Computational Methods Lab

Faculty Name:
SACHIN GARG
Assisstant Professor
I.T. Department

Student: AMNEESH SINGH
Enrollment: 14114803121
Semester: III
Group: I6

Maharaja Agrasen Institute of Technology, PSP Area, Sector-22, Rohini, New Delhi 110086

COMPUTATIONAL METHODS LAB
PRACTICAL RECORD

Paper Code	:	ES-251
Name of the student	:	Amneesh Singh
University Enrollment number	:	14114803121
Branch	:	Information Technology
Group	:	I6

PRACTICAL DETAILS

Experiments according to the lab syllabus prescribed by GGSIPU

Exp. No.	Experiment Name	Performance Date	Date Checked	Remarks	Marks
1	Program for finding roots of f(x) = 0 using Newton Raphson Method				
2	Program for finding roots of f(x) = 0 using Bisection Method				
3	Program for finding roots of f(x) = 0 using Secant Method				
4	Program to implement Langrange Interpolation				
5	Program to implement Newton's Divided Difference formula				
6	Program for solving numerical integration by trapezoidal method				
7	Program for solving numerical integration by Simpson's 1/3 rule				
8	Program for solving numerical integration by Simpson's 3/8 rule				
9	Program for finding inverse of linear equations using Gauss Jordan method				
10	Program for finding eigen values using power method				
11	Program for solving ordinary differential equation using Renge Kutta method				

Programs are followed by their respective inputs and outputs i.e, both stdin and stdout are shown together

1 Program for finding roots of f(x) = 0 using Newton Raphson Method

```c
#include <math.h>
#include <stdio.h>
#define EPSILON 0.0000001

#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)
#define f1(x) ((1056 * x * x) - (128 * x) + 198)

double newtonRaphson(double x) {
  double h = f(x) / f1(x);
  if (f(x) == 0 || fabs(h) < EPSILON)
    return x;
  return newtonRaphson(x - h);
}

int main() {
  printf("Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is %lf",
         newtonRaphson(-4));
}
```

```
Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is 0.181818
```

2 Program for finding roots of f(x) = 0 using bisection method

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

#define EPSILON 0.0000001
#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)

double bisection(double a, double b) {
  double x;
  if (f(a) * f(b) > 0) {
    printf("The values of function at the respective initial guesses must have "
           "opposite signs");
    exit(1);
  }
  x = (a + b) / 2;
  if (f(x) == 0 || fabs(b - a) < EPSILON)
    return x;
  if (f(x) > 0)
    return bisection(x, b);
  return bisection(a, x);
}

int main() {
  printf("Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is %lf",
         bisection(1.6, -4));
}
```

```
Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is 0.181818
```

3 Program for finding roots of f(x) = 0 using secant method

```c
#include <math.h>
#include <stdio.h>
#define EPSILON 0.0000001

#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)

double secant(double a, double b) {
  double x1;
  x1 = (a * f(b) - b * f(a)) / (f(b) - f(a));
  if (f(x1) == 0 || fabs(a - b) < EPSILON)
    return x1;
  return secant(b, x1);
}

int main() {
  printf("Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is %lf", secant(1.6, -4));
}
```

```
Root for f(x) = 352x^3 - 64x^2 + 198x - 36 is 0.181818
```

4 Program to implement Langrange Interpolation.

```c
#include <stdio.h>

int main() {
  double xp, yp = 0, p;
  int i, j, n;

  printf("Number of inputs: ");
  scanf("%d", &n);

  double x[n], y[n];
  printf("Input sample space:\n");

  for (i = 0; i < n; i++) {
    printf("x%d: ", i);
    scanf("%lf", x + i);
    printf("y%d: ", i);
    scanf("%lf", y + i);
  }

  printf("Enter interpolation point x: ");
  scanf("%lf", &xp);

  for (i = 0; i < n; i++) {
    p = 1;
    for (j = 0; j < n; j++) {
      if (i != j) {
        p *= (xp - x[j]) / (x[i] - x[j]);
      }
    }
    yp += p * y[i];
  }
  printf("Interpolated value for %lf is %lf.", xp, yp);
}
```

```
Number of inputs: 4
Input sample space:
x0: 0
y0: 2
x1: 1
y1: 3
x2: 2
y2: 12
x3: 5
y3: 147
Enter interpolation point x: 3
Interpolated value for 3.000000 is 35.000000.
```

5 Program to implement Newton's Divided Difference formula.

```c
#include <stdio.h>

int main() {
  int n, i, j = 1;
  double xp, yp, f1, f2 = 0;

  printf("Enter the number of inputs: ");
  scanf("%d", &n);

  double x[n], y[n];

  printf("Enter input values:\n");
  for (i = 0; i < n; i++) {
    printf("x%d=", i);
    scanf("%lf", x + i);
    printf("y%d=", i);
    scanf("%lf", y + i);
  }

  yp = y[0];

  printf("Enter interpolation point x: ");
  scanf("%lf", &xp);

  do {
    for (i = 0; i < n - 1; i++)
      y[i] = ((y[i + 1] - y[i]) / (x[i + j] - x[i]));

    f1 = 1;

    for (i = 0; i < j; i++) {
      f1 *= (xp - x[i]);
    }

    f2 += (y[0] * f1);
    j++;
  } while ((--n) > 1);

  yp += f2;

  printf("Interpolated value for %lf is %lf.", xp, yp);
}
```

```
Enter the number of inputs: 4
Enter input values:
x0=3
y0=9
x1=5
y1=12
x2=9
y2=666
x3=15
y3=10245
Enter interpolation point x: 999
Interpolated value for 999.000000 is 9525764925.000002.
```

6 Program for solving numerical integration by trapezoidal method.

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

#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)

double trapezoidal_integral(double a, double b, double n) {
  double h = (b - a) / n;

  double s = (f(a) + f(b)) / 2;

  // Add the other heights

  for (int i = 1; i < n; i++)
    s += f(a + i * h);

  return s * h;
}

int main() {
  printf("The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 "
         "to x=4 is %lf",
         trapezoidal_integral(3, 4, 10000));
}
```

```
The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 to x=4 is 1425.444900
```

7 Program for solving numerical integration by Simpson's 1/3 rule.

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

#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)

double simpsons_1_3(double a, double b, double n) {
  double h = (b - a) / n;

  double s = f(a) + f(b);

  // Add the other heights

  for (int i = 1; i < n; i += 2)
    s += 4 * f(a + i * h);

  for (int i = 2; i < n; i += 2)
    s += 2 * f(a + i * h);

  return (h / 3) * s;
}

int main() {
  printf("The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 "
         "to x=4 is %lf",
         simpsons_1_3(3, 4, 10000));
}
```

```
The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 to x=4 is 1424.963267
```

8 Program for solving numerical integration by Simpson's 3/8 rule.

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

#define f(x) ((352 * x * x * x) - (64 * x * x) + (198 * x) - 36)

double simpsons_1_3(double a, double b, double n) {
  double h = (b - a) / n;

  double s = f(a) + f(b);

  // Add the other heights

  for (int i = 1; i < n; i++)
    s += 3 * f(a + i * h);

  for (int i = 3; i < n; i += 3)
    s -= f(a + i * h);

  return (h / 8) * 3 * s;
}

int main() {
  printf("The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 "
         "to x=4 is %lf",
         simpsons_1_3(3, 4, 10000));
}
```

```
The area under the curve f(x) = 352x^3 - 64x^2 + 198x - 36 from x=3 to x=4 is 1425.048025
```

9 Program for finding inverse of linear equations using Gauss Jordan method.

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

double **inverse(double **matrix, int order) {
  double **inverse = calloc(order, sizeof(double *));
  for (int i = 0; i < order; i++) {
    inverse[i] = calloc(2 * order, sizeof(double));
    inverse[i][order + i] = 1;
    memcpy(inverse[i], matrix[i], order * sizeof(double));
  }

  for (int i = 0; i < order; i++) {
    for (int j = 0; j < order; j++) {
      if (i == j)
        continue;
      double r = inverse[j][i] / inverse[i][i];

      for (int k = 0; k < order * 2; k++)
        inverse[j][k] -= r * inverse[i][k];
    }
  }

  for (int i = 0; i < order; i++) {
    for (int j = 0; j < order; j++)
      inverse[i][j + order] /= inverse[i][i];
  }

  return inverse;
}

int main() {
  const int ORDER = 3;

  double **matrix = malloc(ORDER * sizeof(double *));
  for (int i = 0; i < ORDER; i++)
    matrix[i] = malloc(ORDER * sizeof(double));

  matrix[0][0] = 92, matrix[0][1] = 4.5, matrix[0][2] = 61; // 92x + 4.5y + 61z = 0
  matrix[1][0] = -2, matrix[1][1] = 0, matrix[1][2] = 92387; // -2x + 92387z = 0
  matrix[2][0] = -2, matrix[2][1] = 0, matrix[2][2] = -23; // -2x - 23z = 0

  double **inv = inverse(matrix, ORDER);

  for (int i = 0; i < ORDER; i++) {
    for (int j = 0; j < ORDER; j++)
      printf("%lf ", inv[i][j + ORDER]);
    printf("\n");

    free(inv[i]);
```

```
    free(matrix[i]);
  }

  free(inv);
  free(matrix);
}
```

```
-0.000000 -0.000124 -0.499876
0.222222 0.002398 10.219825
-0.000000 0.000011 -0.000011
```

10 Program for finding eigen values using power method.

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

#define EPSILON 0.0000001

double *eigen(double **matrix, int order) {
  double mxerr = 0;
  double *prev = calloc(order, sizeof(double));
  *prev = 1;

  do {
    double *eigen = calloc(order, sizeof(double));

    for (int i = 0; i < order; i++)
      for (int j = 0; j < order; j++)
        eigen[i] += matrix[i][j] * prev[j];

    double mx = fabs(eigen[0]);
    for (int i = 1; i < order; i++)
      mx = fabs(eigen[i]) > mx ? fabs(eigen[i]) : mx;

    for (int i = 0; i < order; i++)
      eigen[i] /= mx;

    double error[order];

    for (int i = 0; i < order; i++)
      error[i] = fabs(eigen[i] - prev[i]);

    mxerr = error[0];
    for (int i = 1; i < order; i++)
      mxerr = error[i] > mxerr ? error[i] : mxerr;

    free(prev);
    prev = eigen;

  } while (mxerr > EPSILON);

  return prev;
}

int main() {
  const int ORDER = 3;

  double **matrix = malloc(ORDER * sizeof(double *));
  for (int i = 0; i < ORDER; i++)
    matrix[i] = malloc(ORDER * sizeof(double));

  matrix[0][0] = 21, matrix[0][1] = -99, matrix[0][2] = 0;
  matrix[1][0] = -12, matrix[1][1] = 64, matrix[1][2] = 0;
```

```
  matrix[2][0] = 9, matrix[2][1] = -0.5, matrix[2][2] = 4;

  double *egn = eigen(matrix, ORDER);

  for (int i = 0; i < ORDER; i++) {
    printf("%lf ", egn[i]);
    free(matrix[i]);
  }

  free(egn);
  free(matrix);
}
```

```
1.000000 -0.627508 0.117712
```

11 Program for solving ordinary differential equation using Renge Kutta method.

```c
#include <stdio.h>

double f(double x, double y) { return (y * y - x) / (y + x * y * y); };

double rungen_kutta_4(double x0, double y0, double xn, int steps) {
  double h = (xn - x0) / steps;

  while (steps--) {
    double k1 = f(x0, y0) * h;
    double k2 = f(x0 + h / 2, y0 + k1 / 2) * h;
    double k3 = f(x0 + h / 2, y0 + k2 / 2) * h;
    double k4 = f(x0 + h, y0 + k3) * h;

    double k = (k1 + k4 + 2 * (k2 + k3)) / 6;

    x0 += h, y0 += k;
  }

  return y0;
}

int main() {
  double x0 = 0, y0 = 5, xn = 1784;
  int steps = 1000000;

  printf("Value of y at x=%0.4lf is y=%0.4lf", xn,
         rungen_kutta_4(x0, y0, xn, steps));
}
```

```
Value of y at x=1784.0000 is y=-14.6222
```