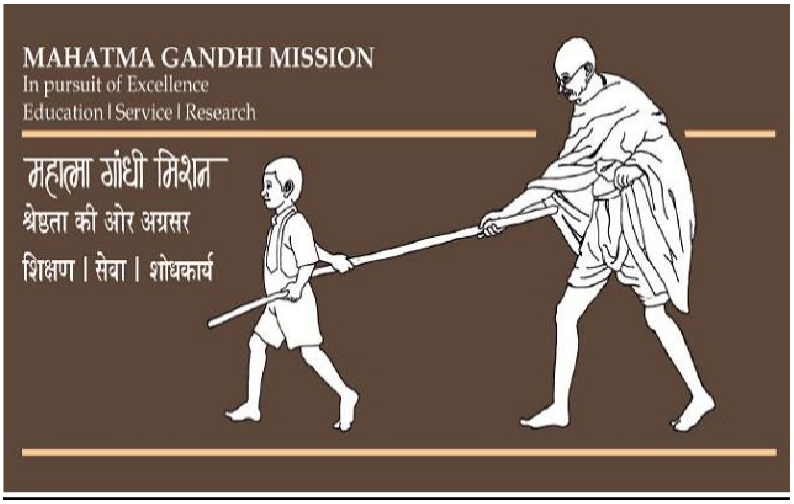
MGM's Jawaharlal Nehru Engineering College



Laboratory Manual

Numerical Methods for Mechanical Engineering

AY:2019-2020 Part 2

CLASS: SYMECH

Prepared By

Prof. A. A. Dudhgaonkar

Prof. K.K.Misal

(Assistant Professor, MCA)

Approved by

Dr.M S Kadam

(Head-Mech. Engg. Dept.)

Mission and vision of the Department

Vision of Mechanical Department

To establish the state of the art learning center in Mechanical Engineering which will impart global competence, enterprising skills, professional attitude and human values in the student.

Mission of Mechanical Department

1. To impart quality technical education to the students.
2. To develop comprehensive competence in the students through various modes of learning.
3. To enable students for higher studies and competitive examinations.
4. To facilitate students and industry professionals for continuous improvement and innovation.

Program Educational Objectives:

1.Use core competence acquired in various areas of Mechanical Engineering to solve techno-managerial issues for creating innovative products that lead to better livelihoods & economy of resources.
2.To establish themselves as effective collaborators and innovators to address technical, managerial and social challenges.
3.To equip students for their professional development through lifelong learning and career advancement along with organizational growth.
4.Serve as a driving force for proactive change in industry, society and nation.

Program Specific Outcomes

Student should have
1) An ability to work professionally in mechanical systems including design, analysis, production, measurement and quality control.
2) An ability to work on diverse disciplinary tasks including manufacturing, materials, thermal, automobile, robotics, mechatronics, engineering software tools, automation and computational fluid dynamics.

LABORATORY MANUAL CONTENTS

This manual is intended for the Second year students of Mechanical Engineering branches in the subject of Numerical Methods for Mechanical Engineering. This typically contains practical/Lab Sessions related Numerical Methods for Mechanical Engineering covering various aspects related the subject to enhanced understanding.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good luck for your Enjoyable Laboratory sessions.

Mechanical Engineering Department

SUBJECT INDEX

List of Experiments

Sr. No	Title of Experiment
01	Program to demonstrates the effect of round off error and significant number
02	Program to find real single root of an equation by Bisection Method
03	Program to find real single root of an equation by Newton -Raphson Method
04	Program to solve linear simultaneous algebraic equations using gauss elimination method
05	Program to solve the integration using multi Trapezoidal Rule.
06	Program to solve the integration using multi Simpson's 1/3.rule
07	Program to solve ordinary differential equations using Euler's Method

Prerequisite Experiments

1) Basics of c language & program for calculating arithmetic operation
2) Program to find out largest of three numbers using nested if else
3) Program to find print number triangle using nested for loop
4) Program to find factorial of number using function
5) Program to calculating addition of 2D matrix
6) Program to display 5's multiplication table using function
7) Program to calculate the sum and average of positive numbers using go-to statement
8) Program to find x^n using while loop

1) Program to demonstrates the effect of round off error and significant number

Introduction :

The round-off error is used because representing every number as a real number isn't possible. So rounding is introduced to adjust for this situation. A round-off error represents the numerical amount between what a figure actually is versus its closest real number value, depending on how the round is applied.

Algorithm:

Step-1. Start of the program.

Step-2. Input the variable trueval, approxval.

Step-3. Calculate absolute error as

abserror=|trueval-approxval|

Step-4. Calculate relative error as

relerror=abserror/trueval

Step-5. Calculate percentage relative error as

percerror=relerror*100

Step-6. PRINT abserror, relerror and percerror

Step-7. STOP

Program

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

void main()
{
float abserror, relerror, percerror, trueval, approxval;

printf("enter true value\n\n");
scanf("%f",&trueval);
printf("enter approx value");
scanf("%f",&approxval);
abserror=fabs(trueval-approxval);
relerror=abserror/trueval;
percerror=relerror*100;
```

```
printf("\n\t absolute error = %f",abserror);
printf("\n\t relative error = %f",relerror);
printf("\n\t percentage error = %f",percerror);
}
```

/* output

```
enter true value
35
enter approx value
34.9997

absolute error = 0.000301
relative error = 0.000009
percentage error = 0.000861
*/
```

2) Program to find real single root of an equation by Bisection Method

INTRODUCTION:

This methods is based on theorem, that if a function f(x) is continuous between a and b, and f(a) * f(b) < 0 i.e. f(a) and f(b) are of opposite signs, then there is exits of at least one root between a and b. Its approximate value will be given by, $x_0=(a+b)/2$. If $f(x_0) = 0$, we conclude that x_0 is a root of equation f(x) = 0. Otherwise the root lies either between x_0 and b or between a and x_0, depending on whether $f(x_0)$ is –ve or +ve. (Considering f(a) is –ve and f(b) is +ve).

Algorithm:

Step-1. Start of the program.

Step-2. Input the variable a,b for the task.

Step-3. Check f(a)*f(b)<0

Step-4. If yes, proceed

Step-5. If no, exit and print error message

Step-6. Repeat 7-11 if condition not satisfied

Step-7. m=(a+b)/2

Step-8. If f(a)*f(m)<0

Step-9. b=m

Step-10. Else

Step-11. a=m

Step-12. Condition:

Step-13. if | (a-b)/a) | < maximum possible error(m-mold) or f(m)==0

Step-14. Print output

Step-15. End of program.

Program

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

#define f(x) ((x*x*x)-18)  //given equation

int main()
{
float a=0,b=0,error=0,m,mold;
int i=0;
printf("Input Interval a & b: \n");
scanf("%f%f",&a,&b);
printf("Ite\ta\t\tb\t\tm\t\tf(m)\t\terror\n");

do{
mold=m;
m=(a+b)/2;
```

```
printf("%2d\t%4.6f\t%4.6f\t%4.6f\t%4.6f\t",i++,a,b,m,f(m));
if(f(m)==0)
{
printf("Root is %4.6f\n",m);
}
else
{

if ((f(a)*f(m))<0)
{  b=m; }
else {a=m;}
}
error=fabs(m-mold);
if(i==1)
{
printf("----\n");
}
else {printf("%4.6f\n",error);}
}while(error>0.00005);

printf("Approximate Root is %4.6f\n\n",m);
return 0;
}
```

Output:

```
Select "C:\Users\dudhg\OneDrive\Desktop\NMME pr\bisection.exe"
Input Interval a & b:
1
3
Ite     a               b               m               f(m)            error
 0      1.000000        3.000000        2.000000        -10.000000      ----
 1      2.000000        3.000000        2.500000        -2.375000       0.500000
 2      2.500000        3.000000        2.750000        2.796875        0.250000
 3      2.500000        2.750000        2.625000        0.087891        0.125000
 4      2.500000        2.625000        2.562500        -1.173584       0.062500
 5      2.562500        2.625000        2.593750        -0.550446       0.031250
 6      2.593750        2.625000        2.609375        -0.233189       0.015625
 7      2.609375        2.625000        2.617188        -0.073128       0.007813
 8      2.617188        2.625000        2.621094        0.007261        0.003906
 9      2.617188        2.621094        2.619141        -0.032963       0.001953
10      2.619141        2.621094        2.620117        -0.012859       0.000977
11      2.620117        2.621094        2.620605        -0.002801       0.000488
12      2.620605        2.621094        2.620850        0.002230        0.000244
13      2.620605        2.620850        2.620728        -0.000285       0.000122
14      2.620728        2.620850        2.620789        0.000972        0.000061
15      2.620728        2.620789        2.620758        0.000343        0.000031
Approximate Root is 2.620758


Process returned 0 (0x0)   execution time : 2.547 s
Press any key to continue.
```

3) Programme to find real single root of an equation by Newton -Raphson Method

Introduction:

In this method the real root of the equation f(x) = 0 can be computed rapidly, when the derivative of f(x) can be easily found and is a simple expression. When an approximate value of a real root of an equation is known, a closer approximation to the root can be obtained by an iterative process, as explained below:

Let x_0 be an approximate value of a root of the equation f(x) = 0.

Let x_1 be the exact root closer to x_0, so that $x_1 = x_0 + h$, where h is small.

Since x_1 is the exact root of f(x) = 0, we have $f(x_1) = 0$, i.e., $f(x_0 + h) = 0$

i.e., $f(x_0) + h f'(x_0) + \frac{h^2}{2!} f''(x_0) + \dots = 0$, by Taylor's theorem

Since h is small, h^2 and higher powers of h may be omitted.

Hence $f(x_0) + h f'(x_0) = 0$, approximately

$\therefore \quad h = -\dfrac{f(x_0)}{f'(x_0)}$ approximately

$\therefore \quad x_1 = x_0 + h = x_0 - \dfrac{f(x_0)}{f'(x_0)}$ approximately.

The value of x_1 thus obtained will be a closer approximation to the actual root of f(x) = 0 than x_0.

Taking x_1 as an approximate value of the root, a still better approximation x_2 can be obtained by using the formula

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

The iterative process is continued until we get the required accuracy, i.e., until $|x_{n+1} - x_n|$ is less than a prescribed small value.

The iterative formula

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

is called the Newton-Raphson formula.

ALORITHM

Step-1. Start of the program.

Step-2. input the variables x0 for the task.

Step-3.Define f(x0) and fd(x0)

Step-4. Repeat the loop for error(x1-x0)>0.00005

Step-5. Calculate

x1=x0-(f(x0)/fd(x0));

Step-6. Print i,x1,error

Step-7.Assign x0=x1;

Step-8.end of loop

Step-9. Print final value of x0

Step-11.stop

Program

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define f(x) ((x*x*x)-18) // given equation
#define fd(x) (3*x*x)  //derivative of given equation

int main()
{
```

```
float x0,x1,error;
int i=0;
printf("Input the initial approximation : ");
scanf("%f",&x0);

printf("Ite\tX1\t\tError\t\n");
do{

x1=x0-(f(x0)/fd(x0));
if(f(x1)==0)
{
break;
}
error=fabs(x1-x0);  //fabs:absolute function for error value
printf("%2d\t%4.6f\t%4.6f\t",++i,x1,error);
printf(" \n");
x0=x1;
}while(error>0.00005);
printf("Since Last Two iteration has same value of x1  in above table \n.Hence Root is %4.6f\n\n",x0);
return 0;
}
```

/*output

```
Input the initial approximation : 2
Ite     X1            Error
1       2.833333        0.833333
2       2.636294        0.197040
3       2.620833        0.015461
4       2.620741        0.000092
5       2.620741        0.000000
Since Last Two iteration has same value of x1  in above table
.Hence Root is 2.620741
```

*/

4) Program to solve linear simultaneous algebraic equations using gauss elimination method

Introduction:

In linear algebra, Gauss Elimination Method is a procedure for solving systems of linear equation. It is also known as Row Reduction Technique. In this method, the problem of systems of linear equation having n unknown variables, matrix having rows n and columns n+1 is formed. This matrix is also known as Augmented Matrix. After forming n x n+1 matrix, matrix is transformed to upper trainagular matrix by row operations. Finally result is obtained by Back Substitution.

Algorithm

1. Start

2. Read Number of Unknowns: n

3. Read Augmented Matrix (A) of n by n+1 Size

4. Transform Augmented Matrix (A)
to Upper Trainagular Matrix by Row Operations.

5. Obtain Solution by Back Substitution.

6. Display Result.

7. Stop

Program

```
#include<stdio.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("\nSolution:\n");
for(i=1;i<=n;i++)
{
printf("x[%d] = %0.3f\n",i, x[i]);
}
getch();
return(0);
}

/* output

Enter number of unknowns: 3
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

Solution:

x[1] = 1.000

x[2] = 3.000

x[3] = 5.000

*/

5) Program to solve the integration using multi Trapezoidal Rule.

Introduction :

In numerical analysis, Trapezoidal method is a technique for evaluating definite integral. This method is also known as Trapezoidal rule or Trapezium rule.

A method for approximating a definite integral $\int_a^b f(x)dx$ using linear approximations of f. The trapezoids are drawn as shown below. The bases are verticallines.

Let y=f(x) be a function defined on [a,b], which is divided into n subintervals each of width h ,so that (b-a/n)=h

Let the values of f(x) for (n+1) equidistant arguments $x_0 = a$, $x_1 = x_0 + h$, $x_2 = x_0 + 2h$,....., $x_n = x_0 + nh = b$ be $y_0, y_1, y_2, \ldots y_n$ respectively.

$$\text{Then } \int_a^b f(x)dx = \int_{x_0}^{x_0+nh} y\,dx$$

$$= h\left[\frac{1}{2}(y_0 + y_n) + (y_1 + y_2 + \ldots\ldots + y_{n-1})\right]$$

$$= \frac{h}{2}\left[(y_0 + y_n) + 2(y_1 + y_2 + \ldots\ldots + y_{n-1})\right]$$

This rule is known as **Trapezoidal rule** .

The geometrical significance of this rule is that the curve y = f(x) is replaced by *n* straight lines joining the points (x_0, y_0) and (x_1, y_1); (x_1, y_1) and (x_2, y_2);.....; (x_{n-1}, y_{n-1}) and (x_n, y_n). The area bounded by the curve y = f(x). The ordinate $x = x_0$ and $x = x_n$ and the *x*-axis, is then approximately equivalent to the sum of the areas of the n trapeziums obtained.

Algorithm

1. Start

2. Define function f(x)

3. Read lower limit of integration, upper limit of
   integration and number of sub interval

4. Calcultae: step size = (upper limit - lower limit)/number of sub interval

5. Set: i = 1

6. If i > number of sub interval then goto

7. Calculate: sum= sum + f(lower limit + i * h)

8. Increment i by 1 i.e. i = i+1 and go to step 6

9. Calculate: result = f(lower limit) + f(upper limit)

+2*sum)* step size/2

10. Display Integration value as required answer

11. Stop

Program

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

/* Define function here */
#define f(x) 1/(1+pow(x,2))
//#define f(x)  sqrt(x)

int main()
{
 float lower, upper, sum=0.0, stepSize,result;
 int i, subInterval;

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

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

 /* Finding Integration Value */
  for(i=1; i<= subInterval-1; i++)
```

```
 {
   sum = sum + f(lower + i*stepSize);
 }
 result=(stepSize/2)*(f(lower) + f(upper)+2*sum);
 printf("\nRequired value of integration is: %.3f", result);

 return 0;
}
```

/*output

Enter lower limit (a) of integration: 0
Enter upper limit(b) of integration: 1
Enter number of sub intervals (n): 6

Required value of integration is: 0.784

*/

6)Program to solve the integration using multi Simpson's 1/3.rule

Introduction

Let y = f(x) be a function defined on [a, b] which is divided into n (an even number) equal parts each of width h, so that b − a = nh.

Suppose the function y = f(x) attains values $y_0, y_1, y_2, \ldots y_n$ at n+1 equidistant points $x_0 = a, x_1 = x_0 + h, x_2 = x_0 + 2h, \ldots, x_n = x_0 + nh = b$ respectively. Then

$$\int_a^b f(x)dx = \int_{x_0}^{x_0+nh} ydx = \frac{h}{3}[(y_0 + y_n) + 4(y_1 + y_3 + y_5 + \ldots + y_{n-1}) + 2(y_2 + y_4 + \ldots + y_{n-2})]$$

= (one-third of the distance between two consecutive ordinates) [(sum of the extreme ordinates) + 4(sum of odd ordinates) + 2(sum of even ordinates)]

This formula is known as **Simpson's one-third rule**. Its geometric significance is that we replace the graph of the given function by n/2 arcs of second degree polynomials, or parabolas with vertical axes. It is to note here that the interval [a, b] is divided into an even number of subinterval of equal width.
Simpson's rule yield more accurate results than the trapezoidal rule. Small size of interval gives more accuracy.

Algorithm

1. Start

2. Define function f(x)

3. Read lower limit (a) of integration, upper limit (b) of
   integration and number of sub interval (n)

4. Calcultae: step size(h) = (upper limit - lower limit)/number of sub interval

5. Set: i = 1

6. If i > number of sub interval then goto

7. If i mod 2 =0 then
sum = sum + f(lower limit + i * h)
   Otherwise

Sum1 = sum1 + f(lower limit + i * h)
End If

8. Increment i by 1 i.e. i = i+1 and go to step 6

9. Calculate: result=(step size/3)*(f(lower limit) + f(upper limit)+2*sum+4*sum1)

10. Display result as required answer

11. Stop

Program

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

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

int main()
{
 float lower, upper, sum=0.0,sum1=0.0, stepSize, result;
 int i, subInterval;
 /* Input */
 printf("Enter lower limit (a) of integration: ");
 scanf("%f", &lower);
 printf("Enter upper limit (b) of integration: ");
 scanf("%f", &upper);
 printf("Enter number of sub intervals (n): ");
 scanf("%d", &subInterval);

 /* Calculation */
```

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

 /* Finding Integration Value */
 for(i=1; i<= subInterval-1; i++)
 {
  if(i%2==0)
  {
   sum = sum + f(lower + i*stepSize);
  }
  else
  {
  sum1 = sum1 + f(lower + i*stepSize);
  }
 }

 result=(stepSize/3)*(f(lower) + f(upper)+4*sum1+2*sum);
 printf("\nRequired value of integration is: %.3f", result);

 return 0;
}
```

/* out-put

```
Enter lower limit (a) of integration: 0
Enter upper limit (b) of integration: 1
Enter number of sub intervals (n): 6

Required value of integration is: 0.693

*/
```

7)Program to solve ordinary differential equations using Euler's Method

Introduction

In summary, Euler's method for approximating the solution to the initial-value problem

Y`= f (x, y), y(x0) = y0

at the points xn+1 = x0 + nh (n = 0, 1, . . .) is

yn+1 = yn + hf (xn, yn), n = 0, 1, ...

Algorithm

1.Start

2. Define function f(x,y)

3. Read values of initial condition(x0 and y0),
   number of steps (n) and calculation point (xn)

4. Calculate step size (h) = (xn - x0)/b

5. Set i=0

6. Loop

   yn = y0 + h * f(x0 + i*h, y0)

   y0 = yn

   i = i + 1

  While i < n

7. Display yn as result

8. Stop

Program

```
#include<stdio.h>

#define f(x,y) (y-2*x/y)  //(-2)*x*y*y //y-x //x+y

int main()
{
 float x0, y0, xn, h, yn, slope;
 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 step size (h)=  ");
 scanf("%f", &h);
 /* Calculating steps */
 n = (xn-x0)/h;
 /* Euler's Method */
 printf("\nIetr.\tx0\ty0\tyn\n");
 printf("------------------------------\n");
 for(i=1; i<=n; i++)
 {
  slope = f(x0, y0);
  yn = y0 + h * slope;
  y0 = yn;
  x0 = x0+h;
  printf("%d\t%.4f\t%.4f\t%.4f\n",i,x0,y0,yn);
 }
```

```
 /* Displaying result */
 printf("\nValue of y at x = %0.2f is %0.3f\n",xn, yn);

 return 0;
}
```

/* output

```
Enter Initial Condition
x0 = 0
y0 = 1
Enter calculation point xn = 0.8
Enter step size (h)= 0.1

Ietr.   x0      y0      yn
------------------------------
1       0.1000  1.1000  1.1000
2       0.2000  1.1918  1.1918
3       0.3000  1.2774  1.2774
4       0.4000  1.3582  1.3582
5       0.5000  1.4351  1.4351
6       0.6000  1.5090  1.5090
7       0.7000  1.5803  1.5803
8       0.8000  1.6498  1.6498

Value of y at x = 0.80 is 1.650

*/
```

Assignment 6-Extra

Program to calculate the sum and average of positive numbers using go-to statement

```
int main()
{

   const int maxInput = 5;
   int i;
   double number, average, sum=0.0;

   for(i=1; i<=maxInput; ++i)
   {
      printf("%d. Enter a number: ", i);
      scanf("%lf",&number);

      if(number < 0.0)
         goto jump;

      sum += number;
   }

   jump:
   average=sum/(i-1);
   printf("Sum = %.2f\n", sum);
   printf("Average = %.2f", average);

   return 0;
}
```

/*OUTPUT

1. Enter a number: 4
2. Enter a number: 7
3. Enter a number: 9
4. Enter a number: 2
5. Enter a number: 8

Sum = 30.00

Average = 6.00

*/

Assignment 7-Extra

Program to find x^n using while loop

```
#include <stdio.h>
int main() {
   int base, exp,result = 1;
   printf("Enter a base number: ");
   scanf("%d", &base);
   printf("Enter an exponent: ");
   scanf("%d", &exp);

   while (exp != 0) {
      result *= base;
      --exp;
   }
   printf("Answer = %d", result);
   return 0;
}
```

/*output

Enter a base number: 2

Enter an exponent: 3

Answer = 8

*/