BISECTION METHOD

Find the roots of the following Transcendental equations, correct upto 4 decimal places using Bisection Method.

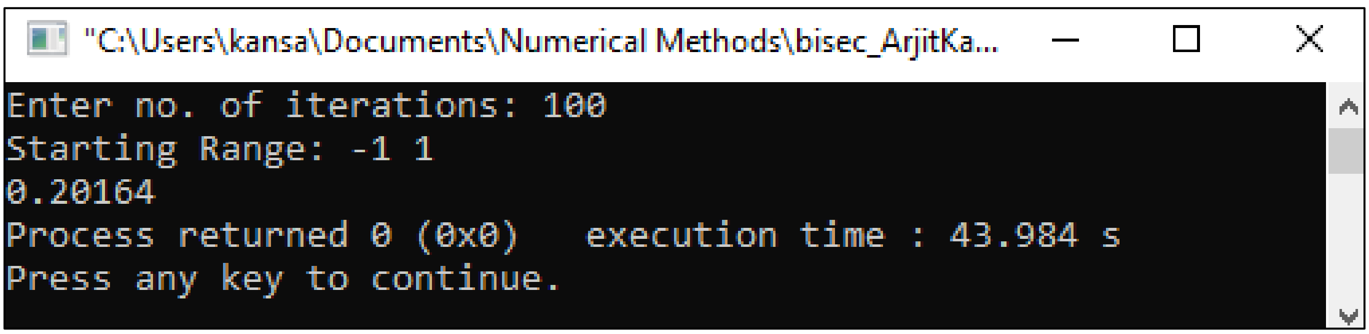
$$f(x) = x^3 - 5x + 1, \qquad -\infty < x < \infty$$

CODE:

```
#include<iostream>
using namespace std;

double funVal(double x) {
	return (x*x*x) - (5.0*x) + 1.0;
}
void solve(int n,double l,double r) {
	double mid;
	if (funVal(l)==0) {
		cout<<l<<"\n";
		return;
	}
	if (funVal(r)==0) {
		cout<<r<<"\n";
		return;
	}
	if (funVal(l)*funVal(r) > 0) {
		cout<<"Invalid Range\n";
		return;
	}
	for (int i=0;i<n;i++) {
		mid = (l+r)/2.0;
		if (funVal(mid) == 0) {
			break;
		}
		if (funVal(l)*funVal(mid) < 0) {
			r = mid;
		}
		else {
			l = mid;
		}
	}
	cout<<mid;
}
```

```
int main() {

	int n;
	double l,r;

	cout<<"Enter no. of iterations: ";
	cin>>n;

	cout<<"Starting Range: ";
	cin>>l>>r;

	solve(n,l,r);
	return 0;
}
```

OUTPUT:



Required Root = 0.20164

SECANT METHOD

Find the roots of the following Transcendental equations, correct upto 4 decimal places using Secant Method.

$$\mathrm{f(x)} = x^3 + x^2 + x + 7, \qquad -\infty < x < \infty$$

CODE:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

double funVal(double x) {
	return (x*x*x) + (x*x) + x + 7.0;
}
void solve(int n, double x0, double x1) {
	double x;
	for (int i=1; i<=n; i++) {
		x = x1 - (x1-x0)/(funVal(x1)-funVal(x0))*funVal(x1);
		x0 = x1;
		x1 =x;
	}
	cout<<fixed<<setprecision(5)<<x;
}
int main() {
	int n;
	double l,r;

	cout<<"Enter no. of iterations: ";
	cin>>n;

	cout<<"Starting Range: ";
	cin>>l>>r;

	solve(n,l,r);
	return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\secant_ArjitKansal.exe"
Enter no. of iterations: 10
Starting Range: -10 10
-2.10487
Process returned 0 (0x0)   execution time : 5.100 s
Press any key to continue.
```

Required Root = -2.10487

$$f(x) = x - e^{-x}, \qquad -\infty < x < \infty$$

CODE:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

double funVal(double x) {
   return (x - exp(-x));
}
void solve(int n, double x0, double x1) {
        double x;
        for (int i=1; i<=n; i++) {
                x = x1 - (x1-x0)/(funVal(x1)-funVal(x0))*funVal(x1);
                x0 = x1;
                x1 =x;
        }
        cout<<fixed<<setprecision(5)<<x;
}
int main() {
        int n; double l,r;
        cout<<"Enter no. of iterations: ";
        cin>>n;
        cout<<"Starting Range: ";
        cin>>l>>r;
        solve(n,l,r);
        return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\secant_ArjitKansal.exe"
Enter no. of iterations: 10
Starting Range: -10 10
0.56714
Process returned 0 (0x0)   execution time : 2.672 s
Press any key to continue.
```

Required Root = 0.56714

REGULA-FALSI METHOD

Find the roots of the following Transcendental equations, correct upto 4 decimal places using Regula-Falsi Method.

$$\mathrm{f(x)} = x^3 + x^2 + x + 7, \qquad -\infty < x < \infty$$

CODE:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

double funVal(double x) {
	return (x*x*x) + (x*x) + x + 7.0;
}
void solve(int n,double l,double r) {
	double mid;
	if (funVal(l)==0) {
		cout<<l<<"\n";
		return;
	}
	if (funVal(r)==0) {
		cout<<r<<"\n";
		return;
	}
	if (funVal(l)*funVal(r) > 0) {
		cout<<"Invalid Range\n";
		return;
	}
	for (int i=0;i<n;i++) {
		mid = l - ((r-l)/(funVal(r)-funVal(l)))*funVal(l);
		if (funVal(mid) == 0) {
			break;
		}
		if (funVal(l)*funVal(mid) < 0) {
			r = mid;
		}
		else {
			l = mid;
		}
	}
	cout<<fixed<<setprecision(5)<<mid;
}
```

```
int main() {
	int n;
	double l,r;

	cout<<"Enter no. of iterations: ";
	cin>>n;

	cout<<"Starting Range: ";
	cin>>l>>r;

	solve(n,l,r);
	return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\regulaFalsi_ArjitKansal.exe"
Enter no. of iterations: 100
Starting Range: -10 10
-2.10468
Process returned 0 (0x0)   execution time : 3.271 s
Press any key to continue.
```

Required Root = -2.10468

$$f(x) = x - e^{-x}, \qquad -\infty < x < \infty$$

CODE:

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;

double funVal(double x) {
   return (x - exp(-x));
}
void solve(int n,double l,double r) {
	double mid;
	if (funVal(l)==0) {
		cout<<l<<"\n";
		return;
	}
```

```
	if (funVal(r)==0) {
		cout<<r<<"\n";
		return;
	}
	if (funVal(l)*funVal(r) > 0) {
		cout<<"Invalid Range\n";
		return;
	}
	for (int i=0;i<n;i++) {
		mid = l - ((r-l)/(funVal(r)-funVal(l)))*funVal(l);
		if (funVal(mid) == 0) {
			break;
		}
		if (funVal(l)*funVal(mid) < 0) {
			r = mid;
		}
		else {
			l = mid;
		}
	}
	cout<<fixed<<setprecision(5)<<mid;
}
int main() {
	int n;
	double l,r;

	cout<<"Enter no. of iterations: ";
	cin>>n;

	cout<<"Starting Range: ";
	cin>>l>>r;

	solve(n,l,r);
	return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\regulaFalsi_ArjitKansal.exe"
Enter no. of iterations: 100
Starting Range: 0 1
0.56714
Process returned 0 (0x0)   execution time : 4.790 s
Press any key to continue.
```

Required Root = 0.56714

NEWTON-RAPHSON METHOD

Find the roots of the following Transcendental equations, correct upto 4 decimal places using Newton-Raphson Method.

$$f(x) = x.\sin(x) + \cos(x), \qquad Near\ to\ x = \pi$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

double df(double x) {
   return x*cos(x);
}
double f(double x) {
        return x*sin(x) + cos(x);
}
void solve(int n, double x0) {
        for (int i=1; i<=n; i++) {
                double x = x0 - f(x0)/df(x0);
                x0 = x;
        }
        cout<<fixed<<setprecision(5)<<x0;
}
int main() {
        int n;
        double x0;

        cout<<"Enter no. of iterations: ";
        cin>>n;

        cout<<"Starting Range: ";
        cin>>x0;

        solve(n, x0);
        return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\NewtonRaphson_ArjitKansal.exe"
Enter no. of iterations: 10
Starting Range: 3.14
2.79839
Process returned 0 (0x0)   execution time : 12.775 s
Press any key to continue.
```

Required Root = 2.79839

$$f(x) = xe^x - \cos(x)$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

double df(double x) {
   return (x+1.0)*exp(x) + sin(x);
}
double f(double x) {
	return x*exp(x) - cos(x);
}
void solve(int n, double x0) {
	for (int i=1; i<=n; i++) {
		double x = x0 - f(x0)/df(x0);
		x0 = x;
	}
	cout<<fixed<<setprecision(5)<<x0;
}
int main() {
	int n;
	double x0;

	cout<<"Enter no. of iterations: ";
	cin>>n;

	cout<<"Starting Range: ";
	cin>>x0;

	solve(n, x0);
	return 0;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\NewtonRaphson_ArjitKansal.exe"
Enter no. of iterations: 10000
Starting Range: 3.14
0.51776
Process returned 0 (0x0)   execution time : 4.586 s
Press any key to continue.
```

Required Root = 0.51776

GAUSSIAN ELIMINATION METHOD

Solve the following system of equations using Gaussian Elimination Method.

$$2x + 2y + z = 12$$
$$3x + 2y + 2z = 8$$
$$5x + 10y - 8z = 10$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

const int n = 3;
double a[][n+2] = {{2,2,1,12},{3,2,2,8},{5,10,-8,10}};
double ans[n+1];
int main() {
	for (int i=0;i<n-1;i++) {
		for (int j=i+1;j<n;j++) {
			double temp = (a[j][i]/a[i][i]);
			for (int k=i;k<n+1;k++) {
				a[j][k] = a[j][k] - temp*a[i][k];
			}
		}
	}
	cout<<"Auxillary Matrix : \n";
	for (int i=0; i<n; i++) {
		for (int j=0; j<=n; j++)
			cout<<a[i][j]<<" ";
		cout<<"\n";
	}
	cout<<"Solution : \n";
	for (int i=n-1;i>=0;i--) {
		double sum = 0.0;
		for (int j=i+1;j<n;j++) {
			sum += a[i][j]*ans[j];
		}
		sum = a[i][n] - sum;
		ans[i] = sum/a[i][i];
	}
	for (int i=0; i<n; i++)
		cout<<(char)('x'+i)<<" = "<<ans[i]<<"\n";
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Gaussian Elimination.exe"
Auxillary Matrix :
2 2 1 12
0 -1 0.5 -10
0 0 -8 -70
Solution :
x = -12.75
y = 14.375
z = 8.75
```

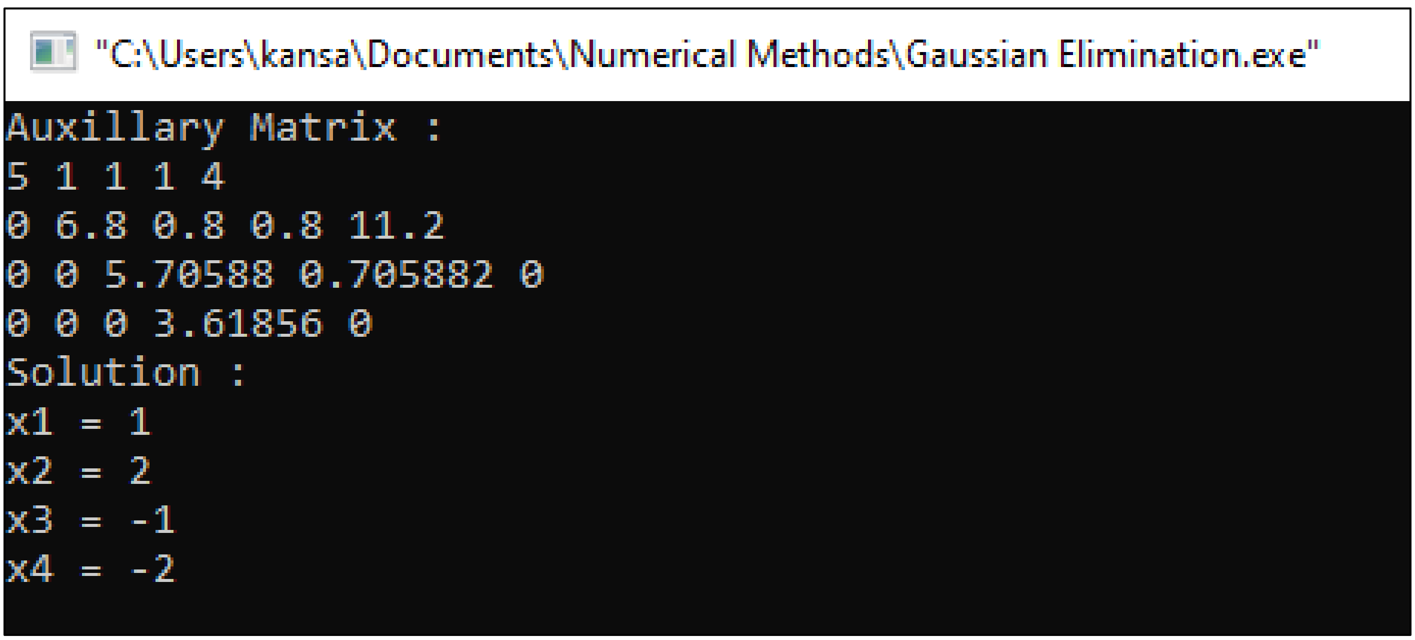
$$
\begin{gathered}
5x_1 + x_2 + x_3 + x_4 = 4 \\
x_1 + 7x_2 + x_3 + x_4 = 12 \\
x_1 + x_2 + 6x_3 + x_4 = -5 \\
x_1 + x_2 + x_3 + 4x_4 = -6
\end{gathered}
$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

const int n = 4;
double a[][n+2] = {{5,1,1,1,4},{1,7,1,1,12},{1,1,6,1,-5},{1,1,1,4,-6}};
double ans[n+1];
int main() {
	for (int i=0;i<n-1;i++) {
		for (int j=i+1;j<n;j++) {
			double temp = (a[j][i]/a[i][i]);
			for (int k=i;k<n+1;k++) {
				a[j][k] = a[j][k] - temp*a[i][k];
			}
		}
	}
	cout<<"Auxillary Matrix : \n";
	for (int i=0; i<n; i++) {
		for (int j=0; j<=n; j++)
			cout<<a[i][j]<<" ";
		cout<<"\n";
	}
```

```
	cout<<"Solution : \n";
	for (int i=n-1;i>=0;i--) {
		double sum = 0.0;
		for (int j=i+1;j<n;j++) {
			sum += a[i][j]*ans[j];
		}
		sum = a[i][n] - sum;
		ans[i] = sum/a[i][i];
	}
	for (int i=0; i<n; i++)
		cout<<"x"<<(i+1)<<" = "<<ans[i]<<"\n";
}
```

OUTPUT:



GAUSS JORDAN METHOD

Solve the following system of equations using Gauss Jordan Method.

$$10x + y + z = 12$$
$$x + 10y + z = 12$$
$$x + y + 10z = 12$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

const int n = 3;
double a[][n+2] = {{10,1,1,12},{1,10,1,12},{1,1,10,12}};
double ans[n+1];
int main() {
	for (int i=0;i<n;i++) {
		double temp = a[i][i];
		for (int j=0;j<=n;j++) {
			a[i][j] /= temp;
		}
		for (int j=0;j<n;j++) {
			if (j!=i) {
				temp = a[j][i];
				for (int k=0;k<=n;k++) {
					a[j][k] = a[j][k] - temp*a[i][k];
				}
			}
		}
	}
	cout<<"Auxillary Matrix : \n";
	for (int i=0; i<n; i++) {
		for (int j=0; j<=n; j++) {
			cout<<a[i][j]<<" ";
		}
		cout<<"\n";
		ans[i] = a[i][n];
	}
	cout<<"Solution : \n";
	for (int i=0; i<n; i++) {
		cout<<(char)('x'+i)<<" = "<<ans[i]<<"\n";
	}
}
```

OUTPUT:`

```
"C:\Users\kansa\Documents\Numerical Methods\Gauss Jordan.exe"
Auxillary Matrix :
1 0 0 1
0 1 0 1
0 0 1 1
Solution :
x = 1
y = 1
z = 1
```

$$
\begin{gathered}
2x_1 + x_2 + 5x_3 + x_4 = 5 \\
x_1 + x_2 - 3x_3 + 4x_4 = -1 \\
3x_1 + 6x_2 - 2x_3 + x_4 = 8 \\
2x_1 + 2x_2 + 2x_3 - 3x_4 = 2
\end{gathered}
$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;

const int n = 4;
double a[][n+2] = {{2,1,5,1,5},{1,1,-3,4,-1},{3,6,-2,1,8},{2,2,2,-3,2}};
double ans[n+1];
int main() {
	for (int i=0;i<n;i++) {
		double temp = a[i][i];
		for (int j=0;j<=n;j++) {
			a[i][j] /= temp;
		}
		for (int j=0;j<n;j++) {
			if (j!=i) {
				temp = a[j][i];
				for (int k=0;k<=n;k++) {
					a[j][k] = a[j][k] - temp*a[i][k];
				}
			}
		}
	}
```

```
        cout<<"Auxillary Matrix : \n";
        for (int i=0; i<n; i++) {
                for (int j=0; j<=n; j++) {
                        cout<<a[i][j]<<" ";
                }
                cout<<"\n";
                ans[i] = a[i][n];
        }
        cout<<"Solution : \n";
        for (int i=0; i<n; i++) {
                cout<<"x"<<(i+1)<<" = "<<ans[i]<<"\n";
        }
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Gauss Jordan.exe"
Auxillary Matrix :
1 0 0 0 -2.17391
0 1 0 0 2.73913
0 0 1 0 1.21739
-0 -0 -0 1 0.521739
Solution :
x1 = -2.17391
x2 = 2.73913
x3 = 1.21739
x4 = 0.521739

Process returned 0 (0x0)   execution time : 0.031 s
Press any key to continue.
```

GAUSS-JACOBI METHOD

Solve the following system of equations using Gauss-Jacobi Method.

$$4x - y + z = 7$$
$$4x - 8y + z = -21$$
$$-2x + y + 5z = 15$$

CODE:

```
#include<bits/stdc++.h>
using namespace std;
double next_x(double y, double z) {
   return (7.0 + y - z)/4.0;
}
double next_y(double x, double z) {
   return (21.0 + 4.0*x + z)/8.0;
}
double next_z(double x, double y) {
   return (15.0 + 2.0*x - y)/5.0;
}
void jacobi(int n, double x0, double y0, double z0) {
   double x, y, z;
   for (int i=1; i<=n; i++) {
      x = next_x(y0, z0);
      y = next_y(x0, z0);
      z = next_z(x0, y0);
      x0 = x, y0 = y, z0 = z;
   }
   cout<<"Solution: "<<"x = "<<x0<<", y = "<<y0<<", z = "<<z0<<endl;
}
int main() {
         int n; double x0, y0, z0;
         cout<<"Enter the no. of iterations: ";
         cin>>n;
         cout<<"Enter starting values: ";
         cin>>x0>>y0>>z0;
         jacobi(n, x0, y0, z0);
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Jacobi Method.exe"
Enter the no. of iterations: 10000
Enter starting values: 1 2 2
Solution: x = 2, y = 4, z = 3
```

TRAPEZOIDAL RULE (NUMERICAL INTEGRATION)

Calculate the value of $\int_0^{\pi/2} \sin(x)\,.\,dx$ by Trapezoidal rule.

CODE:

```
#include<bits/stdc++.h>
using namespace std;
long double pi = 3.141592653589793238462643383279502884197169399375105820974944592307816406286;

long double f(long double x) {
   return (long double)sin(x);
}
long double solve(int n) {
   long double h = pi/(2.0*n);
   long double x = 0.0;

   long double ans = 0.0;
   for (int i=0;i<n;i++) {
      long double f1 = f(x);
      long double f2 = f(x+h);

      ans += (h*(f1+f2))/2.0;
      x += h;
   }
   return ans;
}
int main() {
   int n;
   cout<<"Enter the no. of divisions: ";
   cin>>n;
   cout<<fixed<<setprecision(20)<<solve(n)<<endl;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Trapezoidal Integration.exe"
Enter the no. of divisions: 100000
0.99999999997943829814
```

SIMPSON’S 3/8th RULE

Calculate the value of $\int_0^{\pi/2} \sqrt{\cos(\theta)}\,.\,d\theta$ by Trapezoidal rule.

CODE:

```
#include<bits/stdc++.h>
using namespace std;
long double pi = 3.141592653589793238462643383279502884197169399375105820974944592307816406286;

long double f(long double x) {
   return (long double)sqrt(cos(x));
}
long double solve(int n) {
   long double h = pi/(2.0*n);
   long double x = 0.0;

   long double ans = 0.0;
   for (int i=3; i<=n; i+=3) {
      long double f1 = f(x);
      long double f2 = f(x+h);
      long double f3 = f(x+2.0*h);
      long double f4 = f(x+3.0*h);

      ans += (3.0*h*(f1+3.0*f2+3.0*f3+f4))/8.0;
      x += 3.0*h;
   }
   return ans;
}
int main() {
   int n;
   cout<<"Enter the no. of divisions (Multiple of 3): ";
   cin>>n;
   cout<<fixed<<setprecision(20)<<solve(n)<<endl;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Simpsons 3-8.exe"
Enter the no. of divisions (Multiple of 3): 1000008
1.19814023454103823845
```

SIMPSON'S 1/3rd RULE

Calculate the value of $\int_0^5 \frac{dx}{4x+5}$ by Trapezoidal rule.

CODE:

```
#include<bits/stdc++.h>
using namespace std;

long double f(long double x) {
   return (long double)1.0/(4.0*x + 5.0);
}
long double solve(int n) {
   long double h = (long double)5.0/n;
   long double x = 0.0;

   long double ans = 0.0;
   for (int i=2;i<=n;i+=2) {
      long double f1 = f(x);
      long double f2 = f(x+h);
      long double f3 = f(x+2.0*h);

      ans += (h*(f1+4.0*f2+f3))/3.0;
      x += 2.0*h;
   }
   return ans;
}
int main() {
   int n;
   cout<<"Enter the no. of divisions (Even): ";
   cin>>n;
   cout<<fixed<<setprecision(20)<<solve(n)<<endl;
}
```

OUTPUT:

```
"C:\Users\kansa\Documents\Numerical Methods\Simpsons 1-3.exe"
Enter the no. of divisions (Even): 100000
0.40235947810852506308
```