1. Bisection Method

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

float f(float x) {
  float y;
  y = pow(x, 2) + x - 2;
  return y;
}

int main() {
  float x1, x2, x0, error = 0.0001;
  int i = 0;

  printf("\nEnter two initial guess:\n");
  scanf("%f%f", &x1, &x2);

  if (f(x1) * f(x2) > 0) {
    printf("\nWrong Input !!!");
    exit(0);
  }
  else {
    do{
      x0 =(x1 + x2)/2;
      if (f(x0) * f(x1) > 0) {
        x1 = x0;
      }
      else{
        x2 = x0;
      }
      i++;
    }while (fabs(f(x0)) > error);
```

```
}
  printf("\nRoot = %f", x0);
  printf("\nNumber of iteration = %d", i);
  printf("\n\n\t\t  Abhinaya Aryal\n\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc bisection_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter two initial guess:
0
1

Root = 0.999969
Number of iteration = 15

                  Abhinaya Aryal


[abhinaya@arch nmPractical]$
```

2. Secant Method

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

float f(float x) {
  float y;

  y = pow(x, 2) + x - 2;
  return y;
}

int main() {
  float x1, x2, x0, error = 0.0001;
  int i = 0;

  printf("\nEnter any two initial guess:\n");
  scanf("%f%f", &x1, &x2);

  do {
    x0 = x1 - (f(x1) * (x2 - x1)) / (f(x2) - f(x1));
    x2 = x1;
    x1 = x0;
    i++;
  }while (fabs(f(x0)) > error);

  printf("\nRoot = %f", x0);
  printf("\nNumber of iteration = %d", i);
  printf("\n\n\t\t  Abhinaya Aryal\n\n\n");

  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc secant_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter any two initial guess:
0
1

Root = 1.000000
Number of iteration = 1

                  Abhinaya Aryal


[abhinaya@arch nmPractical]$
```

3. Newton Raphson Method

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

float f(float x) {
  float y;

  y = pow(x, 2) + x - 2;
  return y;
}

float fd(float x) {
  float y;

  y = 2 * x + 1;
  return y;
}

int main() {
  float x0, x1, error = 0.0001;
  int i = 0;

  printf("\nGuess Initial Root:\n");
  scanf("%f", &x1);

  do {
    x0 = x1 - (f(x1) / fd(x1));
    x1 = x0;
    i++;
  }while (fabs(f(x0)) > error);

  printf("\nRoot = %f", x0);
  printf("\nNumber of iteration = %d", i);
```

```
  printf("\n\n\t\t  Abhinaya Aryal\n\n\n");

  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc newton_raphson_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Guess Initial Root:
0

Root = 1.000000
Number of iteration = 5

                Abhinaya Aryal


[abhinaya@arch nmPractical]$
```

4. Fixed Point Method

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

float g(float x) {
  float y;

  y = 2.0 - x * x;
  return y;
}

int main() {
  float x0, x, error, E = 0.0001;

  printf("\nInput initial guess:\n");
  scanf("%f", &x0);

  while (1) {
    x = g(x0);
    error = (x - x0) / x;

    if (fabs(error) < E) {
      printf("\nRoot =  %f", x0);
      printf("\n\n\t\t  Abhinaya Aryal\n\n\n");
      break;
    }

    x0 = x;
  }

  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc fixed_point_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Input initial guess:
1

Root =  1.000000

                  Abhinaya Aryal


[abhinaya@arch nmPractical]$
```

5. Lagrange Interpolation

```
#include <stdio.h>

int main() {
  float x[10], f[10], y, l, sum = 0.0;
  int n, i, j;

  printf("\nInput number of data : \n");
  scanf("%d", &n);
  printf("\nInput data points x(i), & f(i) : \n");

  for (i = 0; i < n; i++) {
    printf("\nx[%d] = ", i);
    scanf("%f", &x[i]);
    printf("\nf[%d] = ", i);
    scanf("%f", &f[i]);
  }

  printf("\n Functional Value: ");
  scanf("%f", &y);

  for (i = 0; i < n; i++) {
    l = 1;

    for (j = 0; j < n; j++) {
      if (j =/= i) {
        l = l * (y - x[j]) / (x[i] - x[j]);
      }
    }

    sum = sum + l * f[i];
  }
```

```
  printf("\n Value at %f = %f\n", y, sum);
  printf("\n\n\t\t  Abhinaya Aryal\n\n\n");

  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc lagrange_interpolation.c
[abhinaya@arch nmPractical]$ ./a.out

Input number of data :
3

Input data points x(i), & f(i) :

x[0] = 2

f[0] = 13

x[1] = 24

f[1] = 35

x[2] = 46

f[2] = 789

 Functional Value: 10

 Value at 10.000000 = -63.694214


                  Abhinaya Aryal
```

6. Curve Fitting Linear Equation

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

#define error 0.001

int main() {
  int i, n;
  float x[10], y[10], sumx = 0.0, sumy = 0.0;
  float sumxx = 0.0, sumxy = 0.0;
  float meanx, meany, denom, a, b;
  printf("\nHow many element? : ");
  scanf("%d", &n);

  for (i = 0; i < n; i++) {
    printf("\nx[%d] = ", i);
    scanf("%f", &x[i]);
    printf("y[%d] = ", i);
    scanf("%f", &y[i]);
  }

  for (i = 0; i < n; i++) {
    sumx += x[i];
    sumy += y[i];
    sumxx += x[i] * x[i];
    sumxy += x[i] * y[i];
  }

  meanx = sumx / n;
  meany = sumy / n;
  denom = n * sumxx - sumx * sumx;
  if (fabs(denom) > error) {
```

```
    b = (n * sumxy - sumx * sumy) / denom;
    a = meany - b * meanx;
    printf("\ny = %fx + %f", b, a);
    printf("\n\n\t\t  Abhinaya Aryal\n\n\n");

  }
  else {
    printf("\nNo solution");
    printf("\n\n\t\t  Abhinaya Aryal\n\n\n");

  }
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc curve_fitting.c -lm
[abhinaya@arch nmPractical]$ ./a.out

How many element? : 3

x[0] = 2
y[0] = 12

x[1] = 4
y[1] = 22

x[2] = 7
y[2] = 34

y = 4.368421x + 3.736841

                  Abhinaya Aryal


[abhinaya@arch nmPractical]$
```

7. Trapezoidal Rule

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

float f(float x) {
  return (1 - exp(-x/2.0));
}

int main() {
  float a, b, h, x, sum = 0;
  int n;

  printf("\nEnter initial and final value of x : \n");
  scanf("%f%f", &a, &b);
  printf("\nNumber of segments : ");
  scanf("%d", &n);
  h = (b - a) / n;

  for (x = a; x ⩽ b; x = x + h) {
    if (x == a) {
      sum = sum + f(x);
    }
    else if (x == b) {
      sum = sum + f(x);
    }
    else {
      sum = sum + 2 * f(x);
    }
  }
  sum = sum * h / 2;

  printf("\nIntegral value of f(x) = %f", sum);
```

```
  printf("\n\n\t\t  Abhinaya Aryal\n\n");

  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc trapezoidal_rule.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter initial and final value of x :
1
2

Number of segments : 6

Integral value of f(x) = 0.575098

                  Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

8. Simpson's 1/3 Rule

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

float f(float x) {
  return (1 - exp(-x/2.0));
}

int main() {
  float a, b, h, x, ans, sum = 0;
  int n, i;

  printf("\nEnter initial and final value of x:\n");
  scanf("%f%f", &a, &b);
  printf("\nNumber of segments : ");
  scanf("%d", &n);
  h = (b - a) / n;

  for (i = 1; i < n; i++) {
    x = a + i * h;
    if (i % 2 == 0) {
      sum = sum + 2 * f(x);
    }
    else {
      sum = sum + 4 * f(x);
    }
  }
  ans = (h / 3) * (f(a) + f(b) + sum);
  printf("\nIntegral value of f(x) = %f", ans);
  printf("\n\n\t\t Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc simpsons_1_by_three.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter initial and final value of x:
1
2

Number of segments : 6

Integral value of f(x) = 0.522697

                Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

9. Simpson's 3/8 Rule

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

float f(float x) {
  return (1 - exp(-x/2.0));
}

int main() {
  float a, b, h, x, ans, sum = 0;
  int n, i;

  printf("\nEnter intial and final value of x : \n");
  scanf("%f%f", &a, &b);
  printf("\nNumber of segment : ");
  scanf("%d", &n);
  h = (b - a) / n;

  for (i = 1; i < n; i++) {
    x = a + i * h;
    if (i % 3 == 0) {
      sum = sum + 2 * f(x);
    }
    else {
      sum = sum + 3 * f(x);
    }
  }

  ans = (3 * h / 8) * (f(a) + f(b) + sum);
  printf("\nIntegral value of f(x) = %f", ans);
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc simpsons_3_by_8.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter intial and final value of x :
1
2

Number of segment : 6

Integral value of f(x) = 0.522697

                  Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

10. Euler Method

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

float fun(float x, float y) {
  float f;

  f = x * y;
  return f;
}

int main() {
  int i, n;
  float x0, y0, xp, h, y;
  printf("\nEnter initial value of x and y : ");
  scanf("%f%f", &x0, &y0);
  printf("\nEnter x at which y is required : ");
  scanf("%f", &xp);
  printf("\nEnter step-size, h : ");
  scanf("%f", &h);

  n = (xp - x0) / h;

  for (i = 0; i < n; i++) {
    y = y0 + h * fun(x0, y0);
    x0 = x0 + h;
    y0 = y;
    printf("%f\t%f\n", x0, y);
  }

  printf("\n Value of y at x = %f is %f", x0, y0);
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
```

```
}
```

```
[abhinaya@arch nmPractical]$ gcc euler_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter initial value of x and y : 2
4

Enter x at which y is required : 4

Enter step-size, h : 0.5
2.500000        8.000000
3.000000        18.000000
3.500000        45.000000
4.000000        123.750000

 Value of y at x = 4.000000 is 123.750000

                 Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

11. Heun's Method

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

float func(float x, float y) {
  float f;
  f = 2.0 * y / x;
  return f;
}

int main() {
  int i, n;
  float x0, y0, xp, h, m1, m2;
  printf("Enter initial value of x and y : ");
  scanf("%f%f", &x0, &y0);
  printf("\n Enter x at which y is required : ");
  scanf("%f", &xp);
  printf("\nEnter step size, h : ");
  scanf("%f", &h);
  n = (xp - x0) / h;

  for (i = 1; i ⩽ n; i++) {
    m1 = func(x0, y0);
    m2 = func(x0 + h, y0 + m1 * h);
    x0 = x0 + h;
    y0 = y0 + 0.5 * h * (m1 + m2);
    printf("%f\t%f\n", x0, y0);
  }

  printf("\nValue of y at x = %f is %f", x0, y0);
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc heuns_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out
Enter initial value of x and y : 2
2.8

 Enter x at which y is required : 4

Enter step size, h : 0.5
2.500000        4.340000
3.000000        6.220667
3.500000        8.442333
4.000000        11.005184

Value of y at x = 4.000000 is 11.005184

                  Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

12. 4th Order Runge Kutta Method

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

float func(float x, float y) {
  float f;
  f = 2.0 * y / x;
  return f;
}

int main() {
  int i, n;
  float x0, y0, xp, h, m1, m2, m3, m4;

  printf("\nEnter initial value of x and y : ");
  scanf("%f%f", &x0, &y0);

  printf("\nEnter x at which y is required : ");
  scanf("%f", &xp);

  printf("\nEnter step size, h : ");
  scanf("%f", &h);

  n = (xp - x0) / h;

  for (i = 1; i ⩽ n; i++) {
    m1 = func(x0, y0);
    m2 = func(x0 + 0.5 * h, y0 + 0.5 * m1 * h);
    m3 = func(x0 + 0.5 * h, y0 + 0.5 * m2 * h);
    m4 = func(x0 + h, y0 + m3 * h);
    x0 = x0 + h;
    y0 = y0 + (m1 + 2 * m2 + 2 * m3 + m4) * h / 6;
    printf("%f\t%f\n", x0, y0);
```

```
  }
  printf("\n Value of y at x = %f is %f", x0, y0);
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc runge_kutta.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter initial value of x and y : 2
2.2

Enter x at which y is required : 4

Enter step size, h : 0.5
2.500000        3.437160
3.000000        4.949322
3.500000        6.736461
4.000000        8.798566

 Value of y at x = 4.000000 is 8.798566

                 Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

13. Gauss Elimination Method

```
#include <stdio.h>

int main() {
  int i, j, k, n;
  float A[20][20], r, x[10], sum = 0.0;

  printf("\nEnter the order of matrix : ");
  scanf("%d", &n);
  printf("\nEnter the elements of augmented matrix row-wise : \n\
n");

  for (i = 1; i ⩽ n; i++) {
    for (j = 1; j ⩽ n+1; j++) {
      printf("A[%d][%d] : ", i, j);
      scanf("%f", &A[i][j]);
    }
  }
  // Generation of upper triangle matrix:
  for (j = 1; j ⩽ n; j++) {
    for (i = 1; i ⩽ n; i++) {
      if (i > j) {
        r = A[i][j] / A[j][j];
        for (k = 1; k ⩽ n+1; k++) {
          A[i][k] = A[i][k] - r * A[j][k];
        }
      }
    }
  }
  x[n] = A[n][n+1] / A[n][n];

  // backward substitution
```

```
  for (i = n-1; i >= 1; i--) {
    sum = 0;
    for (j = i+1; j <= n; j++) {
      sum = sum + A[i][j] * x[j];
    }
    x[i] = (A[i][n+1] - sum) / A[i][i];
  }
  printf("\nThe solution is : \n");
  for (i = 1; i <= n; i++) {
    printf("\nx%d = %f\t", i, x[i]);
  }
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc gauss_elimination_method.c
[abhinaya@arch nmPractical]$ ./a.out

Enter the order of matrix : 2

Enter the elements of augmented matrix row-wise :

A[1][1] : 3
A[1][2] : 4
A[1][3] : 5
A[2][1] : 2
A[2][2] : 6
A[2][3] : 4

The solution is :

x1 = 1.400000
x2 = 0.200000

                  Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

14. Gauss Jordan Method

```
#include <stdio.h>

int main() {
  int i, j, k, n;
  float A[20][20], r, x[10];
  printf("\nEnter the size of matrix : ");
  scanf("%d", &n);
  printf("\nEnter the elements of augmented matrix row-wise : \n");

  for (i = 1; i ⩽ n; i++) {
    for (j = 1; j ⩽ n+1; j++) {
      printf("A[%d][%d] : ", i, j);
      scanf("%f", &A[i][j]);
    }
  }

  // finding diaognal matrix
  for (j = 1; j ⩽ n; j++) {
    for (i = 1; i ⩽ n; i++) {
      if(i ≠ j) {
        r = A[i][j] / A[j][j];
        for (k = 1; k ⩽ n + 1; k++) {
          A[i][k] = A[i][k] - r * A[j][k];
        }
      }
    }
  }

  printf("\nThe solution is: \n");
  for (i = 1; i ⩽ n; i++) {
    x[i] = A[i][n+1] / A[i][i];
    printf("\nx%d = %f", i, x[i]);
```

```
  }
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc gauss_jordan_method.c
[abhinaya@arch nmPractical]$ ./a.out

Enter the size of matrix : 2

Enter the elements of augmented matrix row-wise :
A[1][1] : 2
A[1][2] : 7
A[1][3] : 5
A[2][1] : 4
A[2][2] : 0
A[2][3] : 6

The solution is:

x1 = 1.500000
x2 = 0.285714

                  Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

15. Gauss Jacobi Iterative Method

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

#define f1(x, y, z) (15-y-z)/10
#define f2(x, y, z) (24-x-z)/10
#define f3(x, y, z) (33-x-y)/10

int main() {
  float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1, e1, e2, e3, e;
  int i = 1;
  printf("Enter the allowed error : \n");
  scanf("%f", &e);
  printf("\ni\tx\ty\tz\n");

  do {
    x1 = f1(x0, y0, z0);
    y1 = f2(x0, y0, z0);
    z1 = f3(x0, y0, z0);
    printf("%d\t%f\t%f\t%f\n", i, x1, y1, z1);

    // error
    e1 = fabs(x0 - x1);
    e2 = fabs(y0 - y1);
    e3 = fabs(z0 - z1);
    i++;

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

```
  printf("\n Solution : x = %f, y = %f and z = %f \n", x1, y1, z1);
  printf("\n\n\t\t  Abhinaya Aryal \n\n");
  return 0;
}
```

```
[abhinaya@arch nmPractical]$ gcc gauss_jacobi_iteration_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out
Enter the allowed error :
0.001

i       x       y       z
1       1.500000        2.400000        3.300000
2       0.930000        1.920000        2.910000
3       1.017000        2.016000        3.015000
4       0.996900        1.996800        2.996700
5       1.000650        2.000640        3.000630
6       0.999873        1.999872        2.999871

 Solution : x = 0.999873, y = 1.999872 and z = 2.999871


                Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

16. Gauss Seidal Iteration Method

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

#define f1(x, y, z) (15-y-z)/10
#define f2(x, y, z) (24-x-z)/10
#define f3(x, y, z) (33-x-y)/10

int main() {
  float x0 = 0, y0 = 0, z0 = 0, x1, y1, z1, e1, e2, e3, e;
  int i = 1;
  printf("\nEnter the allowed error : ");
  scanf("%f", &e);
  printf("\ni\tx\ty\tz\n");
  do {
    x1 = f1(x0, y0, z0);
    y1 = f2(x1, y0, z0);
    z1 = f3(x1, y1, z0);
    printf("%d\t%f\t%f\t%f\n", i, x1, y1, z1);

    e1 = fabs(x0 - x1);
    e2 = fabs(y0 - y1);
    e3 = fabs(z0 - z1);
    i++;

    x0 = x1;
    y0 = y1;
    z0 = z1;
  }while (e1 > e && e2 > e && e3 > e);

  printf("\nSolution: x = %f, y = %f and z = %f \n", x1, y1, z1);
  printf("\n\n\t\t  Abhinaya Aryal\n\n");
  return 0;
```

```
}
```

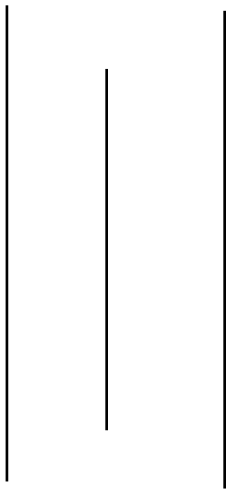
```
[abhinaya@arch nmPractical]$ gcc gauss_seidal_iteration_method.c -lm
[abhinaya@arch nmPractical]$ ./a.out

Enter the allowed error : 0.001

i       x       y       z
1       1.500000        2.250000        2.925000
2       0.982500        2.009250        3.000825
3       0.998992        2.000018        3.000099

Solution: x = 0.998992, y = 2.000018 and z = 3.000099


                Abhinaya Aryal

[abhinaya@arch nmPractical]$
```

Thank You !!

Practical Of Numerical Method

Butwal Multiple Campus

CSIT 3rd Sem.



Submitted By:-

Abhinaya Aryal
Sec 'A'
Roll no. :- 2
Symbol no. :- 27654/077