APPENDIX

1. Bisection Method using C Program

```
/*
Program: Finding real roots of nonlinear equation using Bisection Method
#include<stdio.h>
#include<conio.h>
#include<math.h>
/*Defining equation to be solved, change this equation to solve another
problem.
*/
#define f(x) cos(x) – x * exp(x)
void main()
{
        float xl, xu, xn, fl, fu, fn, e;
        int step = 1;
        /* Inputs */
        up:
        printf("\n Enter first guess value: ");
        scanf("%f", &xl);
        printf("\n Enter second guess value: ");
        scanf("%f", &xu);
        printf("\n Enter tolerable error: ");
        scanf("%f", &e);
        /* Calculating Functional Value */
        fl = f(xl);
        fu = f(xu);
        /* Checking whether given guesses brackets the root or not.*/
        if( fl * fu > 0.0)
        {
                printf("\n !!!  Incorrect Initial Guesses !!! \n");
                goto up;
        }
        /* Implementing Bisection Method */
        printf("\n\nStep\t\txl\t\txu\t\txn\t\tf(xn)\n ");
        do
        {
                xn = (xl + xu)/2;
                fn = f(xn);
```

```
            printf("%d\t\t%f\t%f\t%f\t%f\n",step, xl, xu, xn, fn);
            If( fl * fn < 0)
            {
                  xu = xn;
                  fu = fn;
            }
            else
            {
                  xl = xn;
                  fl = fn;
            }
            step = step + 1;
      }
      while(fabs(fn)>e);
      printf("\n Hence, the required Root is = %f", xn);
      getch();
}
```

Output:

```
Enter first guess value: 0
Enter second guess value: 4
Enter tolerable error: 0.0001
```

Step	xl	xu	xn	f(xn)
1	0.000000	4.000000	2.000000	−15.194259
2	0.000000	2.000000	1.000000	−2.177979
3	0.000000	1.000000	0.500000	0.053222
4	0.500000	1.000000	0.750000	−0.856061
5	0.500000	0.750000	0.625000	−0.356691
6	0.500000	0.625000	0.562500	−0.141294
7	0.500000	0.562500	0.531250	−0.041512
8	0.500000	0.531250	0.515625	0.006475
9	0.515625	0.531250	0.523438	−0.017362
10	0.515625	0.523438	0.519531	−0.005404
11	0.515625	0.519531	0.517578	0.000545
12	0.517578	0.519531	0.518555	−0.002427
13	0.517578	0.518555	0.518066	−0.000940
14	0.517578	0.518066	0.517822	−0.000197
15	0.517578	0.517822	0.517700	0.000174
16	0.517700	0.517822	0.517761	−0.000012

Hence, the required Root is = 0.517761

Output:

```
Enter first guess value: -2
Enter second guess value: 3
Enter tolerable error: 0.0001
!!! Incorrect Initial Guesses !!!
Enter first guess value: 0
Enter second guess value: 1
Enter tolerable error: 0.0001
```

Step	xl	xu	xn	f(xn)
1	0.000000	1.000000	0.500000	0.053222
2	0.500000	1.000000	0.750000	-0.856061
3	0.500000	0.750000	0.625000	-0.356691
4	0.500000	0.625000	0.562500	-0.141294
5	0.500000	0.562500	0.531250	-0.041512
6	0.500000	0.531250	0.515625	0.006475
7	0.515625	0.531250	0.523438	-0.017362
8	0.515625	0.523438	0.519531	-0.005404
9	0.515625	0.519531	0.517578	0.000545
10	0.517578	0.519531	0.518555	-0.002427
11	0.517578	0.518555	0.518066	-0.000940
12	0.517578	0.518066	0.517822	-0.000197
13	0.517578	0.517822	0.517700	0.000174
14	0.517700	0.517822	0.517761	-0.000012

Hence, the required Root is = 0.517761

2. Bisection Method Using C++ Program

```
#include<iostream>
#include<iomanip>
#include<math.h>
/*
Defining equation to be solved, change this equation to solve another
problem.
*/
#define f(x) cos(x) - x * exp(x)
using namespace std;
int main()
{
        /* Declaring required variables */
        float xl, xu, xn, fl, fu, fxn, e;
        int step = 1;
        /* Setting precision and writing floating point values in fixed-point
        notation. */
        cout<< setprecision(6)<< fixed;
```

```
/* Inputs */
up:
cout<<"Enter first guess value: ";
cin>>xl;
cout<<"Enter second guess value: ";
cin>>xu;
cout<<"Enter tolerable error: ";
cin>>e;
/* Calculating Functional Value */
fl = f(xl);
fu = f(xu);
/* Checking whether given guesses brackets the root or not. */
if( fl * fu > 0.0)
{
        cout<<"Incorrect Initial Guesses, Try Other values "<< endl<<
        endl;
        goto up;
}
/* Implementing Bisection Method */
cout<<" Using Bisection Method"<< endl;
do
{
        /* Bisecting Interval */
        xn = (xl + xu)/2;
        fxn = f(xn);
        cout<<"Iteration-"<< step<<":\t xn = "<< setw(10)<< xn<<"
        and f(xn) = "<< setw(10)<< f(xn)<< endl;
        if( fl * fxn < 0)
        {
                xu = xn;
        }
        else
        {
                xl = xn;
        }
        step = step + 1;
}
while(fabs(fxn)>e);
cout<< endl<<"Then, Required Root of given equation is: "<< xn<<
endl;
return 0;
}
```

Output:

```
Enter first guess value: 0
Enter second guess value
Enter tolerable error: 0.0

Step    xl
1       0.000000
2       0.500000
3       0.500000
4       0.500000
5       0.500000
6       0.500000
7       0.515625
8       0.515625
9       0.515625
10      0.517578
11      0.517578
12      0.517578
13      0.517578
14      0.517700
15      0.517700
16      0.517731
17      0.517746
18      0.517754
```

Hence, the require

3. C Progra

```
/*Program: Findi
False Position Me
/* Header Files */
#include<stdio.h>
#include<conio.h
#include<math.h
/* Defining equa
Change this equ
#define f(x) x*
int main()
{
        float x0,
        int step
        /* Input
        up:
```

Output:

```
Enter first guess value: 0
Enter second guess value: 1
Enter tolerable error: 0.00001
```

Step	xl	xu	xn	f(xn)
1	0.000000	1.000000	0.500000	0.053222
2	0.500000	1.000000	0.750000	-0.856061
3	0.500000	0.750000	0.625000	-0.356691
4	0.500000	0.625000	0.562500	-0.141294
5	0.500000	0.562500	0.531250	-0.041512
6	0.500000	0.531250	0.515625	0.006475
7	0.515625	0.531250	0.523438	-0.017362
8	0.515625	0.523438	0.519531	-0.005404
9	0.515625	0.519531	0.517578	0.000545
10	0.517578	0.519531	0.518555	-0.002427
11	0.517578	0.518555	0.518066	-0.000940
12	0.517578	0.518066	0.517822	-0.000197
13	0.517578	0.517822	0.517700	0.000174
14	0.517700	0.517822	0.517761	-0.000012
15	0.517700	0.517761	0.517731	0.000081
16	0.517731	0.517761	0.517746	0.000035
17	0.517746	0.517761	0.517754	0.000011
18	0.517754	0.517761	0.517757	-0.000000

Hence, the required Root is = 0.517757

3. C Program for Regula Falsi (False Position) Method

```
/*Program: Finding real roots of nonlinear equation using Regula Falsi or
False Position Method
 /* Header Files */
 #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("\n Enter first initial guess: ");
```

```
    scanf("%f",&x0);
    printf("\n Enter second initial guess: ");
    scanf("%f", &x1);
    printf("\n Enter tolerable error: ");
    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("\n 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("\n\n Required Root is: %f", x2);
    getch();
    return 0;
}
```

Output:

```
Enter first initial guess: 2
Enter second initial guess: 5
Enter tolerable error: 0.001
```

Step	x0	x1	x2	f(x2)
1	2.000000	5.000000	2.620100	-0.103965
2	2.620100	5.000000	2.723246	-0.015152
3	2.723246	5.000000	2.738179	-0.002151
4	2.738179	5.000000	2.740297	-0.000304

Required Root is: 2.740297

Output:

Enter first initial guess: 2
Enter second initial guess: 7
Enter tolerable error: 0.000001

Step	x0	x1	x2	f(x2)
1	2.000000	7.000000	2.562648	-0.152674
2	2.562648	7.000000	2.701805	-0.033755
3	2.701805	7.000000	2.732353	-0.007227
4	2.732353	7.000000	2.738884	-0.001537
5	2.738884	7.000000	2.740272	-0.000326
6	2.740272	7.000000	2.740567	-0.000069
7	2.740567	7.000000	2.740629	-0.000015
8	2.740629	7.000000	2.740643	-0.000003
9	2.740643	7.000000	2.740645	-0.000001

Required Root is: 2.740645

4. C++ Program for Regula Falsi (False Position) Method

```
nclude<iostream>
nclude<iomanip>
nclude<math.h>
```

...solve another

Output:

```
Enter first guess: 1
Enter second guess: 4
Enter tolerable error: 0.0001
False Position Method
Iteration-1:    x =  2.494868 and f(x) =  -0.209419
Iteration-2:    x =  2.717208 and f(x) =  -0.020397
Iteration-3:    x =  2.738505 and f(x) =  -0.001867
Iteration-4:    x =  2.740451 and f(x) =  -0.000170
Iteration-5:    x =  2.740628 and f(x) =  -0.000015
Root is: 2.740628
```

5. C Program for Newton Raphson Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
/*Defining equation to be solved, change this equation to solve another
problem. */
#define  f(x)  3*x – cos(x) –1
/* Defining derivative of g(x).
As you change f(x), change this function also. */
#define  g(x)  3 + sin(x)
int main()
{
        float x0, x1, f0, f1, g0, e;
        int step = 1, N;
        /* Inputs */
        printf("\n Enter initial guess:");
        scanf("%f", &x0);
        printf("\n Enter tolerable error:");
        scanf("%f", &e);
        printf("\n Enter maximum iteration:\n");
        scanf("%d", &N);
        /* Implementing Newton Raphson Method */
        printf("\n Step\t\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
        do
        {
                g0 = g(x0);
                f0 = f(x0);
                if(g0 == 0.0)
```

```
        {
                printf("\n Mathematical Error.");
                exit(0);
        }
        x1 = x0 - f0/g0;
        printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,f0,x1,f1);
        x0 = x1;
        step = step+1;
        if(step > N)
        {
                printf("\n Not Convergent.");
                exit(0);
        }
        f1 = f(x1);
    } while(fabs(f1)>e);
    printf("\n\n Hence the required Root is: %f", x1);
    getch();
}
```

Output:

```
Enter initial guess: 0
Enter tolerable error: 0.0001
Enter maximum iteration: 6
```

Step	x0	f(x0)	x1	f(x1)
1	0.000000	-2.000000	0.666667	0.000000
2	0.666667	0.214113	0.607493	0.214113
3	0.607493	0.001397	0.607102	0.001397

```
Hence the required Root is: 0.607102
```

6. C++ Program for Newton Raphson Method

```
#include<iostream>
#include<iomanip>
#include<math.h>
#include<stdlib.h>
```

4	0.607102	0.001397	0.607102	0.214113
5	0.607102	0.000000	0.607102	0.001397
6	0.607102	-0.000000	0.607102	0.000000
7	0.607102	0.000000	0.607102	-0.000000
		-0.000000	0.607102	0.000000
				-0.000000

Not Convergent.

7. C Program for Secant Method

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
/* Defining equation to be solved.
Change this equation to solve another problem. */
#define f(x) cos(x) - x * exp(x)
int main()
{
        float x0, x1, x2, f0, f1, f2, e;
        int step = 1, N;
        /* Inputs */
        printf("\n Enter first initial guess : ");
        scanf("%f", &x0);
        printf("\n Enter second initial guess : ");
        scanf("%f", &x1);
```

```
    printf("\n Enter tolerable error: ");
    scanf("%f", &e);
    printf("\n Enter maximum iteration: ");
    scanf("%d", &N);
    /* Implementing Secant Method */
    printf("\nStep\t\tx0\t\tx1\t\tx2\t\tf(x2)\n");
    do
    {
        f0 = f(x0);
        f1 = f(x1);
        if(f0 == f1)
        {
            printf("Mathematical Error.");
            exit(0);
        }
        x2 = x1 - (x1 - x0) * f1/(f1-f0);
        f2 = f(x2);
        printf("%d\t\t%f\t%f\t%f\t%f\n",step,x0,x1,x2, f2);
        x0 = x1;
        f0 = f1;
        x1 = x2;
        f1 = f2;
        step = step + 1;
        if(step > N)
        {
            printf("Not Convergent.");
            exit(0);
        }
    }
    while(fabs(f2)>e);
    printf("\nRoot is: %f", x2);
    getch();
}
```

Output:

```
Enter first initial guess: 0
Enter second initial guess: 2
Enter tolerable error: 0.0001
Enter maximum iteration: 10
```

Step	x0	x1	x2	f(x2)
1	0.000000	2.000000	0.123501	0.852648
2	2.000000	0.123501	0.223208	0.696164
3	0.123501	0.223208	0.666784	-0.513057
4	0.223208	0.666784	0.478581	0.115324
5	0.666784	0.478581	0.513121	0.014050
6	0.478581	0.513121	0.517913	-0.000473
7	0.513121	0.517913	0.517757	0.000002

Root is: 0.517757

8. C++ Program for Secant Method

```
#include<iostream>
#include<iomanip>
#include<math.h>
#include<stdlib.h>
```

```
gent.";
        }
    }
    while(fabs(f2)>e);
    cout<< endl<<"Hence the required Root is: "<< x2;
    return 0;
}
```

Output:

```
Enter first guess: 0
Enter second guess: 1
Enter tolerable error: 0.000001
Enter maximum iteration: 12
Iteration–1:  x2 =  0.314665 and f(x2) =  0.519871
Iteration–2:  x2 =  0.446728 and f(x2) =  0.203545
Iteration–3:  x2 =  0.531706 and f(x2) = –0.042931
Iteration–4:  x2 =  0.516904 and f(x2) =  0.002593
Iteration–5:  x2 =  0.517747 and f(x2) =  0.000030
Iteration–6:  x2 =  0.517757 and f(x2) =  0.000000
Hence the required Root is: 0.517757
```

9. Fixed Point Iteration Method using C Program

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

```
/* Define function f(x) which is to be solved */
#define  f(x)  cos(x)-3*x+1
/* Write f(x) as x = g(x) and 0define g(x) here */
#define  g(x)  (1+cos(x))/3
int main()
{
        int step=1, N;
        float x0, x1, e;
        /* Inputs */
        printf(" Enter initial guess: ");
        scanf("%f", &x0);
        printf("\n Enter tolerable error: ");
        scanf("%f", &e);
        printf("\n Enter maximum iteration: ");
        scanf("%d", &N);
        printf("\nStep\tx0\t\tf(x0)\t\tx1\t\tf(x1)\n");
        do
        {
                x1 = g(x0);
                printf("%d\t%f\t%f\t%f\t%f\n",step, x0, f(x0), x1, f(x1));
                step = step + 1;
                if(step>N)
                {
                        printf("\n Not Convergent.");
                        exit(0);
                }
                x0 = x1;
        }
        while( fabs(f(x1)) > e);
        printf("\n Hence the Required Root having tolerable error %f is
        %f .", e, x1);
        getch();
        return(0);
}
```

Output:

Initial guess: 0
Enter tolerable error: 0.001
Enter maximum iteration: 7

Step	x0	f(x0)	x1	f(x1)
1	0.000000	2.000000	0.666667	-0.214113
2	0.666667	-0.214113	0.595296	0.042095
3	0.595296	0.042095	0.609328	-0.007950
4	0.609328	-0.007950	0.606678	0.001514
5	0.606678	0.001514	0.607182	-0.000288

Hence the Required Root having tolerable error 0.001000 is 0.607182.

10. Fixed Point Iteration Method using C++ Program

```
#include<iostream>
#include<iomanip>
#include<math.h>
#include<stdlib.h>
/* Define function f(x) which is to be solved */
#define  f(x)  cos(x)-3*x+1
/* Write f(x) as x = g(x) and define g(x) here */
#define  g(x)  (1+cos(x))/3
using namespace std;
int main()
```

```
    return(0);
}
```

Output:

Enter initial guess: 2
Enter tolerable error: 0.001
Enter maximum iteration: 9

Step	x0	f(x0)	x1	f(x1)
1	2.000000	-5.416147	0.194618	1.397269
2	0.194618	1.397269	0.660374	-0.191359
3	0.660374	-0.191359	0.596588	0.037495
4	0.596588	0.037495	0.609086	-0.007086
5	0.609086	-0.007086	0.606724	0.001349
6	0.606724	0.001349	0.607174	-0.000257

Hence the Required Root having tolerable error 0.001000 is 0.607174.

11. Gauss Elimination Method using C Program

```
#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];
                }
        }
}
/* Obtaining Solution by Back Subsitution */
x[n] = a[n][n+1]/a[n][n];
for(i=n-1;i>=1;i--)
{
        x[i] = a[i][n+1];
        for(j=i+1;j<=n;j++)
        {
                x[i] = x[i] - a[i][j]*x[j];
        }
        x[i] = x[i]/a[i][i];
}
/* Displaying Solution */
printf("\n Solution:\n");
for(i=1;i<=n;i++)
{
        printf("x[%d] = %0.3f\n",i, x[i]);
}
```

```
        getch();
        return(0);
}
```

Output for example 4.2:

Enter number of unknowns: 3
a[1][1] = 1
a[1][2] = 4
a[1][3] = –1
a[1][4] = –5
a[2][1] = 1
a[2][2] = 1
a[2][3] = –6
a[2][4] = –12
a[3][1] = 3
a[3][2] = –1
a[3][3] = –1
a[3][4] = 4

Solution:
x[1] = 1.648
x[2] = –1.141
x[3] = 2.085

12. Gauss Jordan Method using C Program

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define SIZE 10
int main()
{
        float a[SIZE][SIZE], x[SIZE], ratio;
        int i,j,k,n;
        /* Inputs */
```

Output for example 4.2:

Enter number of unknowns: 3
a[1][1] = 1
a[1][2] = 4
a[1][3] = -1
a[1][4] = -5
a[2][1] = 1
a[2][2] = 1
a[2][3] = -6
a[2][4] = -12
a[3][1] = 3
a[3][2] = -1
a[3][3] = -1
a[3][4] = 4

Solution:
x[1] = 1.648
x[2] = -1.141
x[3] = 2.085

12. Gauss Jordan Method using C Program

```
#include<stdio.h>
#include<conio.h>
#include<math.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 */
    printf("\nEnter coefficients of Augmented Matrix:\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]);
        }
}
/* Applying Gauss Jordan Elimination */
for(i=1;i<=n;i++)
{
        if(a[i][i] == 0.0)
        {
                printf("Mathematical Error!");
        }
        for(j=1;j<=n;j++)
        {
                if(i!=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];
                        }
                }
        }
}
/* Obtaining Solution */
for(i=1;i<=n;i++)
{
        x[i] = a[i][n+1]/a[i][i];
}
/* Displaying Solution */
printf("\n Required Solution: \n");
for(i=1;i<=n;i++)
{
        printf("x[%d] = %0.3f\n",i, x[i]);
}
getch();
return(0);
}
```

Output for Example 4.4:

```
Enter number of unknowns: 3
Enter coefficients of Augmented Matrix:
a[1][1] = 1
a[1][2] = 1
a[1][3] = 1
a[1][4] = 9
a[2][1] = 2
a[2][2] = -3
a[2][3] = 4
a[2][4] = 13
a[3][1] = 3
a[3][2] = 4
a[3][3] = 5
a[3][4] = 40

Required Solution:
x[1] = 1.000
x[2] = 3.000
x[3] = 5.000
```

13. Power Method using C Program

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define SIZE 10
int main()
{
        float a[SIZE][SIZE], x[SIZE],x_new[SIZE];
        float temp, lambda_new, lambda_old, error;
        int i,j,n, step=1;
        /* Inputs */
        printf("Enter Order of Matrix: ");
        scanf("%d", &n);
        printf("\n Enter Tolerable Error: ");
        scanf("%f", &error);
        /* Reading Matrix */
        printf("\n Enter Coefficient of Matrix:\n");
        for(i=1;i<=n;i++)
        {
                for(j=1;j<=n;j++)
```

```
        {
                printf("a[%d][%d]=",i,j);
                scanf("%f", &a[i][j]);
        }
}
/* Reading Intial Guess Vector */
printf("\n Enter Initial Guess Vector:\n");
for(i=1;i<=n;i++)
{
        printf("x[%d]=",i);
        scanf("%f", &x[i]);
}
/* Initializing Lambda_Old */
lambda_old = 1;
/* Multiplication */
up:
for(i=1;i<=n;i++)
{
        temp = 0.0;
        for(j=1;j<=n;j++)
        {
                temp = temp + a[i][j]*x[j];
        }
        x_new[i] = temp;
}
/* Replacing */
for(i=1;i<=n;i++)
{
        x[i] = x_new[i];
}
/* Finding Largest */
lambda_new = fabs(x[1]);
for(i=2;i<=n;i++)
{
        if(fabs(x[i])>lambda_new)
        {
                lambda_new = fabs(x[i]);
        }
}
/* Normalization */
for(i=1;i<=n;i++)
```

```
        {
                x[i] = x[i]/lambda_new;
        }
        /* Display */
        printf("\n\nSTEP-%d:\n", step);
        printf("\n Eigen Value = %f\n", lambda_new);
        printf("\n Eigen Vector:\n");
        for(i=1;i<=n;i++)
        {
                printf("%f\t", x[i]);
        }
        /* Checking Accuracy */
        if(fabs(lambda_new-lambda_old)>error)
        {
                lambda_old=lambda_new;
                step++;
                goto up;
        }
        getch();
        return(0);
}
```

Output for Example 4.19:

```
Enter Order of Matrix: 3
Enter Tolerable Error: 0.001
Enter Coefficient of Matrix:
a[1][1] = 2
a[1][2] = -1
a[1][3] = 0
a[2][1] = -1
a[2][2] = 2
a[2][3] = -1
a[3][1] = 0
a[3][2] = -1
a[3][3] = 2

Enter Initial Guess Vector:
x[1] = 1
x[2] = 0
x[3] = 0
```

STEP–1:
Eigen Value = 2.000000

Eigen Vector:
1.000000 –0.500000 0.000000

STEP–2:
Eigen Value = 2.500000

Eigen Vector:
1.000000 –0.800000 0.200000

STEP–3:
Eigen Value = 2.800000

Eigen Vector:
1.000000 –1.000000 0.428571

STEP–4:
Eigen Value = 3.428571

Eigen Vector:
0.875000 –1.000000 0.541667

STEP–5:
Eigen Value = 3.416667

Eigen Vector:
.804878 –1.000000 0.609756

EP–6:
igen Value = 3.414634

Eigen Vector:
0.764286 –1.000000 0.650000

STEP–7:
Eigen Value = 3.414286

Eigen Vector:
0.740586 –1.000000 0.673640

14. C Program for Fitting y = a + bx

```
#include<stdio.h>
#include<conio.h>
#define S 50
int main()
{
        int n, i;
        float x[S], y[S], sumX=0, sumX2=0, sumY=0, sumXY=0, a, b;
        /* Input */
```

```
STEP-1:
Eigen Value = 2.000000
Eigen Vector:
1.000000        -0.500000        0.000000
STEP-2:
Eigen Value = 2.500000
Eigen Vector:
1.000000        -0.800000        0.200000
STEP-3:
Eigen Value = 2.800000
Eigen Vector:
1.000000        -1.000000        0.428571
STEP-4:
Eigen Value = 3.428571
Eigen Vector:
0.875000        -1.000000        0.541667
STEP-5:
Eigen Value = 3.416667
Eigen Vector:
0.804878        -1.000000        0.609756
STEP-6:
Eigen Value = 3.414634
Eigen Vector:
0.764286        -1.000000        0.650000
STEP-7:
Eigen Value = 3.414286
Eigen Vector:
0.740586        -1.000000        0.673640
```

14. C Program for Fitting y = a + bx

```
#include<stdio.h>
#include<conio.h>
#define S 50
int main()
{
        int n, i;
        float x[S], y[S], sumX=0, sumX2=0, sumY=0, sumXY=0, a, b;
        /* Input */
```

```
    printf("How many data points?\n");
    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]);
    }
    /* Calculating Required Sum */
    for(i=1;i<=n;i++)
    {
            sumX = sumX + x[i];
            sumX2 = sumX2 + x[i]*x[i];
            sumY = sumY + y[i];
            sumXY = sumXY + x[i]*y[i];
    }
    /* Calculating a and b */
    b = (n*sumXY-sumX*sumY)/(n*sumX2-sumX*sumX);
    a = (sumY - b*sumX)/n;
    /* Displaying value of a and b */
    printf("\nValues are: a=%0.2f and b = %0.2f",a,b);
    printf("\nEquation of best fit (y=a+bx)is: y = %0.2f + %0.2fx",a,b);
    getch();
    return(0);
}
```

Output of Example Number 2.14:

```
How many data points?
5
Enter data:
x[1]=1
y[1]=3
x[2]=2
y[2]=4
x[3]=3
y[3]=5
x[4]=4
y[4]=6
x[5]=5
y[5]=8
```

Values are: a=1.60 and b = 1.20
Equation of best fit (y=a+bx)is: y = 1.60 + 1.20x

15. Program for Fitting $y = ax^b$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define S 50
int main()
{
    int n, i;
    float x[S], y[S], sumX=0, sumX2=0, sumY=0, sumXY=0, a, b, A;
    /* Input */
    printf("How many data points?\n ");
    scanf("%d", &n);
    for(i=1;i<=n;i++)
    {
        printf("\nx[%d]=",i);
        scanf("%f", &x[i]);
        printf("y[%d]=",i);
        scanf("%f", &y[i]);
    }
    /* Calculating Required Sum */
    for(i=1;i<=n;i++)
    {
        sumX = sumX + log(x[i]);
        sumX2 = sumX2 + log(x[i])*log(x[i]);
        sumY = sumY + log(y[i]);
        sumXY = sumXY + log(x[i])*log(y[i]);
    }
    /* Calculating A and b */
    b = (n*sumXY-sumX*sumY)/(n*sumX2-sumX*sumX);
    A = (sumY - b*sumX)/n;
    /* Transformation of A to a */
    a = exp(A);
    /* Displaying value of a and b */
    printf("\nValues are: a=%0.2f and b = %0.2f",a,b);
    getch();
    return(0);
}
```

Output of Example 2.15:

```
How many data points?
5
x[1]=1
y[1]=0.5
x[2]=2
y[2]=2
x[3]=3
y[3]=4.5
x[4]=4
y[4]=8
x[5]=5
y[5]=12.5
Values are: a=0.50 and b = 2.00
```

16. Gauss Seidal Method using C Program

```
#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
        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
/* 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("\n Count\tx\ty\tz\n");
    do
    {
        /* Calculation */
        x1 = f1(x0,y0,z0);
        y1 = f2(x1,y0,z0);
        z1 = f3(x1,y1,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("\n Solution: x=%0.3f, y=%0.3f and z = %0.3f\n",x1,y1,z1);
    getch();
    return 0;
}
```

Output of Example Number 4.13:

```
Enter tolerable error:
0.0001
Count       x          y          z
1           0.8500     -1.0275    1.0109
2           1.0025     -0.9998    0.9998
3           1.0000     -1.0000    1.0000
4           1.0000     -1.0000    1.0000
lution: x = 1.000, y = -1.000 and z = 1.000.
```