1-Roots of Quadratic equation

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
//calculating the roots of equation
void calc_roots(double a, double b, double c)
{
   if (a == 0)
{
      printf("Invalid Equation");
      return;
   }
   int d = b * b - 4 * a * c;
   double sqrt_val = sqrt(abs(d));
   if (d > 0)
   {
      printf("Roots are both real and different \n");
      printf("%f \n %f", (-b + sqrt_val) / (2 * a), (-b - sqrt_val) / (2 * a));
   }
   else if (d == 0)
   {
      printf("Roots are real and same \n");
      printf("%f", -b / (2 * a));
   }
   else
   {
      printf("Roots are complex \n");
      printf("%f + i%f\n%f - i%f", -b / (2 * a), sqrt_val, -b / (2 * a), sqrt_val);
   }
}
int main()
{
   double a = 4, b = 4, c = 1;
   calc_roots(a, b, c);
   return 0;
}
```

2-Finding root using Newton Raphson method

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

/*funtion returns a float*/

float fn(float x)
{
    return ((x * x * x) - (8 * x) - 4);
}

float find(float x)
{
    return ((3 * x * x) - 8);
}

int main()
{
    float x = 1;

    for (int i = 0; i < 10; i++)
    {

        x = x - (fn(x) / find(x));

        printf("Iter %d =%f\n", i + 1, x);
    }
    return 0;
}
```

3-Finding root using Bisection

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

#define F(x) (x * x * x - 4 * x + 2)

int main()
{
    int i = 1;
```

```
    float a, b, c, f;
    printf("\n Enter the value of a and b : ");
    scanf("%f%f", &a, &b);

    do
    {
        c = (a + b) / 2;
        f = F(c);
        printf("\n i=%d a=%f b=%f c=%f F(c)=%f", i, a, b, c, f);
        if (F(a) * F(c) < 0)
        {
            b = c;
        }
        else
        {
            a = c;
        }
        i++;
    } while (fabs(F(c)) > 0.001);

    printf("\n\n\n approximate root=%.3f \n\n", c);
    return 0;
}
```

4-Forward (Newton’s) Difference Table

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

int main()
{
    float x[20], y[20][20];
    int i, j, n;

    /* Input Section */
    printf("Enter number of data?\n");
    scanf("%d", &n);
    printf("Enter data:\n");
    for (i = 0; i < n; i++)
    {
```

```
        printf("x[%d]=", i);
        scanf("%f", &x[i]);
        printf("y[%d]=", i);
        scanf("%f", &y[i][0]);
    }

    /* Generating Forward Difference Table */
    for (i = 1; i < n; i++)
    {
        for (j = 0; j < n - i; j++)
        {
            y[j][i] = y[j + 1][i - 1] - y[j][i - 1];
        }
    }

    /* Displaying Forward Difference Table */
    printf("\nFORWARD DIFFERENCE TABLE\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%0.2f", x[i]);
        for (j = 0; j < n - i; j++)
        {
            printf("\t%0.2f", y[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

5-Backward (Newton’s) Difference Table

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

int main()
{
    float x[20], y[20][20];
    int i, j, n;
```

```
    /* Input Section */
    printf("Enter number of data?\n");
    scanf("%d", &n);
    printf("Enter data:\n");
    for (i = 0; i < n; i++)
    {
        printf("x[%d]=", i);
        scanf("%f", &x[i]);
        printf("y[%d]=", i);
        scanf("%f", &y[i][0]);
    }

    /* Generating Backward Difference Table */
    for (i = 1; i < n; i++)
    {
        for (j = n - 1; j > i - 1; j--)
        {
            y[j][i] = y[j][i - 1] - y[j - 1][i - 1];
        }
    }

    /* Displaying Backward Difference Table */
    printf("\nBACKWARD DIFFERENCE TABLE\n\n");
    for (i = 0; i < n; i++)
    {
        printf("%0.2f", x[i]);
        for (j = 0; j <= i; j++)
        {
            printf("\t%0.2f", y[i][j]);
        }
        printf("\n");
    }

    getch(); /* Holding Screen */
    return 0;
}
```

6-Lagrange Interpolation

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

```
void main()
{
    float x[100], y[100], xp, yp = 0, p;
    int i, j, n;

    /* Input Section */
    printf("Enter number of data: ");
    scanf("%d", &n);
    printf("Enter data:\n");
    for (i = 1; i <= n; i++)
    {
        printf("x[%d] = ", i);
        scanf("%f", &x[i]);
        printf("y[%d] = ", i);
        scanf("%f", &y[i]);
    }
    printf("Enter interpolation point: ");
    scanf("%f", &xp);
    /* Implementing Lagrange Interpolation */
    for (i = 1; i <= n; i++)
    {
        p = 1;
        for (j = 1; j <= n; j++)
        {
            if (i != j)
            {
                p = p * (xp - x[j]) / (x[i] - x[j]);
            }
        }
        yp = yp + p * y[i];
    }
    printf("Interpolated value at %.3f is %.3f.", xp, yp);
    getch();
}
```

7-Trapezoidal Method

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

```
/* Define function here */
#define f(x) 4 * x - (3 * x * x)

int main()
{
    float lower, upper, integration = 0.0, stepSize, k;
    int i, subInterval;

    /* Input */
    printf("Enter lower limit of integration: ");
    scanf("%f", &lower);
    printf("Enter upper limit of integration: ");
    scanf("%f", &upper);
    printf("Enter number of sub intervals: ");
    scanf("%d", &subInterval);

    /* Calculation */
    /* Finding step size */
    stepSize = (upper - lower) / subInterval;

    /* Finding Integration Value */
    integration = f(lower) + f(upper);
    for (i = 1; i <= subInterval - 1; i++)
    {
        k = lower + i * stepSize;
        integration = integration + 2 * f(k);
    }
    integration = integration * stepSize / 2;
    printf("\nRequired value of integration is: %.3f", integration);

    /* declare abs value */
    float abs_val = 1.00;

    /*  Absolute errors */
    float abs_err = abs_val - integration;
    printf("\nAbsolute errors is: %.3f", abs_err);

    /*  Relative errors */
    float rel_err = (abs_val - integration) / abs_val;
```

```
    printf("\nRelative errors is: %.3f", rel_err);

    return 0;
}
```

8-Simpson one third Rule

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

/* Define function here */
#define f(x) 4 * x - (3 * x * x)

int main()
{
    float lower, upper, integration, stepSize, k;
    int i, subInterval;

    /* Input */
    printf("Enter lower limit of integration: ");
    scanf("%f", &lower);
    printf("Enter upper limit of integration: ");
    scanf("%f", &upper);
    printf("Enter number of sub intervals: ");
    scanf("%d", &subInterval);

    /* Calculation */
    /* Finding step size */
    stepSize = (upper - lower) / subInterval;

    /* Finding Integration Value */
    integration = f(lower) + f(upper);
    for (i = 1; i < subInterval; i++)
    {
        k = lower + i * stepSize;
        if (i % 2 == 0)
        {
            integration = integration + 2 * f(k);
        }
        else
        {
```

```
            integration = integration + 4 * f(k);
        }
    }
    integration = integration * stepSize / 3;
    printf("\nRequired value of integration is: %.3f", integration);

    /* declare abs value */
    float abs_val = 1.00;

    /*  Absolute errors */
    float abs_err = abs_val - integration;
    printf("\nAbsolute errors is: %.3f", abs_err);

    /*  Relative errors */
    float rel_err = (abs_val - integration) / abs_val;
    printf("\nRelative errors is: %.3f", rel_err);
    return 0;
}
```

9-Simpson three eight Method

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

/* Define function here */
#define f(x) 1 / (1 + x * x)

int main()
{
    float lower, upper, integration = 0.0, stepSize, k;
    int i, subInterval;

    /* Input */
    printf("Enter lower limit of integration: ");
    scanf("%f", &lower);
    printf("Enter upper limit of integration: ");
    scanf("%f", &upper);
    printf("Enter number of sub intervals: ");
    scanf("%d", &subInterval);
```

```
    /* Calculation */
    /* Finding step size */
    stepSize = (upper - lower) / subInterval;

    /* Finding Integration Value */
    integration = f(lower) + f(upper);
    for (i = 1; i <= subInterval - 1; i++)
    {
        k = lower + i * stepSize;
        if (i % 3 == 0)
        {
            integration = integration + 2 * f(k);
        }
        else
        {
            integration = integration + 3 * f(k);
        }
    }
    integration = integration * stepSize * 3 / 8;
    printf("\nRequired value of integration is: %.3f", integration);
    getch();
    return 0;
}
```

10-Weddle’s rule

```
#include <stdio.h>
float y(float x)
{
    return 1 / (1 + x * x); //function of which integration is to be calculated
}
int main()
{
    float a, b, h, sum;
    int i, n, m;

    printf("Enter a=x0(lower limit), b=xn(upper limit), number of subintervals:
");
    scanf("%f%f%d", &a, &b, &n);
    h = (b - a) / n;
    sum = 0;
```

```
    if (n % 6 == 0)
    {
        sum = sum + ((3 * h / 10) * (y(a) + y(a + 2 * h) + 5 * y(a + h) + 6 * y(a +
3 * h) + y(a + 4 * h) + 5 * y(a + 5 * h) + y(a + 6 * h)));
        a = a + 6 * h;

        printf("Value of integral is %f\n", sum);
    }
    else
    {
        printf("Sorry ! Weddle rule is not applicable");
    }
}
```

11-Euler's method

```
#include <conio.h>
#include <stdio.h>
#define f(x, y) (y - x) / (y + x)
void main()
{
    float x, y, h, xn, l;

    printf("Program for Solution of Ordinary Differential Equation\nEuler's
Method\n");
    printf("Enter value for x and y\n");
    scanf("%f%f", &x, &y);
    printf("Enter value for h and last of x\n");
    scanf("%f%f", &h, &xn);
    while (x + h <= xn)
    {
        l = h * f(x, y);
        y = y + l;
        x = x + h;

        printf("y = %f\tx = %f\n", y, x);
    }

    return 0;
}
```

12-Runge-Kutta 4th order method

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

```
#include <conio.h>

#define f(x, y) (y * y - x * x) / (y * y + x * x)

int main()
{
    float x0, y0, xn, h, yn, k1, k2, k3, k4, k;
    int i, n;

    printf("Enter Initial Condition\n");
    printf("x0 = ");
    scanf("%f", &x0);
    printf("y0 = ");
    scanf("%f", &y0);
    printf("Enter calculation point xn = ");
    scanf("%f", &xn);
    printf("Enter number of steps: ");
    scanf("%d", &n);

    /* Calculating step size (h) */
    h = (xn - x0) / n;

    /* Runge Kutta Method */
    printf("\nx0\ty0\tyn\n");
    for (i = 0; i < n; i++)
    {
        k1 = h * (f(x0, y0));
        k2 = h * (f((x0 + h / 2), (y0 + k1 / 2)));
        k3 = h * (f((x0 + h / 2), (y0 + k2 / 2)));
        k4 = h * (f((x0 + h), (y0 + k3)));
        k = (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        yn = y0 + k;
        printf("%0.4f\t%0.4f\t%0.4f\n", x0, y0, yn);
        x0 = x0 + h;
        y0 = yn;
    }
```

13-Gauss Elimination method

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

```
#include <stdlib.h>

#define SIZE 10

int main()
{
    float a[SIZE][SIZE], x[SIZE], ratio;
    int i, j, k, n;

    /* Inputs */
    /* 1. Reading number of unknowns */
    printf("Enter number of unknowns: ");
    scanf("%d", &n);
    /* 2. Reading Augmented Matrix */
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n + 1; j++)
        {
            printf("a[%d][%d] = ", i, j);
            scanf("%f", &a[i][j]);
        }
    }
    /* Applying Gauss Elimination */
    for (i = 1; i <= n - 1; i++)
    {
        if (a[i][i] == 0.0)
        {
            printf("Mathematical Error!");
            exit(0);
        }
        for (j = i + 1; j <= n; j++)
        {
            ratio = a[j][i] / a[i][i];

            for (k = 1; k <= n + 1; k++)
            {
                a[j][k] = a[j][k] - ratio * a[i][k];
            }
        }
    }
```

14-Gauss-Seidel Method

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

/* Arrange systems of linear
   equations to be solved in
   diagonally dominant form
   and form equation for each
   unknown and define here
*/
/* In this example we are solving
   3x + 20y - z = -18
   2x - 3y + 20z = 25
   20x + y - 2z = 17
*/
/* Arranging given system of linear
   equations in diagonally dominant
   form:
   20x + y - 2z = 17
   3x + 20y -z = -18
   2x - 3y + 20z = 25
*/
/* Equations:
   x = (17-y+2z)/20
   y = (-18-3x+z)/20
   z = (25-2x+3y)/20
*/
/* Defining function */
#define f1(x, y, z) (17 - y + 2 * z) / 20
#define f2(x, y, z) (-18 - 3 * x + z) / 20
#define f3(x, y, z) (25 - 2 * x + 3 * y) / 20
```

15-Jacobi Iteration Method

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

/* Defining function */
#define f1(x, y, z) (17 - y + 2 * z) / 20
#define f2(x, y, z) (-18 - 3 * x + z) / 20
```

```c
#define f3(x, y, z) (25 - 2 * x + 3 * y) / 20

/* Main function */
int main()
{
    float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1, e1, e2, e3, e;
    int count = 1;

    printf("Enter tolerable error:\n");
    scanf("%f", &e);

    printf("\nCount\tx\ty\tz\n");
    do
    {
        /* Calculation */
        x1 = f1(x0, y0, z0);
        y1 = f2(x0, y0, z0);
        z1 = f3(x0, y0, z0);
        printf("%d\t%0.4f\t%0.4f\t%0.4f\n", count, x1, y1, z1);

        /* Error */
        e1 = fabs(x0 - x1);
        e2 = fabs(y0 - y1);
        e3 = fabs(z0 - z1);

        count++;

        /* Set value for next iteration */
        x0 = x1;
        y0 = y1;
        z0 = z1;
    } while (e1 > e && e2 > e && e3 > e);

    printf("\nSolution: x=%0.3f, y=%0.3f and z = %0.3f\n", x1, y1, z1);

    getch();
    return 0;
}
```

16-Regula Falsi method

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

```
#include <conio.h>
#include <math.h>
/* Defining equation to be solved.
   Change this equation to solve another problem. */
#define f(x) x *log10(x) - 1.2

int main()
{

    float x0, x1, x2, f0, f1, f2, e;
    int step = 1;

/* Inputs */
up:
    printf("\nEnter two initial guesses:\n");
    scanf("%f%f", &x0, &x1);
    printf("Enter tolerable error:\n");
    scanf("%f", &e);
    /* Calculating Functional Values */
    f0 = f(x0);
    f1 = f(x1);
    /* Checking whether given guesses brackets the root or not. */
    if (f0 * f1 > 0.0)
    {
        printf("Incorrect Initial Guesses.\n");
        goto up;
    }
    /* Implementing Regula Falsi or False Position Method */
    printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");
    do
    {
        x2 = x0 - (x0 - x1) * f0 / (f0 - f1);
        f2 = f(x2);
        printf("%d\t\t%f\t%f\t%f\t%f\n", step, x0, x1, x2, f2);

        if (f0 * f2 < 0)
        {
            x1 = x2;
            f1 = f2;
        }
```

```
        else
        {
            x0 = x2;
            f0 = f2;
        }
        step = step + 1;

    } while (fabs(f2) > e);

    printf("\nRoot is: %f", x2);
    getch();
    return 0;
}
```