KATHMANDU UNIVERSITY

SCHOOL OF ENGINEERING

DEPARTMENT OF ELECTRICAL & ELECTRONICS ENGINEERING

PROJECT REPORT

Implementation of Finite difference method in MATLAB

BY:

MISSON KHAND (22015)

ROMANCH NYAUPANE (22048)

SUBMITTED TO:

DR. SARASWATI ACHARYA

ASST. PROF. DEPARTMENT OF MATHEMATICS,

MAY 26, 2023

ABSTRACT

This project focuses on solving a boundary value problem for a second-order linear ordinary differential equation algorithm in MATLAB. The algorithm involves converting the boundary value problem into a tridiagonal matrix equation and solving it to obtain the solution. The user is prompted to input various parameters, including the number of intervals, initial and boundary conditions, and coefficients of the differential equation. The code then computes the tridiagonal matrix coefficients and right-hand side values, adjusts the boundary conditions, and solves the tridiagonal system. The resulting solution is plotted, and each point is annotate

TABLE OF CONTENTS

1. INTRODUCTION

A boundary value problem is a mathematical problem that involves finding a solution to a differential equation subject to specified conditions at the boundaries of the domain. It is concerned with determining the values of the unknown function and its derivatives at the boundaries.

The finite difference method is a numerical technique used to solve boundary value problems by approximating derivatives using finite differences. It discretizes the domain into a set of grid points and replaces derivatives with finite difference approximations. The method involves converting the differential equation into a system of algebraic equations by evaluating the finite differences at each grid point. These equations are then solved to obtain an approximation of the unknown function at the grid points. The accuracy of the finite difference method depends on the grid spacing, with smaller spacings yielding more accurate solutions.

1.1 Derivation of finite difference method

Consider a differential equation with given boundary condition

$$y''(x) + p(x)y'(x) + q(x)y(x) = r(x), \quad y(x_0) = a,\ y(x_n) = b$$

Let $[x_0, x_n]$ be divided into n equal subintervals of width h. At $x = x_i$, the central difference approximations for $y'(x)$ and $y''(x)$ are

$$y_i'(x_i) = y_i' = \frac{y(x_i+h)-y(x_i-h)}{2h} = \frac{y(x_{i+1})-y(x_{i-1})}{2h} = \frac{y_{i+1}-y_{i-1})}{2h}$$

$$y''(x_i) = y_i'' = \frac{y_{i-1}-2y_i + y_{i+1})}{h^2}$$

Now, at $x = x_i$, given differential equation is $y_i''+ p_iy_i'+ q_iy_i = r_i$ Substituting y_i'' and y_i', we get

$$\frac{y_{i-1}-2y_i + y_{i+1})}{h^2} + p_i \frac{(y_{i+1}-y_{i-1})}{2h} + q_i y_i = r_i \qquad i=1, 2, 3, \cdots, n - 1$$

Multiply both sides by $2h^2$ we get

$$(y_{i-1} - 2y_i + y_{i+1}) + hp_i (y_{i+1} - y_{i-1}) + 2h^2 q_i y_i = 2h^2 r_i$$

Collecting the coefficients of y_{i-1}, y_i and y_{i+1}, we get

$$y_{i-1}(2-hp_i)+y_i(-4+2h^2q_i)+y_{i+1}(2+hp_i)= 2h^2r_i$$

Substituting $i = 1, 2, 3, \cdots, n - 1$ successively, we get $n - 1$ system of equation as

$$i = 1 : \ y_0(2 - hp_1) + y_1(-4 + 2h^2q_1) + y_2 (2 + hp_1) = 2h^2r_1$$

$$i = 2 : \ y_1(2 - hp_2) + y_2(-4 + 2h^2q_2) + y_3 (2 + hp_2) = 2h^2r_2$$

$$i = 3 : \ y_2(2 - hp_3) + y_3(-4 + 2h^2q_3) + y_4 (2 + hp_3) = 2h^2r_3$$

$$i = n\text{-}1 : \ yn\text{-}_2(2 - hp_{n-1}) + y_{n-1}(-4 + 2h^2q_{n-1}) + y_n (2 + hp_{n-1}) = 2h^2r_{n-1}$$

Substituting boundary conditions $y(x_0) = y_0 = a$ and $y(x_n) = y_n = b$ and solving

the system of equations, we get the required solution.

The accuracy of the finite difference method depends on the h, with a smaller value of h gives more accurate solutions. Since y_0 and y_{n-1} are constants, we obtain a tridiagonal matrix from the system of equations at the left-hand side of the equation.

1.2 Example problem

Use the finite difference method to solve

$$y'' - y = x(x-4) \text{ given } y(0)=0 , y(4)=0$$

Solution:

given $p(x) = 0$, $q(x) = -1$, $r(x) = x(x - 4)$

Take $n = 4$, $h = \frac{4-0}{4} = 1$

at $i = 1$: $y_0(2-0) + y_1(-4+2*(-1)) + y_2(2+0) = 2(1)^3$

$2y_0 - 6y_1 + 2y_2 = 2*1(-3)$

$-6y_1+ 2y_2 = -6$ **----(1)**

at $i = 2$: $y_1(2-0) +y_2(-4+2*(-1)) +y_3(2+1*0) = 8$

$2y_1-6y_2+2y_3 = 8$ **----(2)**

at $i = 3$: $y_2(2-0)+y_3(-4-2)+y_4(2+1) = 2*1*3(-1)$

$2y_2-6y_3=-6$ **----(3)**

Solve 1, 2, and 3, we get

$y_1= 1.8571$ $y_2= 2.5714$ $y_3= 1.8571$

Similarly,

Take $n = 5$ $h = 0.8$

at $i = 1$: $y_0(2-0)+y_1(-4-2*0.8^2)+y_2(2+0)= 0.8(0.8-4)*2*(0.8)^2$

$-5.28y_1+2y_2= -3.2768$ **----(1)**

$i = 2$: $y_1(2-0)+y_2(-4-2)+y_3(2)= 1.6(1.6-4)*2*(0.8)^2$

$2y_2$-$5.28y_2$+$2y_3$= -4.9152 **----(2)**

i = 3: $y_2(2-0)+y_3(-4-2)+y_4(2)= 2.4(2.4-4)*2*(0.8)^2$

$2y_2$-$5.28y_4$+$2y_4$= -4.9152 **----(3)**

i = 4: $y_3(2-0)+y_4(-4-2)+y_5(2)= 3.2(3.2-4)*2*(0.8)^2$

$2y_3$-$5.28y_4$= -3.2768 **----(4)**

Solve 1, 2, 3, and 4, we get

y_1 =1.5451 y_2= 2.44067 y_3 = 2.44067 y_4= 1.5451

2. OBJECTIVES AND PROGRAM ALGORITHM

2.1 Objectives

The objective of this project is to solve a second-order linear ordinary differential equation using the tridiagonal matrix algorithm in MATLAB. The specific goals of the project are as follows:

1. Take inputs from the user, including the value of 'n', initial and boundary conditions, and the coefficients of the differential equation.
2. Compute the tridiagonal matrix coefficients and right-hand side values.
3. Solve the tridiagonal system and plot the points obtained.

2.2 Program Algorithm

1. Accept input values from the user, such as the boundary conditions, step size, and the number of grid points.
2. Compute the tridiagonal matrix coefficients (diagonal, lower diagonal, and upper diagonal) based on the given equation and discretization scheme.
3. Calculate the right-hand side values for each grid point based on the boundary conditions and the equation.
4. Adjust the boundary conditions if necessary to match the discrete form of the equation.
5. Solve the tridiagonal system of equations using a suitable numerical method (e.g., Thomas algorithm or Gaussian elimination with optimized tridiagonal matrix solver).
6. Obtain the solution values for each grid point.
7. Plot the solution curve using a plotting library or tool.
8. Display the solution curve to the user.

3. MATLAB IMPLEMENTATION

3.1 Block Diagram

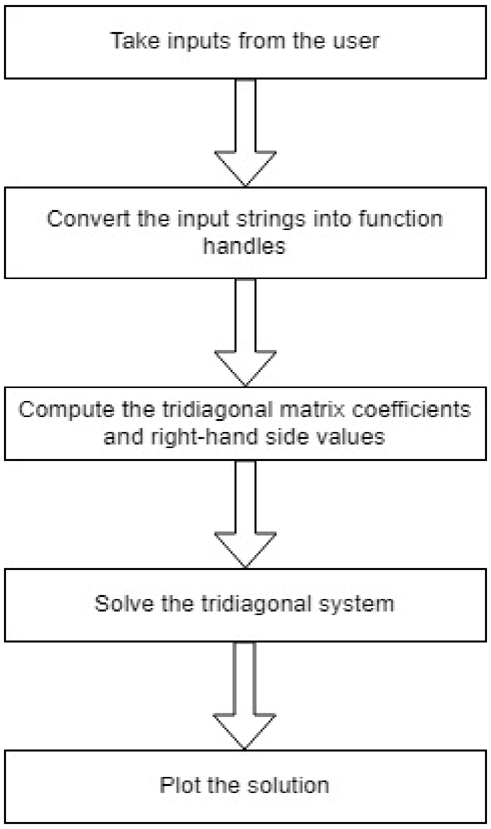

Take inputs from the user
Convert the input strings into function handles
Compute the tridiagonal matrix coefficients and right-hand side values
Solve the tridiagonal system
Plot the solution

Fig: Block diagram of code implementation

The code prompts the user to enter values for n, $y(x_0)$, x_0, $y(x_n)$, x_n, p, q, and r related to a differential equation. After taking the user from the input, It calculates the step size h based on the provided information and generates function handles p(x), q(x), and r(x). Using arrays, the code computes the coefficients of the tridiagonal matrix and the corresponding right-hand side values through a loop. It adjusts the right-hand side values to incorporate the boundary conditions by subtracting the respective coefficients multiplied by 'y(x0)' and 'y(xn)'. To solve the tridiagonal system, the code employs the 'Tridiag' function, which utilizes forward elimination and back substitution to derive the solution vector. Finally, the code visualizes the solution curve by plotting the points (x, y) for each interval, differentiating the initial and boundary points for clarity. Annotations are included to label the coordinates of each plotted point.

3.2 MATLAB Code

The code is divided into two sections. One is a program section and another is a function section that is called during the runtime of the program.

3.2.1 Program Section

This section encompasses the main program code.

```
% take inputs from the user
        n = input('enter the value of n: ');
        y0 = input('enter y(x0): ');
        x0 = input('enter x0: ');
        yn = input('enter y(xn): ');
        xn = input('enter xn: ');
        p = input('enter p(x): ','s');
        q = input('enter q(x): ','s');
        r = input('enter r(x): ','s');

% convert the input strings into function handles
        p = inline(p,'x');
        q = inline(q,'x');
        r = inline(r,'x');
% calculate h and range of x values
        h = (xn-x0)/n;
        xval = x0:h:xn;
        if(n<2)
           disp('the value of n cannot be less than 2');
           return;
        end

% initialize arrays
        arr_sub_diag = zeros(1,n-1);
        arr_diag = zeros(1,n-1);
        arr_super_diag = zeros(1,n-1);
        arr_rhs = zeros(1,n-1);

% compute the tridiagonal matrix coefficients and right-hand side values
        for i = 0:n-2
        arr_sub_diag(i+1) = 2-h*p(xval(i+2));
        arr_diag(i+1) =-4+2*h*h*q(xval(i+2));
        arr_super_diag(i+1) = 2+h*p(xval(i+2));
        arr_rhs(i+1) = 2*h*h*r(xval(i+2));

        % adjust the right-hand side values for boundary conditions
        if(i==0)
        arr_rhs(i+1) = arr_rhs(i+1) - (2-h*p(xval(i+2)))*y0;
         end
        if(i==n-2)
        arr_rhs(i+1) = arr_rhs(i+1) - (2+h*p(xval(i+2)))*yn;
        end
        end

        arr_rhs = arr_rhs';

% solve the tridiagonal system
        solution = Tridiag(arr_sub_diag, arr_diag, arr_super_diag, arr_rhs);

% plot the solution
        plot(xval, [y0 solution yn], 'b-');  % plot the solution points with initial values
        hold on;
        yval=[y0 solution yn];

        plot(xval, yval, 'ro');  % plot x vs y with red dots at specified points
        hold on;

% annotate the points
        for i = 1:length(xval)
           text(xval(i), yval(i), ['(' num2str(xval(i)) ', ' num2str(yval(i)) ')'], 'VerticalAlignment',
        'bottom');
        end

        xlabel('x');
        ylabel('y');
        title('Plot of x vs y');
        hold off;
```

3.2.2 Function Section

This section includes the definition of the 'Tridiag' function. It implements the tridiagonal matrix algorithm, which is a key component of the overall solution approach. The 'Tridiag' function takes in the tridiagonal matrix coefficients and the right-hand side values as input and solves the tridiagonal system using forward elimination and back substitution.

```
%function definition

        function x = Tridiag(e,f,g,r)
        % e = sub-diagonal vector
        % f = diagonal vector
        % g = super-diagonal vector
        % r = RHS vector
        % x = solution vector
        n=length (f);

% forward elimination
        for k= 2:n
        % compute factor for elimination
        factor = e(k)/f(k-1);
        % update diagonal and RHS vectors
        f(k) = f(k) - factorg(k-1);
        r(k) = r(k) - factorr(k-1);
        end

% back substitution
        x(n) = r(n)/f(n);
        for k = n-1:-1:1
        % compute solution vector backwards
        x(k) = (r(k)-g(k)*x(k+1))/f(k);
        end

% display solution
        for i = 1:length(x)
        fprintf('\ny%d = %f\n', i, x(i));
        end
        for i = 1:length(x)
        fprintf('\ny%d = %f\n', i, x(i));
        end
```

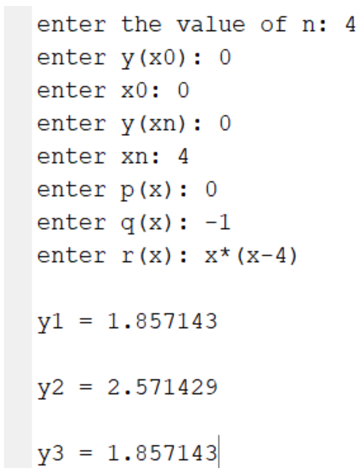
4. RESULT

From example at section 2.2, we obtained the following results:

y_1= 1.8571 y_2= 2.5714 y_3= 1.8571 for n=4

y_1 =1.5451 y_2= 2.44067 y_3 = 2.44067 y_4= 1.5451 for n=5

Entering the problem in the implemented MATLAB program we obtain



```
enter the value of n: 4
enter y(x0): 0
enter x0: 0
enter y(xn): 0
enter xn: 4
enter p(x): 0
enter q(x): -1
enter r(x): x*(x-4)

y1 = 1.857143

y2 = 2.571429

y3 = 1.857143
```

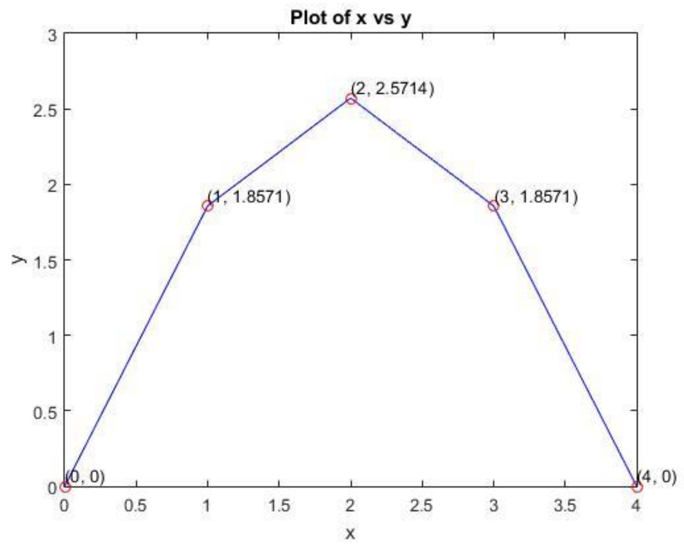


Fig: example problem (section 2.2) implemented for n=4

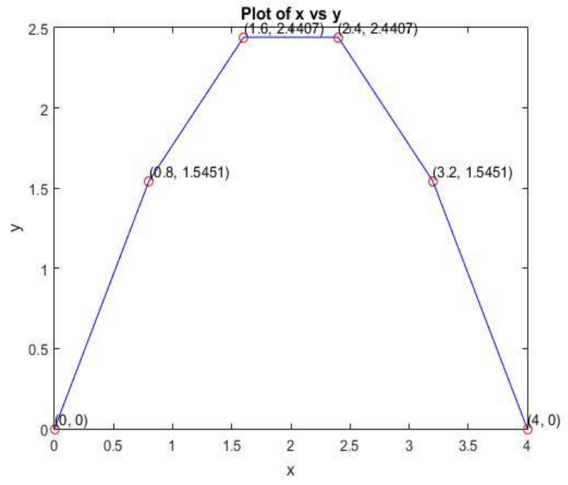


Fig: example problem (section 2.2) implemented for n=5

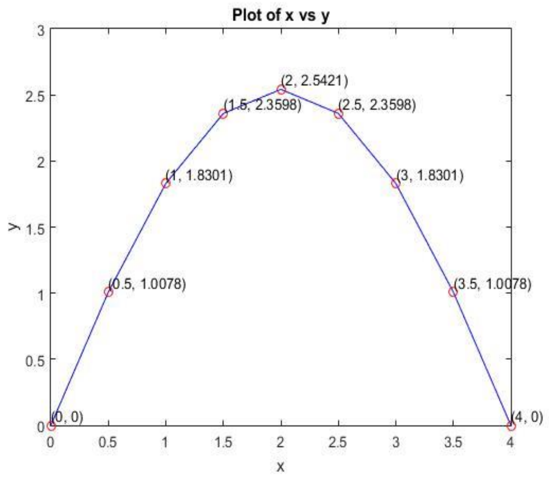


Fig: example problem (section 2.2) implemented for n=8

5. CONCLUSION

In conclusion, this project successfully implemented the tridiagonal matrix algorithm to solve a second-order linear ordinary differential equation. The code takes inputs from the user, computes the tridiagonal matrix coefficients and right-hand side values, adjusts the boundary conditions, solves the tridiagonal system, and plots the solution curve. The result of the calculation done manually in the example in section 2.2 and the result of the calculation of implemented program match with each other. Also, we can verify an increase in accuracy with an increase in discretization of the domain. Thus, it can be concluded that the implementation of the program in MATLAB is successful.