IMPLEMENTATION OF QUADRATIC EQUATION

```
#include <math.h>

#include <stdio.h>

#include <stdlib.h>

void findRoots(int a, int b, int c)

{

   if (a == 0)

   {

      printf("Invalid");

      return;

   }

   int d = b * b - 4 * a * c;

   double sqrt_val = sqrt(abs(d));

   if (d > 0)

   {

      printf("Roots are real and different \n");

      printf("%f\n%f", (double)(-b + sqrt_val) / (2 * a), (double)(-b - sqrt_val) / (2 * a));

   }

   else if (d == 0)

   {

      printf("Roots are real and same \n");

      printf("%f", -(double)b / (2 * a));

   }

   else

   {

      printf("Roots are complex \n");

      printf("%f + i%f\n%f - i%f", -(double)b / (2 * a), sqrt_val / (2 * a), -(double)b / (2 * a), sqrt_val / (2
* a));

   }

}

int main()
```

```
{
    int a, b, c;
    printf("\n-------------------Shubhangi Joshi-----------------------\n\n");
    printf("Enter the values of a,b andc: ");
    scanf("%d%d%d", &a, &b, &c);
    findRoots(a, b, c);
    return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi-----------------------

Enter the values of a,b andc: 1
2
5
Roots are complex
-1.000000 + i2.000000
-1.000000 - i2.000000
```

Implementation of Regular- Falsi Method.

```
#include<stdio.h>
#include<math.h>
double func(double x){
return x*x*x-2*x-5;
}
void regular_falsi(double a,double b,double error)
{
if(func(a)*func(b)>0)
printf("Invalid guesses");
else{
double c;
int i=1;
while((b-a)>error)
{
c=(a*func(b)-b*func(a))/(func(b)-func(a));
if(i<15)
{
printf("Iteration %d: c=%lf f(c)=%lf\n", i, c,func(c));
i++;
}
else{
break;
}
if(func(a)*func(c)<0)
b=c;
if(func(b)*func(c)<0)
a=c;
}
printf("Approximate root of the equation is: %lf\n", c);
```

```
}
}
int main()
{
double a,b,error;
printf("\n------------------Shubhangi Joshi----------------------\n\n");
printf("the equation is : x^3 - 2x - 5 = 0\n");
printf("enter the guesses\n");
printf("enter the value of a=");
scanf("%lf",&a);
printf("enter the value of b=");
scanf("%lf",&b);
printf("enter the value of error=");
scanf("%lf",&error);
regular_falsi(a,b,error);
return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi----------------------

the equation is : x^3 - 2x - 5 = 0
enter the guesses
enter the value of a=2
enter the value of b=3
enter the value of error=0.001
Iteration 1: c=2.058824 f(c)=-0.390800
Iteration 2: c=2.081264 f(c)=-0.147204
Iteration 3: c=2.089639 f(c)=-0.054677
Iteration 4: c=2.092740 f(c)=-0.020203
Iteration 5: c=2.093884 f(c)=-0.007451
Iteration 6: c=2.094305 f(c)=-0.002746
Iteration 7: c=2.094461 f(c)=-0.001012
Iteration 8: c=2.094518 f(c)=-0.000373
Iteration 9: c=2.094539 f(c)=-0.000137
Iteration 10: c=2.094547 f(c)=-0.000051
Iteration 11: c=2.094550 f(c)=-0.000019
Iteration 12: c=2.094551 f(c)=-0.000007
Iteration 13: c=2.094551 f(c)=-0.000003
Iteration 14: c=2.094551 f(c)=-0.000001
Approximate root of the equation is: 2.094551
```

Implementation of Bisection Method

```
#include<stdio.h>
#include<math.h>
double equation(double x)
{
return x-cos(x);
}
void bisection(double a, double b, double e)
{
double x2;
int i = 1;
if(equation(a)*equation(b) >= 0.0)
{
printf("Invalid guess, please try again!");
}
else
{
while( (b-a) > e)
{
x2 = (b+a)/2.0;
printf("Iteration %d : x2 = %lf\n", i, x2);
i++;
if(equation(x2)== 0.0)
break;
else if(equation(a)*equation(x2) < 0)
b=x2;
else
a=x2;
}
printf("\n The root of the equation is %lf ", x2);
```

```
}
}
int main()
{
double a, b , e;
printf("\n------------------Shubhangi Joshi----------------------\n\n");
printf("The given equation is : x-cos(x) \n");
printf("Enter the values of a and b: ");
scanf("%lf %lf", &a,&b);
printf("Enter the precision value:");
scanf("%lf", &e);
bisection(a,b,e);
return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi-----------------------

The given equation is : x-cos(x)
Enter the values of a and b: 0.7
0.8
Enter the precision value:0.001
Iteration 1 : x2 = 0.750000
Iteration 2 : x2 = 0.725000
Iteration 3 : x2 = 0.737500
Iteration 4 : x2 = 0.743750
Iteration 5 : x2 = 0.740625
Iteration 6 : x2 = 0.739063
Iteration 7 : x2 = 0.739844

 The root of the equation is 0.739844
```

Implementation of Newton – Rapshon Method

```
#include <stdio.h>
#include <math.h>
float f(float x)
{
return x * x -(2* x)- 3 * cos(x);
}
float df(float x)
{
return 2 * x - 2 + 3 * sin(x);
}
void Newton_Rapson(float a,float b,float e){
float x0,x1;
int iter = 0, max_iter = 10;
x0 = (f(a)<f(b))?a:b;
printf("x0 value:%f\n", x0);
do {
x1 = x0 - f(x0) / df(x0);
iter++;
printf("Iteration %d: x%d = %.5f\n", iter, iter, x1);
if ((f(x1)) < e)
{
printf("Approximate root found: %.5f\n", x1);
break;
}
x0 = x1;
}
while (iter < max_iter);
if (iter >= max_iter)
{
```

```
printf("no approximate root found.\n");
}
}
int main() {
float a, b, e;
printf("\n-------------------Shubhangi Joshi-----------------------\n\n");
printf("Enter a: ");
scanf("%f", &a);
printf("Enter b: ");
scanf("%f", &b);
printf("Enter error limit: ");
scanf("%f", &e);
Newton_Rapson(a,b,e);
return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi-----------------------

Enter a: 1.6
Enter b: 1.7
Enter error limit: 0.001
x0 value:1.600000
Iteration 1: x1 = 1.73156
Iteration 2: x2 = 1.72808
Approximate root found: 1.72808
```

IMPLEMENTATION OF SECANT METHOD

```
#include <stdio.h>
#include <math.h>

double f(double x)
{
    return cos(x) + 2 * sin(x) + x * x;
}

void secant_method(double x0, double x1, double e, int n)
{
    if (f(x0) == f(x1))
        printf("Mathematical error !");
    double x2;
    int i = 1;
    do
    {
        x2 = x1 - ((x1 - x0) / (f(x1) - f(x0))) * f(x1);
        x2 = floor(x2 * 10000) / 10000;
        printf("Iteration %d is %lf\n", i, x2);
        x0 = floor(x1 * 10000) / 10000;
        x1 = floor(x2 * 10000) / 10000;
        i++;
        n--;

    } while (f(x2) > e && n > 0);
    printf("\nThe approximate root is %.4f", x2);
}
```

```
int main()
{
    double x0, x1, e;
    int n;
    printf("\n-------------------Shubhangi Joshi-----------------------\n\n");
    printf("\nThe given function f(x) is : cos(x)+2*sin(x)+x*x");
    printf("\nEnter the interval x0 and x1 :");
    scanf("%lf %lf", &x0, &x1);
    printf("\nEnter the tolerable error: ");
    scanf("%lf", &e);
    printf("\n Enter the no. of iterations: ");
    scanf("%d", &n);
    secant_method(x0, x1, e, n);
    return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi-----------------------



The given function f(x) is : cos(x)+2*sin(x)+x*x
Enter the interval x0 and x1 :0
-0.1
Enter the tolerable error: 0.0001
Enter the no. of iterations: 10
Iteration 1 is -0.513800
Iteration 2 is -0.610000
Iteration 3 is -0.651900
Iteration 4 is -0.658900
Iteration 5 is -0.659300

The approximate root is -0.6593
```

IMPLEMENTATION OF GAUSS ELIMINATION METHOD

```
#include <stdio.h>

int main() {
    int n;
    int m;
    printf("\n-------------------Shubhangi Joshi----------------------\n\n");
    printf("Input size of matrix:");
    scanf("%d %d", &n, &m);

    double mat[n][m + 1];

    // Input the augmented matrix
    printf("Enter the augmented matrix coefficients:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= m; j++) {
            printf("Enter a[%d][%d]: ", i + 1, j + 1);
            scanf("%lf", &mat[i][j]);
        }
    }

    // Perform Gaussian elimination
    printf("Gaussian Elimination Steps:\n");
    for (int k = 0; k < n - 1; k++) {
        for (int i = k + 1; i < m; i++) {
            double factor = mat[i][k] / mat[k][k];
            for (int j = k; j <= n; j++) {
                mat[i][j] -= factor * mat[k][j];
            }
        }

        // Print the current upper triangular matrix
        printf("Step %d:\n", k + 1);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j <= m; j++) {
                printf("%.2lf\t", mat[i][j]);
            }
            printf("\n");
        }
    }

    // Back-substitution to find the solution
    double x[n];
    x[n - 1] = mat[n - 1][m] / mat[n - 1][m - 1];
    for (int i = n - 2; i >= 0; i--) {
        double sum = 0;
```

```
        for (int j = i + 1; j < n; j++) {
            sum += mat[i][j] * x[j];
        }
        x[i] = (mat[i][n] - sum) / mat[i][i];
    }

    // Output the solution
    printf("\nSolution:\n");
    for (int i = 0; i < n; i++) {
        printf("x[%d] = %.2lf\n", i + 1, x[i]);
    }

    return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi-----------------------

Input size of matrix:3
3
Enter the augmented matrix coefficients:
Enter a[1][1]: 1
Enter a[1][2]: 1
Enter a[1][3]: 1
Enter a[1][4]: 2
Enter a[2][1]: 5
Enter a[2][2]: 3
Enter a[2][3]: 9
Enter a[2][4]: 2
Enter a[3][1]: 2
Enter a[3][2]: 0
Enter a[3][3]: 4
Enter a[3][4]: 1
Gaussian Elimination Steps:
Step 1:
1.00    1.00    1.00    2.00
0.00    -2.00   4.00    -8.00
0.00    -2.00   2.00    -3.00
Step 2:
1.00    1.00    1.00    2.00
0.00    -2.00   4.00    -8.00
0.00    0.00    -2.00   5.00

Solution:
x[1] = 5.50
x[2] = -1.00
x[3] = -2.50
```

IMPLEMENTATION OF GAUSS JORDAN METHOD

```
#include <stdio.h>

int main() {
    int n;
    int m;
    printf("\n-------------------Shubhangi Joshi----------------------\n\n");
    printf("Input size of matrix:");
    scanf("%d %d", &n, &m);

    double mat[n][m + 1];

    // Input the augmented matrix
    printf("Enter the augmented matrix coefficients:\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= m; j++) {
            printf("Enter a[%d][%d]: ", i + 1, j + 1);
            scanf("%lf", &mat[i][j]);
        }
    }

    // Gauss-Jordan elimination
    for (int k = 0; k < n; k++) {
        // Make the diagonal element of this row 1
        double pivot = mat[k][k];
        for (int j = k; j <= m; j++) {
            mat[k][j] /= pivot;
        }
```

```
        // Make other rows' elements in this column 0
        for (int i = 0; i < n; i++) {
            if (i != k) {
                double factor = mat[i][k];
                for (int j = k; j <= m; j++) {
                    mat[i][j] -= factor * mat[k][j];
                }
            }
        }

        // Print the augmented matrix at this step
        printf("Step %d:\n", k + 1);
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                printf("%.2lf ", mat[i][j]);
            }
            printf("| %.2lf\n", mat[i][m]);
        }
    }

    // Output the solution
    printf("\n Solution:\n");
    for (int i = 0; i < n; i++) {
        printf("x[%d] = %.2lf\n", i + 1, mat[i][m]);
    }

    return 0;
}
```

OUTPUT

```
-------------------Shubhangi Joshi----------------------

Input size of matrix:3
3
Enter the augmented matrix coefficients:
Enter a[1][1]: 1
Enter a[1][2]: 1
Enter a[1][3]: 1
Enter a[1][4]: 9
Enter a[2][1]: 2
Enter a[2][2]: -3
Enter a[2][3]: 4
Enter a[2][4]: 13
Enter a[3][1]: 3
Enter a[3][2]: 4
5Enter a[3][3]: 5
Enter a[3][4]: 40
Step 1:
1.00 1.00 1.00 | 9.00
0.00 -5.00 2.00 | -5.00
0.00 1.00 2.00 | 13.00
Step 2:
1.00 0.00 1.40 | 8.00
0.00 1.00 -0.40 | 1.00
0.00 0.00 2.40 | 12.00
Step 3:
1.00 0.00 0.00 | 1.00
0.00 1.00 0.00 | 3.00
0.00 0.00 1.00 | 5.00

Solution:
x[1] = 1.00
x[2] = 3.00
x[3] = 5.00
```