

A Novel Approach to Solve Numerical Methods Using Matlab Programming

Diya Mehra, Priyanka Sond, Reena Agnihotri

GNA University, Phagwara, Punjab

Corresponding author: Reena Agnihotri, Email: reena.agnihotri@gnauniversity.edu.in

Numerical techniques is a branch of mathematics that explores problem-solving methods that account for approximation errors. This paper defines the algorithms for some of the most popular iterative procedures for solving polynomial problems, including the Bisection Method, Newton- Raphson Method, Secant Method, and Gauss-Jacobi Method and many more. The purpose of this research is to use MATLAB software to solve numerical method problems. The primary goal of this study is to describe and resolve various numerical methodologies using a single platform. We also provide a brief overview of different methodologies and their underlying algorithms, which may make them easier to comprehend.

Keywords: Bisection, MATLAB, Gauss-Seidel, Secant, Simpson's

1. Introduction

MATLAB is a high-performance language for technical computing. It integrates computation, visualization, and programming environment [1]. MATLAB is a very powerful software package that has many built-in tools for solving problems and for graphical illustrations [2]. MATLAB stands for MATrix LABORatory. The heart of MATLAB is the MATLAB language, a matrix-based language allowing the most natural expression of computational mathematics. The MATLAB software package offers a setting in which we can study programming and investigate the architecture of numerical algorithms [4].

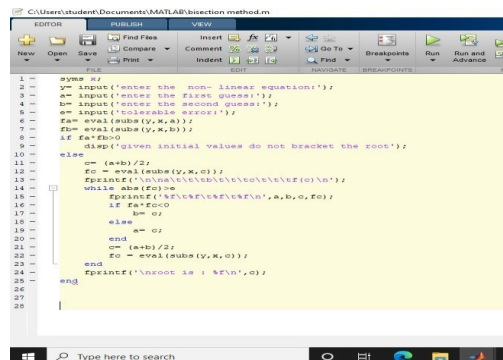
2. Bisection Method

For the solution of polynomial equations, use the bisection method. The interval containing the equation's root is divided and separated by it. The guiding principle for this method is the intermediate theorem for continuous functions. The proper answer is eventually attained by gradually reducing the distance between the positive and negative intervals. [5] In this procedure, the gap is closed using the average of the positive and negative periods. It is a simple method, but it is time-consuming. Other name of bisection method, the Interval Halving method, the Root-finding method, the Binary search method, and the Bolzano method.

2.1 Bisection Method Algorithm

1. Start
2. Create a function $f(x)$
3. Choose initial guesses x_0 and x_1 such that $f(x_0)f(x_1) < 0$
4. Choose the acceptable error e .
5. Calculate new approximated root as $c = (a + b)/2$
6. Calculate $f(a)f(b)$
 - if $f(x_0)f(x_2) < 0$ then $x_1 = x_2$
 - if $f(x_0)f(x_2) > 0$ then $x_0 = x_2$
 - if $f(x_0)f(x_2) = 0$ then goto (8)
7. if $|f(x_2)| > e$ then repeat (5) otherwise goto (8)
8. Display x_2 as root of the equation .
9. Stop

A Example



```
1 - syms x;  
2 - % Input('enter the non-linear equation');  
3 - A= input('enter the first guess');  
4 - B= input('enter the second guess');  
5 - e= input('tolerable error');  
6 - Fx= eval(subs(y,M,A));  
7 - Fx= eval(subs(y,M,B));  
8 - if Fx*Fx>0  
9 - disp('Given initial values do not bracket the root');  
10 - else  
11 - c= (A+B)/2;  
12 - Fc = eval(subs(y,M,c));  
13 - fprintf('\nRoot is %f\n',c);  
14 - while abs(Fc)>e  
15 - if Fx*Fc<0  
16 - A= c;  
17 - Fx= Fc;  
18 - else  
19 - B= c;  
20 - Fc = eval(subs(y,M,c));  
21 - c= (A+B)/2;  
22 - Fc = eval(subs(y,M,c));  
23 - end  
24 - fprintf('\nroot is : %f\n',c);  
25 - end  
26 -  
27 -  
28 -
```

Fig. 1 Bisection Method Example

B. Output

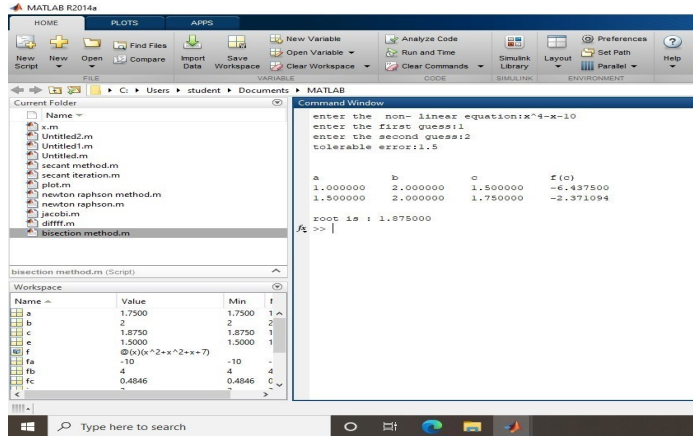


Fig. 2: Bisection Method Output

3. Newton-Raphson Method

Newton's method or Newton–Raphson method is named after Isaac Newton and Joseph Raphson. It is a method for quickly estimating the root of a real-valued function, such as $f(x) = 0$ or $f'(x)=0$. [6]. It is a powerful and fastest technique for solving equations numerically. In differential calculus, it is based on the simple idea of linear approximation. [8].

A. Newton-Raphson Method Algorithm

1. Start
2. create function as $f(x)$
3. define derivative of $f(x)$ as $g(x)$
4. Enter starting guess (x_0), tolerable error (e) and maximum iteration (N)
5. Initialize iteration counter $i = 1$
6. If $g(x_0) = 0$ then print "Mathematical Error" and exit otherwise proceed (7)
7. Calculate: $x_1 = x_0 - f(x_0) / g(x_0)$
8. Increment iteration counter $i = i + 1$
9. If $i >= N$ then print "Non Convergent" and proceed (12) otherwise proceed (10)
10. If $|f(x_1)| > e$ then set $x_0 = x_1$ and go to step (6) otherwise to step (11)
11. Print root as x_1
12. Stop

A. Example

```

1 % syms x;
2 y= input('enter the non-linear equations:');
3 a= input('enter the initial guess:');
4 e= input('tolerable error:');
5 N= input('enter maximum number of steps:');
6 step =1;
7 g= diff(y,x);
8 Ea = eval(subs(y,x,a));
9 while abs(Ea)>e
10     fa= eval(subs(y,x,a));
11     ga= eval(subs(g,x,a));
12     if ga==0
13         disp('division by zero. ');
14         break;
15     end
16     b= a - fa/ga;
17     fprintf('step=%d\t a=%f\t f(a)= %f\n',step,a,fa);
18     a= b;
19     if step >N
20         disp('Not convergent');
21         break;
22     end
23     step = step + 1;
24 end
25 fprintf('root is %f\n',a);
26
27

```

Fig. 3 Newton-Raphson Method Example

B. Output

The screenshot shows the MATLAB R2014a interface. The Command Window displays the following output:

```

Enter non-linear equations: x^3-x-1
Enter initial guess: 1.5
Tolerable error: .1
Enter maximum number of steps: 4
step=1 a=1.500000 f(a)=0.875000
step=2 a=1.347826 f(a)=0.100682
step=3 a=1.325200 f(a)=0.002058
Root is 1.324718
fx >> |

```

The Workspace window shows the following variables:

| Name | Value | Min | Max |
|------|-----------------|--------|-----|
| a | 1.3247 | 1.3247 | 1 |
| b | 1.3247 | 1.3247 | 1 |
| c | 1.8750 | 1.8750 | 1 |
| e | 0.1000 | 0.1000 | 0 |
| f | @(x)x^2+x^2+x+7 | 0.0021 | 0 |
| fa | 4 | 4 | 4 |
| fb | 0.0021 | 0.0021 | 0 |
| fc | 0.4846 | 0.4846 | 0 |

Fig. 4 Newton-Raphson Method Output

4. Secant Method

The Secant method is a root-finding methodology for numerical analysis that makes use of a series of secant line roots to more accurately approximation the root of a function f . Secant Method is open method. It starts with two initial guesses for finding the real root of the non- linear equations. In this method if x_0 and x_1 are initial guesses then next approximated root x_2 is derived by following formula: $x_2 = x_1 - (x_1 - x_0) * f(x_1) / (f(x_1) - f(x_0))$ This is an algorithm for Secant method which involves the repetition of the above process i.e., we use x_1 and x_2 to find x_3 and so on until we find the root within desired accuracy.[10]

A. Secant Method Algorithm

1. Start
2. create function as $f(x)$
3. Enter initial guesses (x_0 and x_1), tolerable error (tol) and maximum iteration (n)
4. Initialize iteration counter $i = 1$
5. If $f(x_0) = f(x_1)$ then prints "Mathematical Error" and proceed (11) otherwise goto (6)
6. Calculate: $x_2 = x_1 - (x_1 - x_0) * f(x_1) / (f(x_1) - f(x_0))$
7. Increment iteration counter $i = i + 1$
8. If $i \geq N$ then print "Not Converge" and proceed(11) otherwise proceed (9)
9. If $|f(x_2)| > e$ then set $x_0 = x_1, x_1 = x_2$ and proceed with step (5) otherwise goto (10)
10. Print root as x_2
11. Stop

B. Example

```

1 - f = @(x) (cos(x));
2 - p0 = input('Enter 1st approximation, p0: ');
3 - p1 = input('Enter 2nd approximation, p1: ');
4 - n = input('Enter no. of iterations, n1: ');
5 - tol = input('Enter tolerance, tol: ');
6 - i = 2;
7 - f0 = f(p0);
8 - f1 = f(p1);
9 - while i <= n
10 -    p = p1-f1*(p1-p0)/(f1-f0);
11 -    fp = f(p);
12 -    if abs(p-p1) < tol
13 -        fprintf('\nApproximate solution p = %11.8f\n\n',p);
14 -        break;
15 -    else
16 -        i = i+1;
17 -        p0 = p1;
18 -        f0 = f1;
19 -        p1 = p;
20 -        f1 = fp;
21 -    end
22 - end
23 -
24 -
25 -

```

Fig. 5 Secant Method Example

C. Output

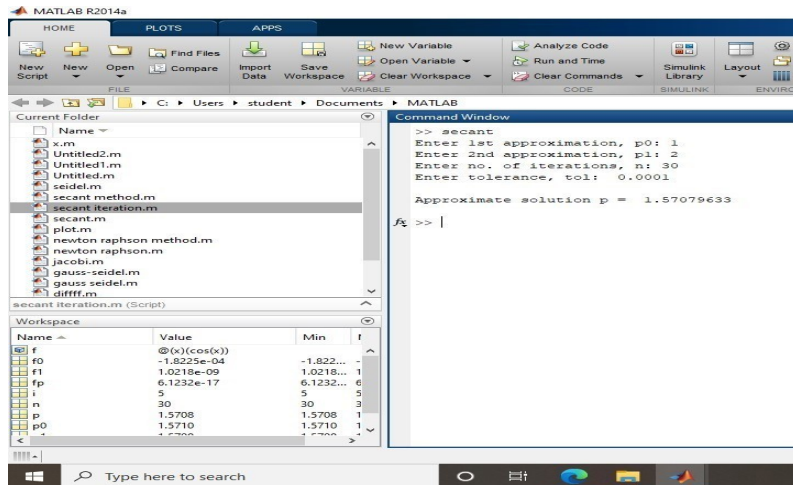


Fig. 6 Secant Method Output

5. Gauss-Jacobi Method

The Jacobi method is an iterative algorithm in numerical linear algebra for finding the solutions of a strictly diagonally dominant system of linear equations. A rough value is entered after solving for each diagonal element. The process is then repeated until convergence is reached. [11].

A. Gauss-Jacobi Method Algorithm

1. Start
2. Arrange given system of linear equations in diagonally dominant form
3. Read tolerable error (tol)
4. Convert the first equation in terms of first variable, second equation in terms of second variable and so on.
5. Set initial guesses for x_0, y_0, z_0 as zero and so on
6. Substitute value of $x_0, y_0, z_0 \dots$ from step 5 in equation obtained in step 4 to calculate new values x_1, y_1, z_1 and so on
7. If $|x_0 - x_1| > \text{tol}$ and $|y_0 - y_1| > \text{tol}$ and $|z_0 - z_1| > \text{tol}$ and so on then proceed step 9
8. Set $x_0 = x_1, y_0 = y_1, z_0 = z_1$ and so on and proceed step 6
9. Print value of x_1, y_1, z_1 and so on
10. Stop

A. Example

```

5  tol = input('enter the tolerance,tol:');
6  m= input('enter the maximum number of iterations,m:');
7  A= [4 2 3 8 ; 3 -5 2 -14; -2 3 8 27];
8  x1= [0 0 0 ];
9  k = 1;
10 while k<=m
11     err = 0;
12     for i = 1:n
13         a = 0;
14         for j = 1:n
15             a = a+h(i,j)* x1(j);
16         end
17         a = (a+A(i,n+1))/A(i,i);
18         if abs(a)>err
19             err = abs(a);
20         end
21         x2(i)=x1(i)+a;
22     end
23     if err <=tol
24         break;
25     else
26         k = k+1;
27         for i=1:n
28             x1(i)=x2(i);
29         end
30     end
31 end
32 fprintf('solution vector after %d iterations is :\n',k-1);
33 for i= 1:n
34     fprintf('%1.8f\n',x2(i));
35 end
    
```

Fig. 7 Gauss Jacobi Method Example

B. Output

Command Window

```

>> jacobi
enter the number of equations,n:3
enter the tolerance,tol:0.001
enter the maximum number of iterations,m:4
solution vector after 4 iterations is :
-0.08007812
1.47812500
1.63734375
    
```

Workspace

| Name | Value | Min | Max |
|------|------------|--------|-----|
| A | 3x3 double | -14 | 27 |
| err | 2.7437 | 2.7437 | 2 |
| i | 3 | 3 | 3 |
| k | 3 | 3 | 3 |
| m | 5 | 3 | 5 |
| n | 4 | 4 | 4 |
| x1 | 3 | 3 | 3 |
| x2 | 0.7514 | 0.7514 | 0 |
| x | 1.6373 | 1.6373 | 0 |

Fig. 8 Gauss Jacobi Method Output

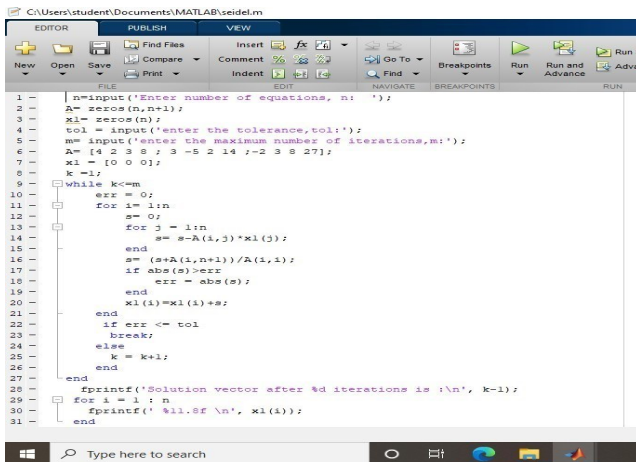
6. Gauss-Seidel Method

The Liebman method and the sequential displacement method are other names for the Gauss-Seidel approach. This approach bears the names of the mathematicians Philipp L. Seidel (1821–1896) and Carl Friedrich Gauss (1777–1855). In numerical techniques, an iterative approach is used to solve a set of linear equations. Round-off error can be managed by the user using the Gauss-Seidel Method. Round-off errors can have an impact on the number of methods for elimination. This method is modification of the Jacobi's iteration method. [12].

A. Gauss-Seidel Method Algorithm

1. Start
2. Arrange the system of linear equations in diagonally dominant form
3. Read tolerable error (tol)
4. Convert the first equation in terms of first variable, second equation in terms of second variable and so on.
5. Set initial guesses for y_0, z_0 and so on as zero.
6. Substitute value of $y_0, z_0 \dots$ from step 5 in first equation obtained from step 4 to calculate new value of x_1 . Use x_1, z_0, u_0, \dots in second equation obtained from step 4 to calculate new value of y_1 . Similarly, use $x_1, y_1, u_0 \dots$ to find new x_2 and so on.
7. If $|x_0 - x_1| > \text{tol}$ and $|y_0 - y_1| > \text{tol}$ and $|z_0 - z_1| > \text{tol}$ and so on then proceed step 9
8. Set $x_0=x_1, y_0=y_1, z_0=z_1$ and so on and proceed step 6
9. Print value of x_1, y_1, z_1 and so on
10. Stop

A. Example



```
1 | n=input('Enter number of equations, n: ');
2 | A= zeros(n,n+1);
3 | x1= zeros(n);
4 | tol = input('enter the tolerance,tol:');
5 | m= input('enter the maximum number of iterations,m:');
6 | A= [4 2 3 8 ; 3 -5 2 14 ;-2 3 8 27];
7 | x1 = [0 0 0];
8 | k =1;
9 | while k<=m
10 |    err = 0;
11 |    for i= 1:n
12 |        s= 0;
13 |        for j = 1:n
14 |            s= s-A(i,j)*x1(j);
15 |        end
16 |        s= (s+A(i,n+1))/A(i,i);
17 |        if abs(s)>err
18 |            err = abs(s);
19 |        end
20 |        x1(i)=x1(i)+s;
21 |    end
22 |    if err <= tol
23 |        break;
24 |    else
25 |        k = k+1;
26 |    end
27 | end
28 | fprintf('Solution vector after %d iterations is :\n', k-1);
29 | for i = 1 : n
30 |     fprintf(' %11.8f \n', x1(i));
31 | end
```

Fig. 9. Gauss Seidel Method Example

B. Output

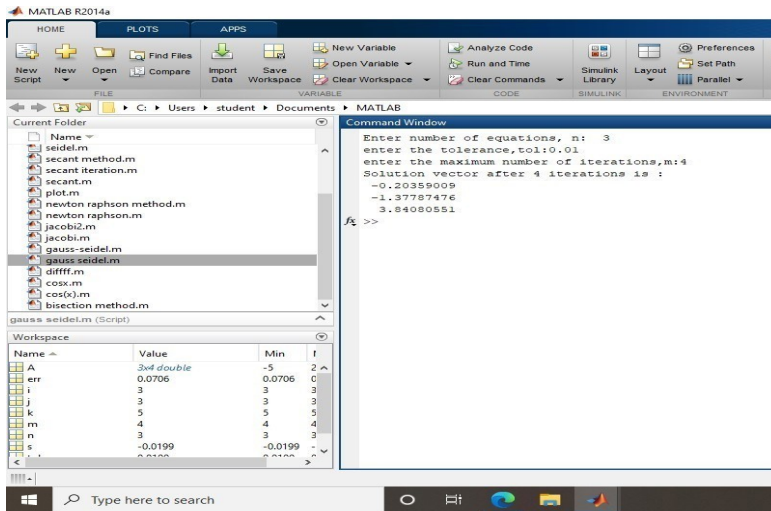


Fig .10 Gauss Seidel Method Output

7. Simpson’s 1/3 Rule Method

A numerical approach to find the integral $\int_a^b y \, dx$ within the limiting limits a and b in numerical integration is the Simpson's 1/3 rule. The integral of the bounded parabola is computed using Simpson's 1/3 approach, and the approximate integral is then shown. A polynomial of degree two, $p(x)$, or a parabola between the two boundaries a and b , is used to approximate $f(x)$. [14].

1. Begin by defining function $f(x)$
2. Read the number of sub intervals, the top limit of integration, and the lower limit of integration.
3. Calculate: step size = (upper limit - lower limit)/number of sub interval
4. Set: integration value = $f(\text{lower limit}) + f(\text{upper limit})$
5. Set: $i = 1$
6. If $i >$ number of sub interval then goto
7. Calculate: $k = \text{lower limit} + i * h$
 If $i \bmod 2 = 0$ then Integration value = Integration Value + $2 * f(k)$ Otherwise
 Integration Value = Integration Value + $4 * f(k)$ End If
8. Increment i by 1 i.e. $i = i + 1$ and go to step 7
9. Compute the Integration value = Integration value * step size/3
10. Show Integration value as the required response
11. Stop

A. Example

```
C:\Users\student\Documents\simpsons rule.m
EDITOR      PUBLISH      VIEW
1 % MATLAB code for syms function that creates a variable
2 % dynamically and automatically assigns
3 % to a MATLAB variable with the same name
4 syms X
5 % Lower Limit
6 a = 4;
7 % Upper Limit
8 b = 5.2;
9 % Number of Segments
10 n = 6;
11 % Declares the function
12 f1 = log(X);
13 % inline creates a function of string containing in f1
14 f = inline(f1);
15 % n is the segment size
16 h = (b - a)/n;
17 % X stores the summation of first and last segment
18 X = f(a)+f(b);
19 % variables Odd and Even to store
20 % summation of odd and even
21 % terms respectively
22 Odd = 0;
23 Even = 0;
24 for i = 1:2:n-1
25     xi=a+(i*h);
26     Odd=Odd+f(xi);
27 end
28 for i = 2:2:n-2
29     xi=a+(i*h);
30     Even=Even+f(xi);
31 end
32 % Formula to calculate numerical integration
33 % using Simpsons 1/3 Rule
34 I = (h/3) * (X+4*Odd+2*Even);
35 disp('The approximation of above integral is: ');
36 disp(I);
```

Fig. 11 Simpsons Rule Example

B. Output

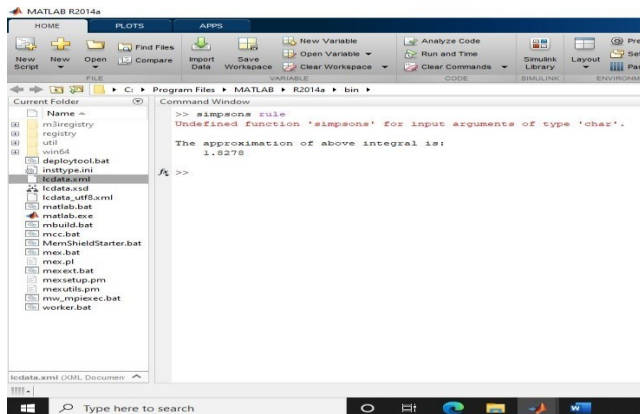


Fig.12 Simpsons Rule Output

8. Conclusion

In this paper, we have discussed variety of numerical methods for finding the solution of ordinary equations with initial conditions by MATLAB Programming. As a result, we can conclude that these numerical methods are useful for obtaining approximate solutions. This work could be expanded in several ways. We have only introduced the fundamental algorithms and MATLAB code in this paper, and we can expand on this by introducing the convergence of the other numerical methods and comparison between different methods by plotting graphs.

References

- [1] David Houcque, The MathWorks Inc. MATLAB 7.0 (R14SP2). The MathWorks Inc., 2005.
- [2] Attaway, Stormy, "Matlab: A Practical Introduction to Programming and Problem Solving" Canada : Elsevier, Inc., 2009.
- [3] Gilat, Amos, "MATLAB An Introduction with Applications", Columbia, Ohio : John Wiley & Sons, Inc., 2011.
- [4] Alka Benny, Maria Goldwin, "Numerical Methods Using Matlab", Volume 5, Issue 5, JETIR May 2018.
- [5] Ali Wahid Nwry , Hemin Muheddin Kareem , Ribwar Bakhtyar Ibrahim, et al, "Comparison Between Bisection, Newton and Secant Methods for determining the root of the Non-Linear equation using MATLAB", Turkish Journal of Computer and Mathematics Education Vol.12, pp 1115 – 1122, 2021.
- [6] Adi Ben-Israel, "A Newton-Raphson Method For The Solution Of Systems Of Equations", Journal Of Mathematical Analysis And Applications 15, pp 243-252, 1966.
- [7] Saba Akram, Qurat ul Ann, "Newton Raphson Method", International Journal of Scientific & Engineering Research, Volume 6, July-2015.
- [8] VISHAL V. MEHTRE , SHUBHAM SINGH, "Obtaining Roots of Non-Linear Equation Using Newton Raphson Method", IRE Journals ,Volume 3, DEC 2019
- [9] Osama. Y. Ababneh, "New Numerical Methods For Solving Differential Equations", Journal of Advances in Mathematics , vol 16, 2019
- [10] Isaac Azure, Golbert Aloliga, Louis Doabil, "Comparative Study of Numerical Methods for Solving Non linear Equations Using Manual Computation", Mathematics Letters, Vol. 5, pp. 41-46, 2019
- [11] Yash Bhojwani, Rishab Singh, "Parallelization of Definite Integration", International Research Journal of Engineering and Technology (IRJET), Volume: 06 , Nov 2019
- [12] Dr.S.Karunanithi, N.Gajalakshmi, M.Malarvizhi, M.Saileshwari, "A Study on Comparison of Jacobi, GaussSeidel and Sor Methods for the Solution in System of Linear Equations", International Journal of Mathematics Trends and Technology (IJMTT) – Volume 56, April 2018
- [13] Riyasdeen S1 , Abbas S2 , Lenin T3. "Illustration of Gauss – Seidel Method Using Matlab", International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, pp. 2234-2237, 2019
- [14] Md. Jashim Uddin, Mir Md. Moheuddin and Md. Kowsher , " A New Study Of Trapezoidal, Simpson's 1/3 And Simpson's 3/8 Rules Of Numerical Integral Problems ", an international journal(mathSJ), Vol.6, December 2019.
- [15] Sharaban Thohura and Azad Rahman, "Comparison Of Numerical Methods For Solving Initial Value Problems For Stiff Differential Equations", Ganit J. Bangladesh Math. Soc. 30, pp 122-132, 2010.