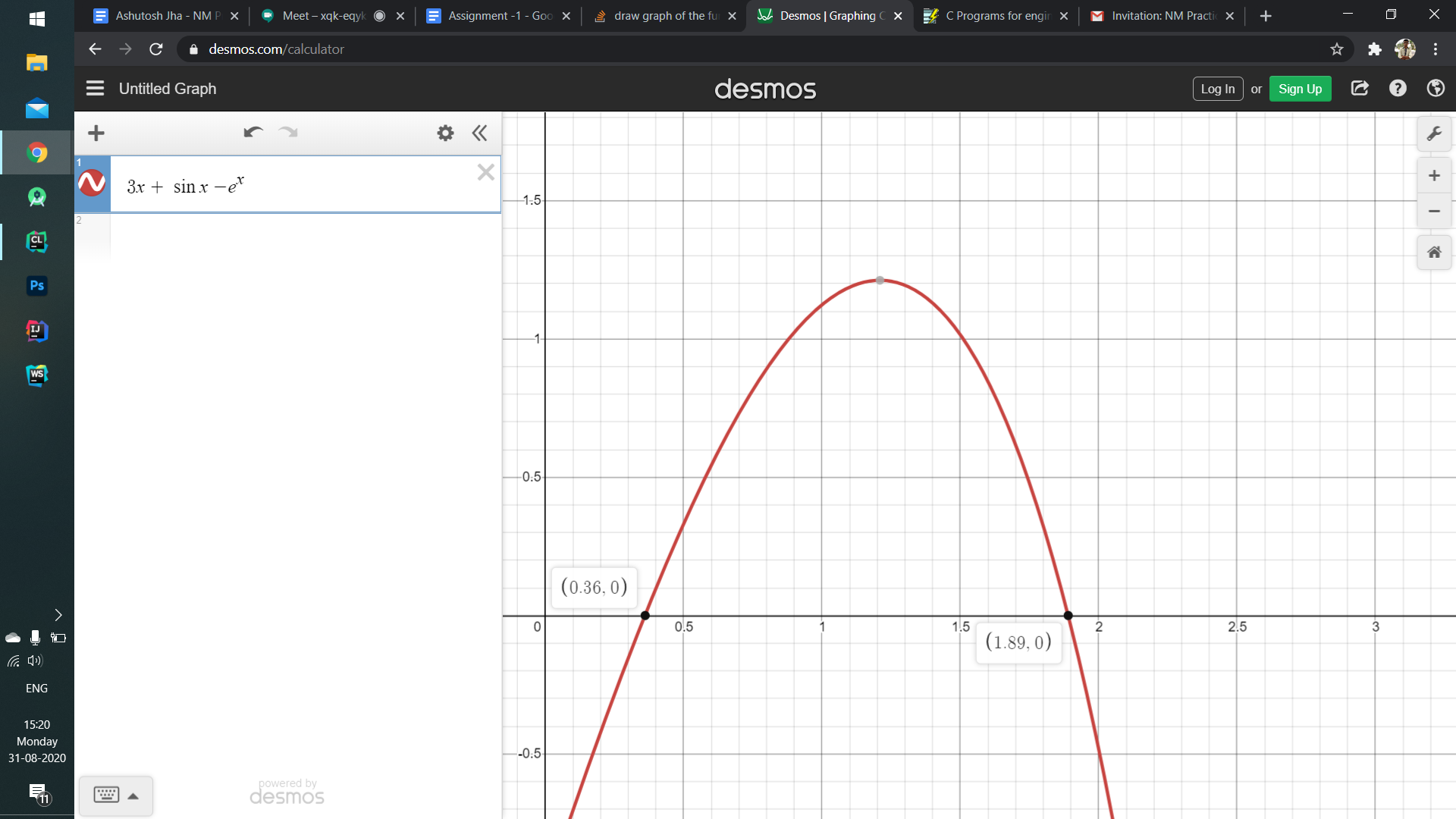
f(x) = 3x + Sin x - ex

Also, write the code for hybrid method for bisection and newton method

Plot your solution for these methods and write a note on your observation regarding convergence of these methods.

Plot of f(x) = 3x + Sin x + ex

(Plotted using Desmos)



**Regula Falsi Method**

For root in interval (0, 1)

Iterations done = 17

Max Error or Epsilon = 0.0000000000000001

r = 0.36042170 (Found using calculator for finding error)

Program

#include<bits/stdc++.h>

#include <iomanip>

using namespace std;

#define **MAX\_ERROR** 0.0000000000000001

int iterations = 0;

double values[30];

double actualError[30];

double functionValueForC[30];

double answer();

double func(double x)

{

return sin(x) + 3\*x - exp(x);

}

// Prints root of func(x) in interval [a, b]

void regulaFalsi(double a, double b)

{

if (func(a) \* func(b) >= 0)

{

cout << "You have not assumed right a and b\n";

return;

}

double c = a; // Initialize result

while (abs(func(c)) >= **MAX\_ERROR**)

{

// Find the point that touches x axis

double multiply = a\*func(b) - b\*func(a);

double divide = (func(b) - func(a));

c = multiply / divide ;

values[iterations] = c;

functionValueForC[iterations] = func(c);

actualError[iterations] = c - answer();

iterations = iterations + 1;

// Check if the above found point is root

if (func(c)==0)

break;

// Decide the side to repeat the steps

else if (func(c)\*func(a) < 0)

b = c;

else

a = c;

}

cout << "The value of root is : " << c;

cout << "The value of the root is : " << c; //Printing the x\_mid for each iteration

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double midValue : values) {

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << midValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForC){

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualError){

std::cout << "\n" << i;

}

}

double answer() {

return 0.36042170;

}

int main()

{

// Initial values assumed

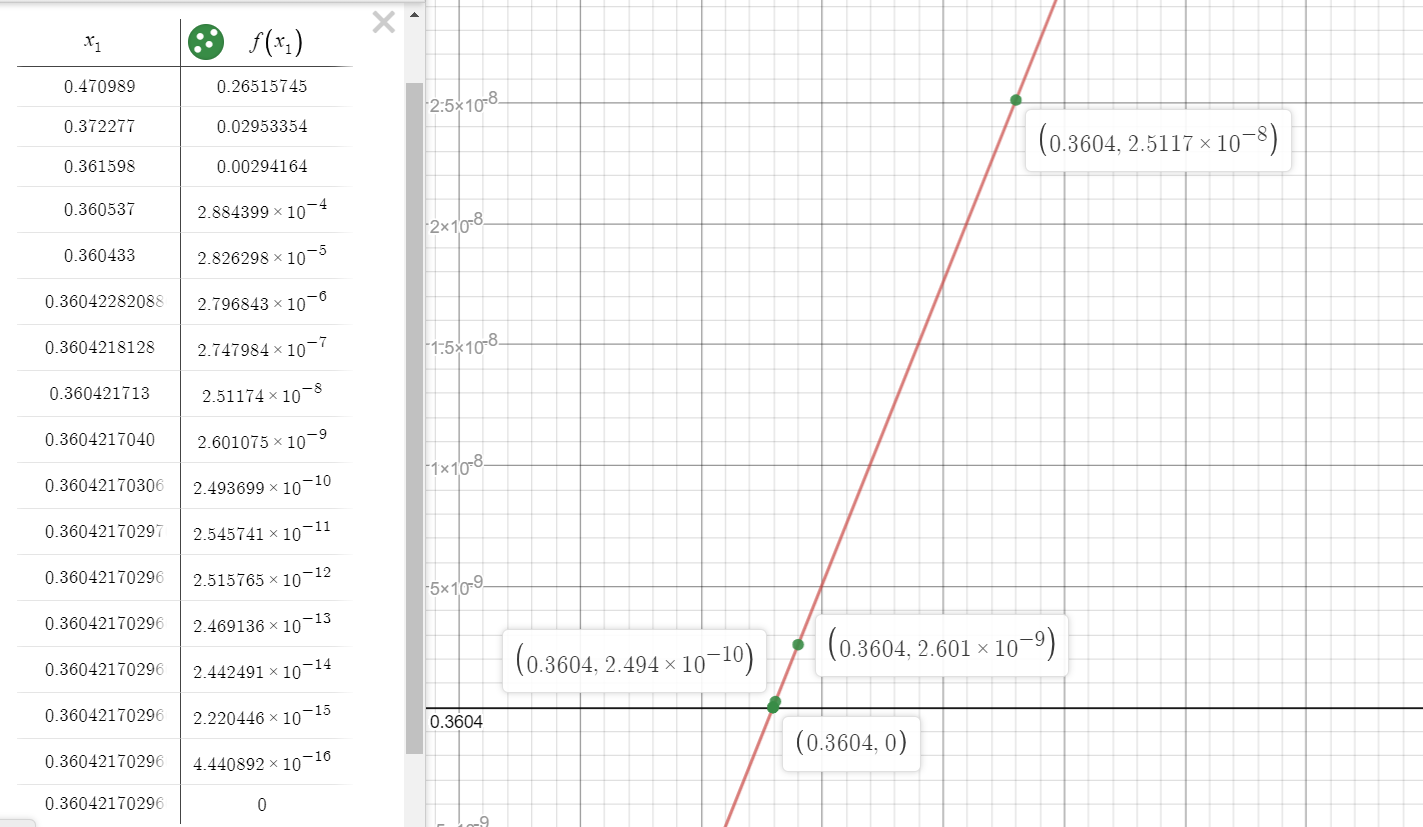
double x0 =0, x1 = 1;

regulaFalsi(x0, x1);

return 0;

}

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 0.47098959459629724522 | 0.26515881591031087439 | 0.11056789459629723371 |
| 2 | 0.37227705223506613041 | 0.02953366933826528928 | 0.01185535223506611890 |
| 3 | 0.36159774402943800631 | 0.00294100010809761336 | 0.00117604402943799480 |
| 4 | 0.36053740348681007077 | 0.00028944926169227614 | 0.00011570348681005926 |
| 5 | 0.36043307639895694194 | 0.00002845411476659621 | 0.00001137639895693043 |
| 6 | 0.36042282088664695516 | 0.00000279684283666093 | 0.00000112088664694365 |
| 7 | 0.36042181284348151493 | 0.00000027490723253365 | 0.00000011284348150342 |
| 8 | 0.36042171376094561674 | 0.00000002702114754172 | 0.00000001376094560523 |
| 9 | 0.36042170402193757228 | 0.00000000265595900828 | 0.00000000402193756077 |
| 10 | 0.36042170306467230523 | 0.00000000026105895223 | 0.00000000306467229372 |
| 11 | 0.36042170297058101491 | 0.00000000002566014068 | 0.00000000297058100340 |
| 12 | 0.36042170296133252405 | 0.00000000000252220467 | 0.00000000296133251254 |
| 13 | 0.36042170296042347344 | 0.00000000000024780178 | 0.00000000296042346193 |
| 14 | 0.36042170296033415600 | 0.00000000000002442491 | 0.00000000296033414449 |
| 15 | 0.36042170296032532972 | 0.00000000000000222045 | 0.00000000296032531821 |
| 16 | 0.36042170296032455257 | 0.00000000000000044409 | 0.00000000296032454106 |
| 17 | 0.36042170296032438603 | 0.00000000000000000000 | 0.00000000296032437452 |



For root in interval (1, 2)

Iterations done = 17

Max Error or Epsilon = 0.0000000000000001

r = 0.36042170 (Found using calculator for finding error)

Program

#include<bits/stdc++.h>

#include <iomanip>

using namespace std;

#define **MAX\_ERROR** 0.0000000000000001

int iterations = 0;

double values[30];

double actualError[30];

double functionValueForC[30];

double answer();

double func(double x)

{

return sin(x) + 3\*x - exp(x);

}

// Prints root of func(x) in interval [a, b]

void regulaFalsi(double a, double b)

{

if (func(a) \* func(b) >= 0)

{

cout << "You have not assumed right a and b\n";

return;

}

double c = a; // Initialize result

while (abs(func(c)) >= **MAX\_ERROR**)

{

// Find the point that touches x axis

double multiply = a\*func(b) - b\*func(a);

double divide = (func(b) - func(a));

c = multiply / divide ;

values[iterations] = c;

functionValueForC[iterations] = func(c);

actualError[iterations] = c - answer();

iterations = iterations + 1;

// Check if the above found point is root

if (func(c)==0)

break;

// Decide the side to repeat the steps

else if (func(c)\*func(a) < 0)

b = c;

else

a = c;

}

cout << "The value of root is : " << c;

cout << "The value of the root is : " << c; //Printing the x\_mid for each iteration

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double midValue : values) {

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << midValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForC){

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualError){

std::cout << "\n" << i;

}

}

double answer() {

return 1.89003;

}

int main()

{

// Initial values assumed

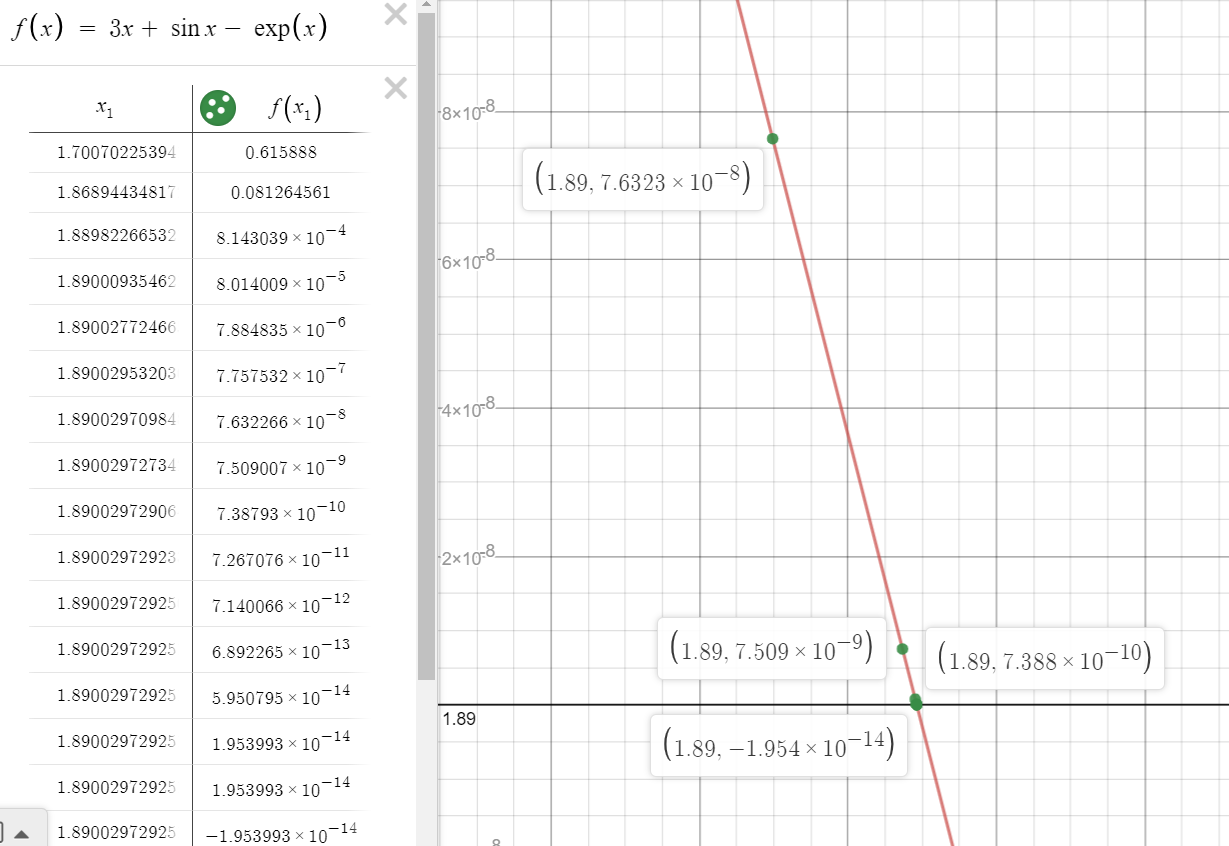
double x0 =1, x1 = 2;

regulaFalsi(x0, x1);

return 0;

}

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 1.70070225394189 | 0.61588800326223 | -0.18932774605811 |
| 2 | 1.86894434817253 | 0.08126456077252 | -0.02108565182747 |
| 3 | 1.88792784004664 | 0.00825086992854 | -0.00210215995336 |
| 4 | 1.88982266532102 | 0.00081430389427 | -0.00020733467898 |
| 5 | 1.89000935462144 | 0.00008014008753 | -0.00002064537856 |
| 6 | 1.89002772466514 | 0.00000788483523 | -0.00000227533486 |
| 7 | 1.89002953203011 | 0.00000077575320 | -0.00000046796989 |
| 8 | 1.89002970984827 | 0.00000007632264 | -0.00000029015173 |
| 9 | 1.89002972734295 | 0.00000000750902 | -0.00000027265705 |
| 10 | 1.89002972906416 | 0.00000000073878 | -0.00000027093584 |
| 11 | 1.89002972923351 | 0.00000000007268 | -0.00000027076649 |
| 12 | 1.89002972925017 | 0.00000000000715 | -0.00000027074983 |
| 13 | 1.89002972925181 | 0.00000000000070 | -0.00000027074819 |
| 14 | 1.89002972925197 | 0.00000000000007 | -0.00000027074803 |
| 15 | 1.89002972925198 | 0.00000000000001 | -0.00000027074802 |
| 16 | 1.89002972925198 | 0.00000000000000 | -0.00000027074802 |
| 17 | 1.89002972925199 | 0.00000000000000 | -0.00000027074801 |



**Secant Method**

For root in interval (0, 1)

Iterations done = 17

Max Error or Epsilon = 0.0000000000000001

r = 0.36042170 (Found using calculator for finding error)

x1 = 0.8

x0 = 1

Program

#include<bits/stdc++.h>

#include <iomanip>

using namespace std;

#define **MAX\_ERROR** 0.0000000000000001

int iterations = 0;

double values[30];

double actualError[30];

double functionValueForC[30];

double answer();

double func(double x)

{

return sin(x) + 3\*x - exp(x);

}

// Prints root of func(x) in interval [a, b]

void secant(double a, double b)

{

double c = a; // Initialize result

while (abs(func(c)) >= **MAX\_ERROR**)

{

// Find the point that touches x axis

double multiply = a\*func(b) - b\*func(a);

double divide = (func(b) - func(a));

c = multiply / divide ;

values[iterations] = c;

functionValueForC[iterations] = func(c);

actualError[iterations] = c - answer();

iterations = iterations + 1;

// Check if the above found point is root

if (func(c)==0)

break;

// Decide the side to repeat the steps

else if (func(c)\*func(a) < 0)

b = c;

else

a = c;

}

cout << "The value of root is : " << c;

cout << "The value of the root is : " << c; //Printing the x\_mid for each iteration

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double midValue : values) {

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << midValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForC){

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualError){

std::cout << "\n" << i;

}

}

double answer() {

return 0.36042170;

}

int main()

{

// Initial values assumed

double x0 =1, x1 = 0.8;

if (func(x0) < func(x1)) {

double temp = x0;

x0 = x1;

x1 = temp;

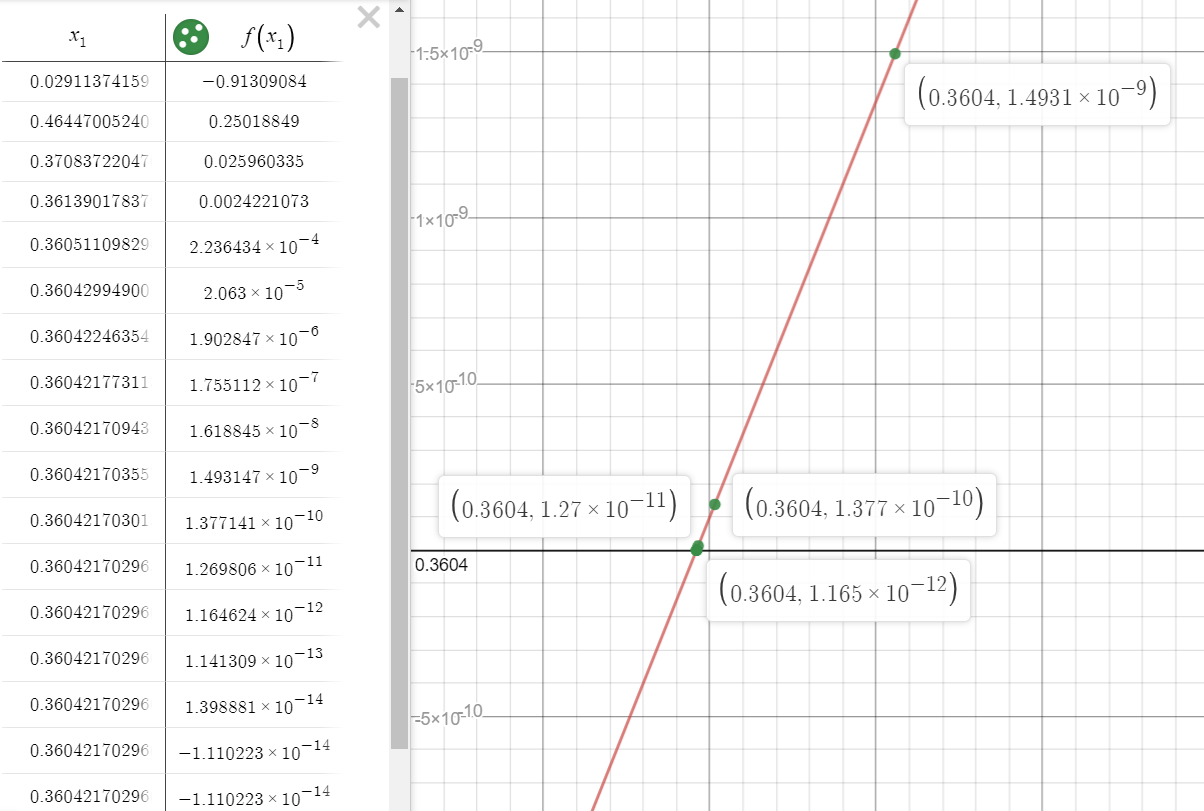
}

secant(x0, x1);

return 0;

}

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 0.02911374159760 | -0.91309083581709 | -0.33130795840240 |
| 2 | 0.46447005240877 | 0.25018848882273 | 0.10404835240877 |
| 3 | 0.37083722047048 | 0.02596033484279 | 0.01041552047048 |
| 4 | 0.36139017837911 | 0.00242210733294 | 0.00096847837911 |
| 5 | 0.36051109829606 | 0.00022364338160 | 0.00008939829606 |
| 6 | 0.36042994900171 | 0.00002063000273 | 0.00000824900171 |
| 7 | 0.36042246354729 | 0.00000190284675 | 0.00000076354729 |
| 8 | 0.36042177311389 | 0.00000017551117 | 0.00000007311389 |
| 9 | 0.36042170943101 | 0.00000001618845 | 0.00000000943101 |
| 10 | 0.36042170355715 | 0.00000000149316 | 0.00000000355715 |
| 11 | 0.36042170301537 | 0.00000000013772 | 0.00000000301537 |
| 12 | 0.36042170296540 | 0.00000000001270 | 0.00000000296540 |
| 13 | 0.36042170296079 | 0.00000000000117 | 0.00000000296079 |
| 14 | 0.36042170296037 | 0.00000000000011 | 0.00000000296037 |
| 15 | 0.36042170296033 | 0.00000000000001 | 0.00000000296033 |
| 16 | 0.36042170296032 | 0.00000000000000 | 0.00000000296032 |
| 17 | 0.36042170296032 | 0.00000000000000 | 0.00000000296032 |



For root in interval (1, 2)

Iterations done = 30

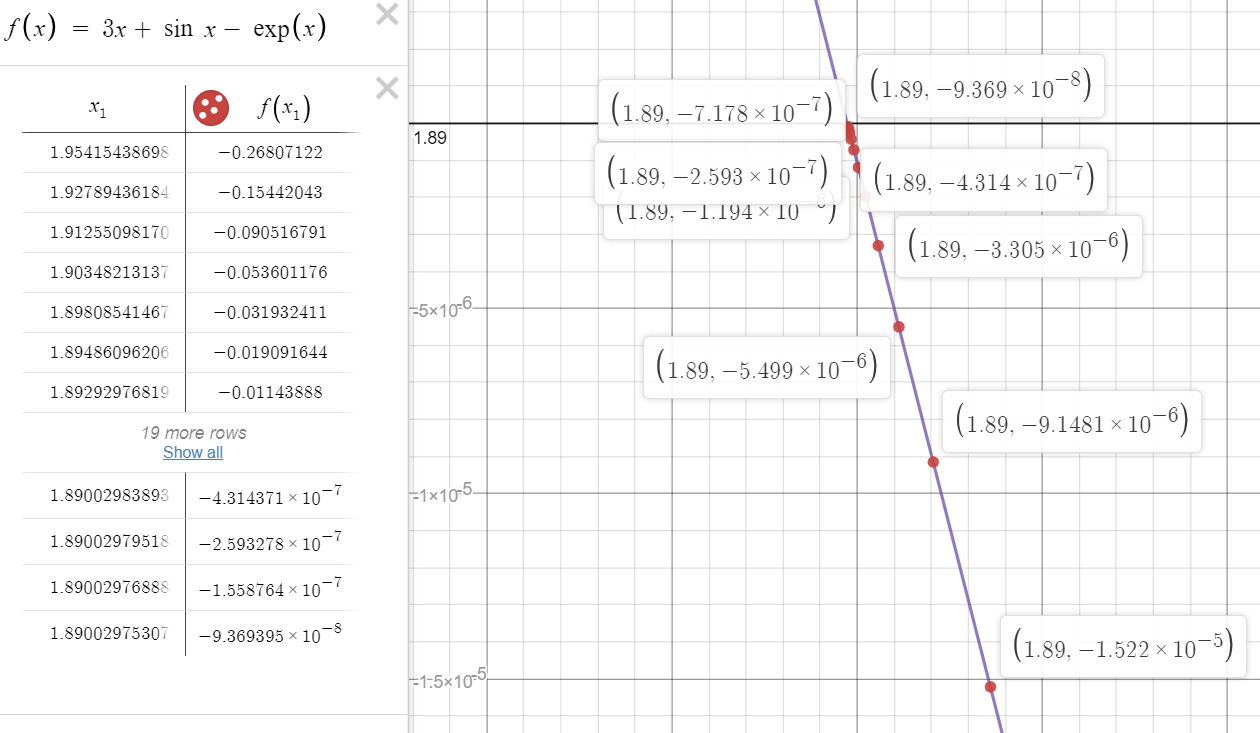
Max Error or Epsilon = 0.0000000000000001

r = 1.89003 (Found using calculator for finding error)

x0 = 3

x1 = 2

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 1.95415438698872 | -0.26807121717011 | 0.06412438698872 |
| 2 | 1.92789436184682 | -0.15442043381555 | 0.03786436184682 |
| 3 | 1.91255098170518 | -0.09051679119669 | 0.02252098170518 |
| 4 | 1.90348213137469 | -0.05360117591653 | 0.01345213137469 |
| 5 | 1.89808541467511 | -0.03193241071128 | 0.00805541467511 |
| 6 | 1.89486096206828 | -0.01909164416293 | 0.00483096206828 |
| 7 | 1.89292976819171 | -0.01143887984324 | 0.00289976819171 |
| 8 | 1.89177147048739 | -0.00686245890885 | 0.00174147048739 |
| 9 | 1.89107614386974 | -0.00412011690733 | 0.00104614386974 |
| 10 | 1.89065852303067 | -0.00247479671875 | 0.00062852303067 |
| 11 | 1.89040761742200 | -0.00148692742187 | 0.00037761742200 |
| 12 | 1.89025684579661 | -0.00089353640924 | 0.00022684579661 |
| 13 | 1.89016623549873 | -0.00053700477569 | 0.00013623549873 |
| 14 | 1.89011177711317 | -0.00032275288408 | 0.00008177711317 |
| 15 | 1.89007904533842 | -0.00019398930882 | 0.00004904533842 |
| 16 | 1.89005937168956 | -0.00011659901563 | 0.00002937168956 |
| 17 | 1.89004754654032 | -0.00007008379924 | 0.00001754654032 |
| 18 | 1.89004043878998 | -0.00004212537641 | 0.00001043878998 |
| 19 | 1.89003616650716 | -0.00002532048376 | 0.00000616650716 |
| 20 | 1.89003359854155 | -0.00001521953673 | 0.00000359854155 |
| 21 | 1.89003205499675 | -0.00000914811482 | 0.00000205499675 |
| 22 | 1.89003112720655 | -0.00000549872784 | 0.00000112720655 |
| 23 | 1.89003056953227 | -0.00000330516473 | 0.00000056953227 |
| 24 | 1.89003023432639 | -0.00000198666278 | 0.00000023432639 |
| 25 | 1.89003003284137 | -0.00000119414014 | 0.00000003284137 |
| 26 | 1.89002991173306 | -0.00000071777198 | -0.00000008826694 |
| 27 | 1.89002983893746 | -0.00000043143734 | -0.00000016106254 |
| 28 | 1.89002979518158 | -0.00000025932775 | -0.00000020481842 |
| 29 | 1.89002976888085 | -0.00000015587636 | -0.00000023111915 |
| 30 | 1.89002975307205 | -0.00000009369395 | -0.00000024692795 |



**Hybrid Method (Combo of Bisection and Newton Rapson)**

For root in interval (1, 2)

Iterations done = 3 (bisection) + 4(Newton)

Max Error or Epsilon = 0.0000000000000001

r = 1.89003 (Found using calculator for finding error)

x0 = 1

x1 = 2

Program

// Bisection Method

#include<bits/stdc++.h>

// Maximum permissible error

#define MAX\_ERROR\_FOR\_BISECTION 0.001

#define MAX\_ERROR\_FOR\_NEWTON 0.00000000001

using namespace std;

//No. of iterations done

int iterations = 0;

int iteration = 0;

double rootValuesBisection[3];

double actualErrorBisection[3];

double functionValueForCBisection[3];

double rootValuesNewton[6];

double actualErrorNewton[6];

double functionValueForCNewton[6];

double deriv(double x);

//Function given in question

double func(double x)

{

return 3\*x - exp(x) + sin(x);

}

double answer() {

//Actual root found using calculator;

return 1.89003;

}

void newtonRaphson(double x)

{

double h = func(x) / deriv(x);

while (abs(h) >= MAX\_ERROR\_FOR\_NEWTON && iteration < 6)

{

h = func(x)/deriv(x);

// x(i+1) = x(i) - f(x) / f'(x)

x = x - h;

rootValuesNewton[iteration] = x;

iteration = iteration + 1;

functionValueForCNewton[iteration] = func(x);

actualErrorNewton[iteration] = x - answer();

}

cout << "The value of the root is : " << x; //Printing the x\_mid for each iteration

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double midValue : rootValuesNewton) {

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << midValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForCNewton){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualErrorNewton){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

}

double deriv(double x) {

return 3 - exp(x) + cos(x);

}

//Function to find the root using bisection Method

float bisectionMethod(double x\_lower, double x\_mid, double x\_upper)

{

while (abs(func(x\_mid)) >= MAX\_ERROR\_FOR\_BISECTION && iterations < 3)

{

if(func(x\_mid) \* func(x\_upper) < 0) {

x\_lower = x\_mid;

} else if (func(x\_mid) \* func(x\_lower) < 0){

x\_upper = x\_mid;

} else {

break;

}

rootValuesBisection[iterations] = x\_mid;

functionValueForCBisection[iterations] = func(x\_mid);

actualErrorBisection[iterations] = x\_mid - answer();

iterations = iterations + 1;

x\_mid = (x\_upper+x\_lower) / 2;

}

//Printing the root

std::cout << "The value of the root is : " << x\_mid;

//No. of iterations done for maximum error

std::cout << "\nThe value of the iterations is : " << iterations;

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double rootValue : rootValuesBisection) {

std::cout << "\n" << rootValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForCBisection){

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualErrorBisection){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

//Printing the last root using bisection

float root = rootValuesBisection[2];

std::cout << "Root using Bisection \n" << root;

return root;

}

int main()

{

double x\_lower = 1;

double x\_upper = 2;

double x\_mid = (x\_lower + x\_upper) / 2;

// Initial values assumed

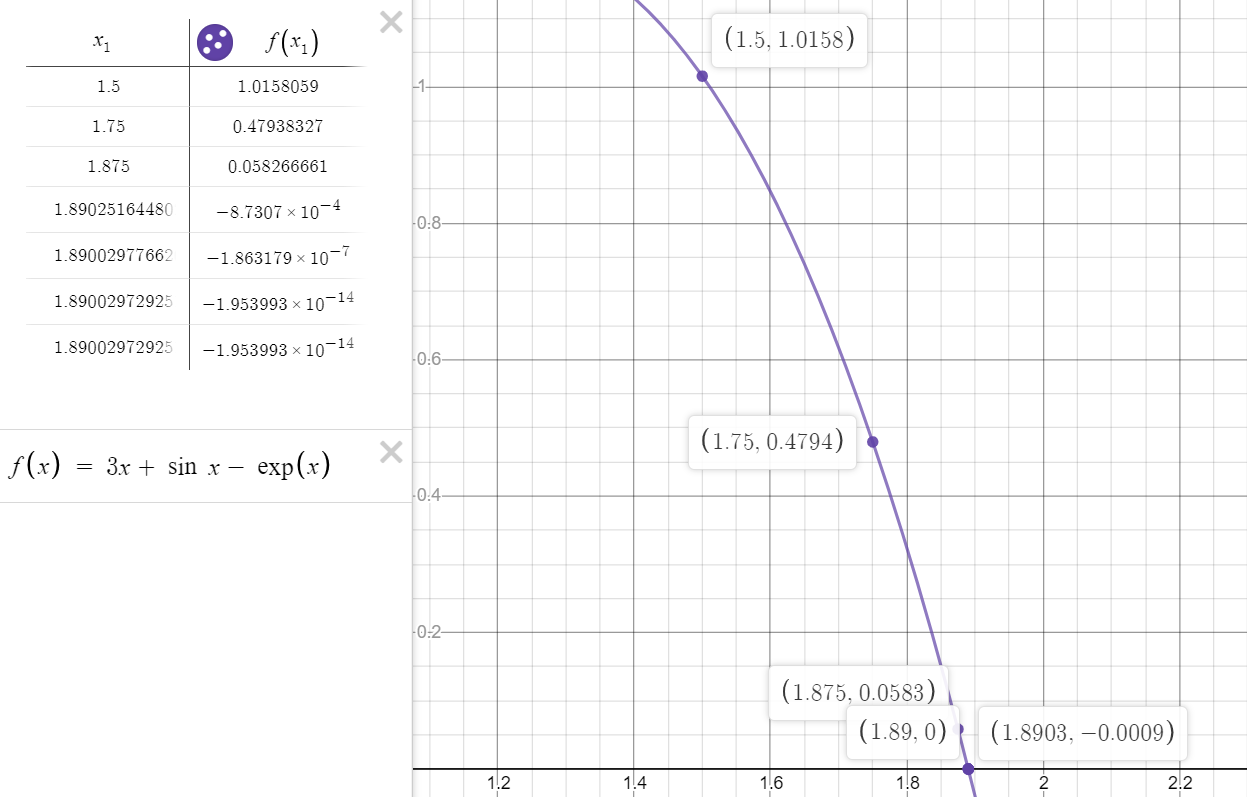
float x0 = bisectionMethod(x\_lower, x\_mid, x\_upper);

newtonRaphson(x0);

return 0;

}

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 1.5 | 1.01581 | -0.39003000000000 |
| 2 | 1.75 | 0.479383 | -0.14003000000000 |
| 3 | 1.875 | 0.0582667 | -0.01503000000000 |
| 4 | 1.89025164480240 | 0.00000000000000 | 0.00022164480240 |
| 5 | 1.89002977662008 | -0.00087306995719 | -0.00000022337992 |
| 6 | 1.89002972925199 | -0.00000018631786 | -0.00000027074801 |
| 7 | 1.89002972925199 | -0.00000000000001 | -0.00000027074801 |

****

For root in interval (0, 1)

Iterations done = 17

Max Error or Epsilon = 0.0000000000000001

r = 0.36042170 (Found using calculator for finding error)

x1 = 0.8

x0 = 1

Program

// Bisection Method

#include<bits/stdc++.h>

// Maximum permissible error

#define **MAX\_ERROR\_FOR\_BISECTION** 0.001

#define **MAX\_ERROR\_FOR\_NEWTON** 0.00000000001

using namespace std;

//No. of iterations done

int iterations = 0;

int iteration = 0;

double rootValuesBisection[3];

double actualErrorBisection[3];

double functionValueForCBisection[3];

double rootValuesNewton[6];

double actualErrorNewton[6];

double functionValueForCNewton[6];

double deriv(double x);

//Function given in question

double func(double x)

{

return 3\*x - exp(x) + sin(x);

}

double answer() {

//Actual root found using calculator;

return 0.36042170;

}

void newtonRaphson(double x)

{

double h = func(x) / deriv(x);

while (abs(h) >= **MAX\_ERROR\_FOR\_NEWTON** && iteration < 6)

{

h = func(x)/deriv(x);

// x(i+1) = x(i) - f(x) / f'(x)

x = x - h;

rootValuesNewton[iteration] = x;

iteration = iteration + 1;

functionValueForCNewton[iteration] = func(x);

actualErrorNewton[iteration] = x - answer();

}

cout << "The value of the root is : " << x; //Printing the x\_mid for each iteration

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double midValue : rootValuesNewton) {

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << midValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForCNewton){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualErrorNewton){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

}

double deriv(double x) {

return 3 - exp(x) + cos(x);

}

//Function to find the root using bisection Method

float bisectionMethod(double x\_lower, double x\_mid, double x\_upper)

{

while (abs(func(x\_mid)) >= **MAX\_ERROR\_FOR\_BISECTION** && iterations < 3)

{

if(func(x\_mid) \* func(x\_upper) < 0) {

x\_lower = x\_mid;

} else if (func(x\_mid) \* func(x\_lower) < 0){

x\_upper = x\_mid;

} else {

break;

}

rootValuesBisection[iterations] = x\_mid;

functionValueForCBisection[iterations] = func(x\_mid);

actualErrorBisection[iterations] = x\_mid - answer();

iterations = iterations + 1;

x\_mid = (x\_upper+x\_lower) / 2;

}

//Printing the root

std::cout << "The value of the root is : " << x\_mid;

//No. of iterations done for maximum error

std::cout << "\nThe value of the iterations is : " << iterations;

//Printing the x\_mid for each iteration

std::cout << "\nx\_mid values";

for(double rootValue : rootValuesBisection) {

std::cout << "\n" << rootValue;

}

//Printing the f(c) value

std::cout << "\nf(x\_mid)";

for (double i : functionValueForCBisection){

std::cout << "\n" << i;

}

//Printing the actual error

std::cout << "\nActual error";

for (double i : actualErrorBisection){

std::cout << std::fixed;

std::cout << std::setprecision(14);

std::cout << "\n" << i;

}

//Printing the last root using bisection

float root = rootValuesBisection[2];

std::cout << "Root using Bisection \n" << root;

return root;

}

int main()

{

double x\_lower = 0;

double x\_upper = 1;

double x\_mid = (x\_lower + x\_upper) / 2;

// Initial values assumed

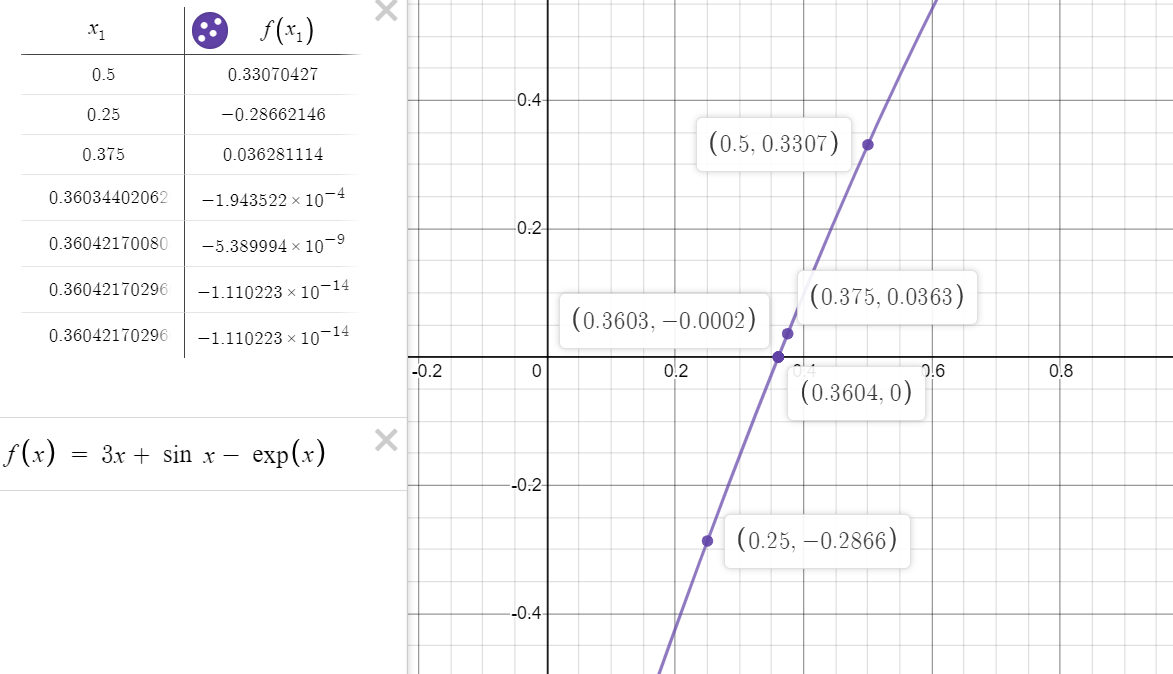
float x0 = bisectionMethod(x\_lower, x\_mid, x\_upper);

newtonRaphson(x0);

return 0;

}

| Iterations | c | f(c) | Actual Error |
| --- | --- | --- | --- |
| 1 | 0.5 | 0.330704 | 0.13957830000000 |
| 2 | 0.25 | -0.286621 | -0.11042170000000 |
| 3 | 0.375 | 0.0362811 | 0.01457830000000 |
| 4 | 0.36034402062853 | -0.00019435215158 | -0.00007767937147 |
| 5 | 0.36042170080589 | -0.00000000539000 | 0.00000000080589 |
| 6 | 0.36042170296032 | -0.00000000000000 | 0.00000000296032 |
| 7 | 0.36042170296032 | -0.00000000000000 | 0.00000000296032 |



**Convergence Analysis**

Regula Falsi - It is converging in this case as the actual error keeps on decreasing till zero and the value of c is approaching towards the root i.e. 0.360422 and 1.89003. We did 17 iterations for allowing a maximum error of 0.0000000000000001

Secant Method - It is also converging in this case as the actual error keeps on decreasing till zero and the value of c is approaching towards the root i.e. 0.360422 and 1.8900. We needed to do 30 iterations for allowing a maximum error of 0.0000000000000001

Both these methods are similar but the former is bracketed method and the latter is non - bracketed method.

Hybrid Method - A combination of bisection and newton method improves the accuracy. 7 iterations were only needed to allow a maximum error of 0.0000000000000001. It is initially slower as we use bisection method to find the range but converges faster once we get the bracketed range of root

**Conclusion**

I conclude from above tables and graphs that :

1. Secant method and regula falsi method both are superlinear.
2. Secant is non - bracketed while regula falsi is bracketed.
3. We need to choose x0 and x1 closely to root otherwise for this case the values starts to oscillate
4. Hybrid of bisection and newton is faster as bisection gives a bracketed range and it serves as a good x0 for Newton
5. So, from both assignments I conclude that
6. Newton is quadratic
7. Bisection is linear
8. Secant and regula falsi is superlinear
9. Hybrid is also somewhat superlinear