Lab 4 (Output) :-

Gauss Elimination method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc gausselimination.c -o gausselimination
 && "/media/brahma/Store_2/NM/"gausselimination
Enter the number of variables: 3
Enter the augmented matrix (row-wise):
1 -2 3 -4
2 5 1 8
5 1 -6 9
Solution:
x1 = 0.75
x2 = 1.43
x3 = -0.63
```

Gauss Jordan method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc gaussjordan.c -o gaussjordan && "/medi
a/brahma/Store_2/NM/"gaussjordan
Enter number of variables: 3
Enter augmented matrix:
-1 2 4 -2
4 9 -6 3
2 6 -9 5
Solution:
x1 = -0.21
x2 = 0.04
x3 = -0.57
```

Doolittle LU decomposition method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc dollitle.c -o dollitle && "/media/brahma/Store_2
/NM/"dollitle
Enter matrix size: 3
Enter matrix elements row-wise:
2 4 7
2 8 0
-3 9 4
L matrix:
  1.00   0.00   0.00
  1.00   1.00   0.00
 -1.50   2.25   1.00
U matrix:
  2.00   4.00   7.00
  0.00   4.00  -7.00
  0.00   0.00  30.25
```

Matrix inversion method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc matrixinversion.c -o matrixinversion && "/media/
brahma/Store_2/NM/"matrixinversion
Enter matrix size: 2
Enter matrix elements row-wise:
3 8
4 9
Inverse matrix:
-1.80 1.60
0.80 -0.60
```

Gauss Jacobi iterative method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ gcc gaussjacobi.c -o gaussjacobi -lm
● Bagish@brahma:/media/brahma/Store_2/NM$ ./gaussjacobi
Enter number of equations: 3
Enter coefficients of the matrix A row-wise:
4 -1 0
-1 4 -1
0 -1 3
Enter constant terms (b): 15 10 10
Enter initial guesses: 0 0 0
Enter maximum iterations and tolerance: 25 0.0001
Solution:
x[1] = 5.0000
x[2] = 5.0000
x[3] = 5.0000
```

Gauss-Seidel iterative method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ gcc gausssiedal.c -o gausssiedal -lm
● Bagish@brahma:/media/brahma/Store_2/NM$ ./gausssiedal
  Enter n, max_iter, tol: 3 25 0.0001
  Enter matrix A:
  4 -1 0 -1 4 -1 0 -1 3
  Enter vector b and initial x:
  15 0 10 0 10 0
  x[1] = 5.0000
  x[2] = 5.0000
  x[3] = 5.0000
```

Eigen value and Eigen vector method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ gcc eigenvalue.c -o eigenvalue -lm
● Bagish@brahma:/media/brahma/Store_2/NM$ ./eigenvalue
  Enter the size of the matrix, maximum iterations, and tolerance: 3 1000 0.0001
  Enter the matrix A:
  4 1 2
  1 3 0
  2 0 3
  Eigenvalue: 5.7913
  Eigenvector: 0.7805 0.2797 0.5592
```

Lab 5 (Output)

Taylor method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc taylor.c -o taylor && "/media/brahma/Store_2/NM/
"taylor
Enter initial conditions (x0, y0), step size (h), and number of steps (n): 0 1 0.1 5
x = 0.0000, y = 1.0000
x = 0.1000, y = 1.1000
x = 0.2000, y = 1.2200
x = 0.3000, y = 1.3620
x = 0.4000, y = 1.5282
x = 0.5000, y = 1.7210
```

Picard's method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc picard.c -o picard && "/media/brahma/Store_2/NM/
"picard
Enter initial conditions (x0, y0), step size (h), and number of steps (n): 0 1 0.1 5
x = 0.0000, y = 1.0000
x = 0.1000, y = 1.1000
x = 0.2000, y = 1.2200
x = 0.3000, y = 1.3620
x = 0.4000, y = 1.5282
x = 0.5000, y = 1.7210
```

Euler's method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc euler.c -o euler && "/media/brahma/Store_2/NM/"e
uler
Enter initial conditions (x0, y0), step size (h), and number of steps (n):  0 1 0.1 5
x = 0.0000, y = 1.0000
x = 0.1000, y = 1.1000
x = 0.2000, y = 1.2200
x = 0.3000, y = 1.3620
x = 0.4000, y = 1.5282
x = 0.5000, y = 1.7210
```

Heun's method :-

```
● Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc heunn.c -o heunn && "/media/brahma/Store_2/NM/"h
eunn
Enter initial conditions (x0, y0), step size (h), and number of steps (n): 0 1 0.1 5
x = 0.0000, y = 1.0000
x = 0.1000, y = 1.1100
x = 0.2000, y = 1.2421
x = 0.3000, y = 1.3985
x = 0.4000, y = 1.5818
x = 0.5000, y = 1.7949
```

Fourth order runge kutta method :-

```
Bagish@brahma:/media/brahma/Store_2/NM$ cd "/media/brahma/Store_2/NM/" && gcc rungekutta.c -o rungeku
tta && "/media/brahma/Store_2/NM/"rungekutta
Enter initial conditions (x0, y0), step size (h), and number of steps (n): 0 1 0.1 5
x = 0.0000, y = 1.0000
x = 0.1000, y = 1.1103
x = 0.2000, y = 1.2428
x = 0.3000, y = 1.3997
x = 0.4000, y = 1.5836
x = 0.5000, y = 1.7974
```

Lab 6 (Output) :-

Laplace's equation :-

```
Bagish@brahma:/media/brahma/Store_2/NM$ gcc laplace.c -o laplace -lm
Bagish@brahma:/media/brahma/Store_2/NM$ ./laplace
Enter grid size (rows and cols): 2 2
Enter boundary conditions (top, bottom, left, right): 10 10 12 12
10 10 12 12
Converged after 1 iterations.
10.00 12.00
10.00 12.00
```

Poisson's equation :-

```
Bagish@brahma:/media/brahma/Store_2/NM$ gcc poisson.c -o poisson -lm
Bagish@brahma:/media/brahma/Store_2/NM$ ./poisson
Enter grid size (n) and tolerance: 3 0.01
0.00 0.00 0.00
0.00 -6.25 0.00
0.00 0.00 0.00
```