Anton Cifirov

A0278446N

Table of Contents

1. Essay on self learning

Creately.com defines self-learning as "an approach to learning where the individual makes the effort to identify their own learning needs, set learning goals, find the necessary resources, and evaluate their own knowledge." We live in a world where its never been easier to find information online. Thousands of free online learning resources, eBooks, YouTube tutorials on topics ranging from "How to knit a sock" to complete University courses from MIT, Harvard, Oxford etc. But should you incorporate self-directed learning in your life?

To understand the matter of self-learning more deeply lets look at historical context. Before 20th century, the major part of population did not have any opportunities to receive advanced academic education. As a matter of fact many of the famous and important inventors of that time had only attended primary school and all additional information they have learned by themselves. Famous example: James Watt, known for inventing different types of steam engine (also for being named after the units of power), is described as being "largely self-educated" in Robinson, Eric; McKie, Doublas. *Partners in Science: Letters of James Watt and Joseph Black*. This proves the power behind self-learning, this proves that one does not have to attend university to achieve great things.

According to statista.com the percentage of population in the United States who have completed high school or more academic institutions have risen from around 40% in year 1960 to more than 90% in year 2022. This may suggest that self-learning is not as important as it used to be, but I do not agree with that. Self-directed learning is a tool for extending your knowledge without the need to pay tuition fees or make a huge commitment to bachelors program in university. Many of the occupations today do not require you to earn a degree.
One particular area that comes to mind is Software development - a booming technological industry of the 21st century is one of the most flexible in terms of hiring people and working conditions. Job applicants are often evaluated only by their skills, knowledge, and university degree is a plus, but not a necessity. Here are some of the most famous self-taught programmers: Mark Zuckerberg (co-founder and CEO of Meta(formerly Facebook), Bill Gates (co-founder of Microsoft), Linus Torvalds (creator of the Linux operating system). That is why self-learning is still relevant today.

For me, self-learning is flexibility and freedom. In the hands of a person who is aware of his own peculiarities of studying, it is a very powerful tool to make studying more effective. In my own experience, there have been a few times where I would learn more material and study more efficiently by not attending the lecture in a university and going through the topic by myself, as there are many different sources that provide different examples, explanations, some are simpler some are more in depth. The flexibility that self-learning gives you in

terms of the pace of studying is another big factor in favor of self-learning. We are all different and I believe that tuning the speed of the flow of information in your brains to your liking and abilities, in some cases, can be better than trying to adapt to the pace and difficulty of the live lectures.

In conclusion, self-learning is a way to extend and deepen your knowledge in addition to formal education, it is a way to learn new skills that can help you in your career or that are just for you to enjoy, either way 2023 is the best time to make use of all the tools and technologies of modern era and upgrade yourself.

2. Essay on PDE

Mathematics is a science of thought and logic. It can be used to describe almost anything in the real world and the abstract world of thoughts. If you ever went trough a Physics textbook and wondered - "how did someone came up with these complicated formulas?". It is very likely that the magic behind those equations involved solving a differential equation.

Differential equations are equations that contain derivatives of a unknown function that we are trying to find. They are often used to describe systems where it is easier to say something about the rate of change of a variable at a time t then to say something about the value of the variable at a time t. These equations are notorious for being extremely difficult to solve. The equations that contain derivatives with respect to only one variable are called Ordinary Differential Equations (or ODE for short). And equations that contain partial derivatives of unknown function with respect to 2 or more independent variable are called Partial Differential Equations (PDE for short). The solution for such problems are not concrete values, but a set of functions. A differential equation with initial conditions y(x0) = y0, y'(x0)= y1,…,y(n)(x0)= yn (number of initial conditions depends on the order of ODE), is called an Initial Value Problem. The simplest example of such equation would be equation of motion: Given that acceleration is some constant a, what is the function of x coordinate of time x(t). We now that acceleration is the rate of change of speed, and speed is the rate of change of position. So we can write down our equation as follows: d2x/dt2 = a. This is a first order separable ODE, after separating the variables and integrating for dt twice we get x(t) = at2/2 + c1t + c2 where c1 are constants defined by the two initial conditions at time t = 0 x(0) = x0 and x'(0) = v0 from here c1= v0, c2 = x0. So now we have x(t) = at2/2 + v0t + x0. This and many more equations in physics (and not only physics) are derived in similar way. Here are few examples: Radioactive decay, pendulum model, harmonic oscillations etc. But what about PDE? The underlying principle is the same, we want to make some mathematical statement about derivatives and then solve the equation to find the unknown set of functions of more than 1 variable. In the case of ODE initial conditions were constants but in this case initial conditions are functions of 1 or more independent variables. Let us pick a simple enough example to get acquainted with this idea. Heat equation

∂u/∂t = k*Laplacian(u); xi1<xi<xi2; t1<t<t2; where Laplacian(u) = ∂2u/∂x12+ ∂2u/∂x22 + … + ∂2u/∂xn2 and xi1,xi2,t1,t2,k,are some real valued constants for all 0 <= i <= n. The heat equation describes the relationship between the rate of the rate of change in temperature with respect to x (position), and the rate of change of temperature with respect to t (time). Lets reduce the dimensions of x to 1 for simplicity. We have ∂u/∂t = k*∂2u/∂x2, x1<x<x2*; t1<t<t2. So the function u(x,t) that we are trying to find describes the temperature at any point in our 1 dimensional object (we can imagine it being a stick or a wire) and at any time, given that we know temperature at each point x at t=0. Although solving this equation analytically also involves separating the variables, complete solution is quite difficult, so I will not attempt to demonstrate it. Here are some other famous examples of PDE's: wave equation; Laplace equation - describes the distribution of a scalar field in

a region of space(generated by chat GPT) and is applied to electrostatics, fluid dynamics, heat conduction, electromagnetics fields; Maxwell equations - they describe the relationship between magnetic and electric fields.

In conclusion, Partial Differential Equations are a very important part of mathematics that is widely used in Engineering sciences, Physics, Biology, Chemistry, even in Machine Learning algorithms. Our world is definitely a better place with PDE's!

3. Finite-difference method for PDE

```
% Consider the following PDE:
```

$$\frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = 0, 0 < x < 2, 0 < t < 10;$$
$$u(0,t) = u(2,t) = 0; 0 < t;$$
$$u(x,0) = \sin(\pi x/2), 0 \le x \le 2.$$

This link will take you to the Google drive pdf file with my pen and paper solution: Solution_by_hand_pdf

As we can see from the solution table the finite difference method with this number of points is highly unstable.

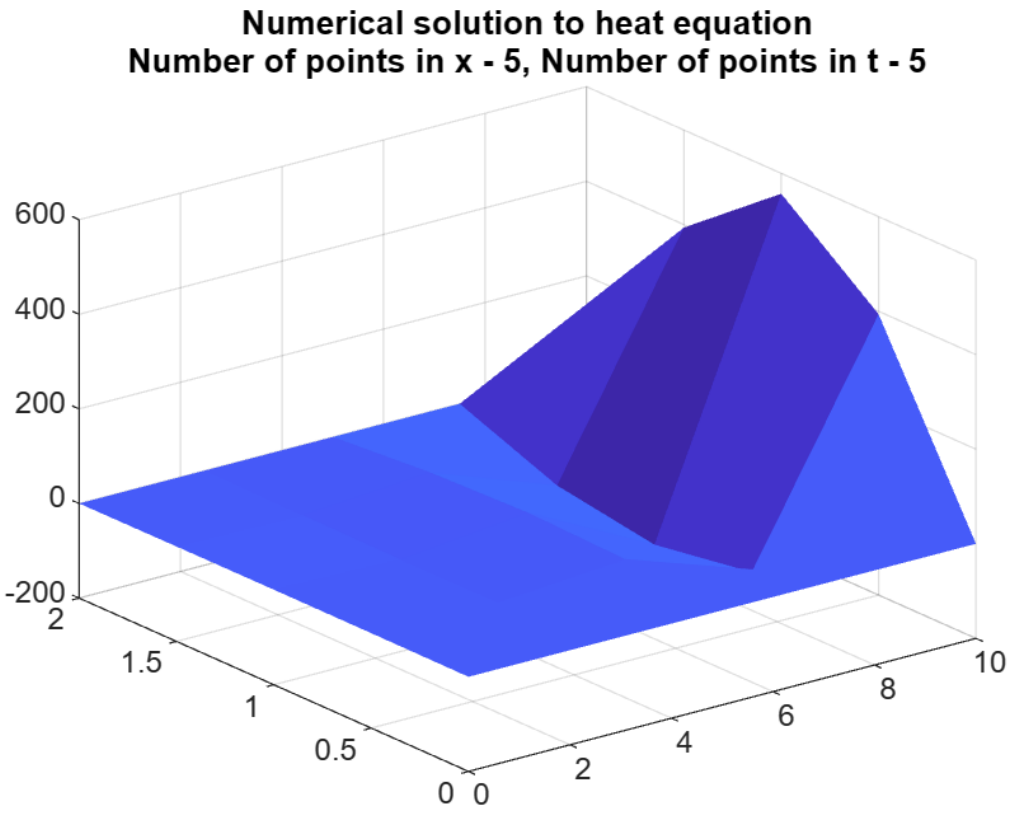
```
%Input data into matlab
x = [0, 0.5, 1, 1.5, 2];
t = [0, 2.5, 5, 7.5, 10];
u = [
    0          0       0       0         0    ;
    sqrt(2)/2  -3.435  16.68  -81.06    393.8;
    1          -4.858  23.60  -114.6     557.0;
    sqrt(2)/2  -3.435  16.68  -81.06    393.8;
    0           0      0       0         0    ;
    ];
```

4. Verification of solution of PDE

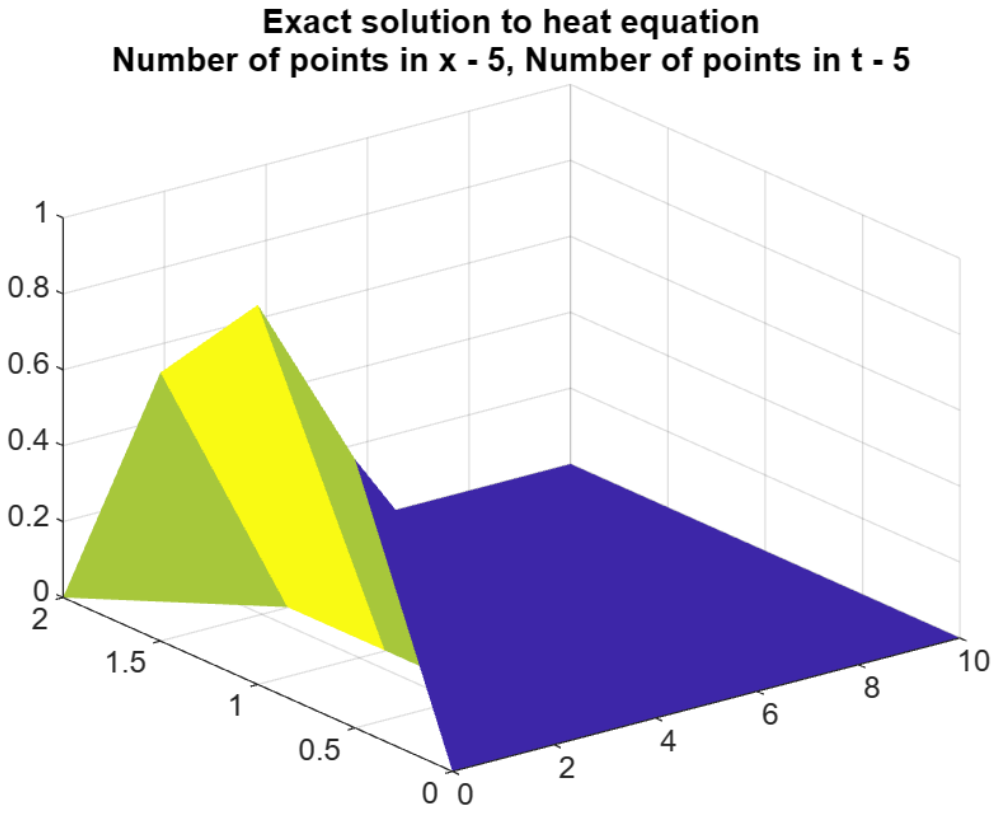
Exact solution to the PDE from section 3 is as follows:

$$u(x,t) = e^{-(\pi^2/4)t}\sin(\pi x/2)$$

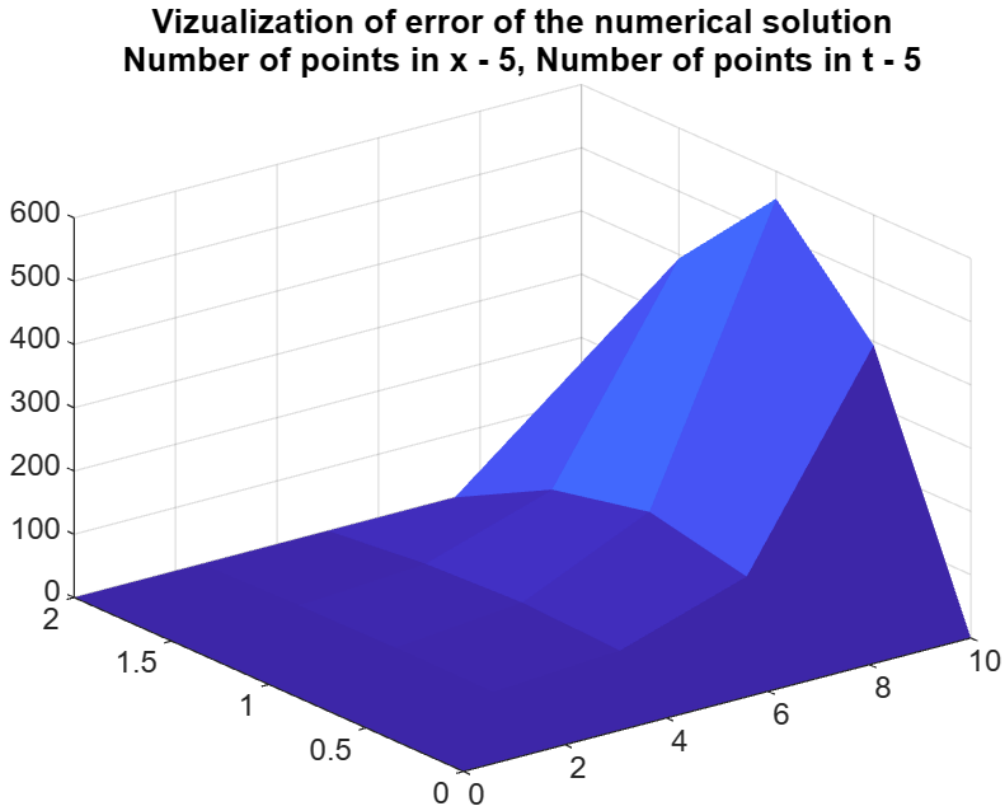
```
%Plotting the numerical solution by hand
surf(t,x,u, 'EdgeColor', 'none')
title_string = sprintf("Numerical solution to heat equation\nNumber of
points in x - %d, Number of points in t - %d", 5, 5);
title(title_string)
```




```
%Exact solution
exact_solution_function = @(x,t) exp(-1*(pi^2/4)*t)*sin(pi*x/2);
[exact_sol,x,t] = exact_solution(exact_solution_function,0,2,0,10,5,5);

%Plotting
surf(t,x,exact_sol, 'EdgeColor', 'none')
title_string = sprintf("Exact solution to heat equation\nNumber of points in
x - %d, Number of points in t - %d", 5, 5);
title(title_string)
```




```
%Absolute error
error_matrix = abs(u - exact_sol);

%Plotting
surf(t,x,error_matrix, 'EdgeColor', 'none')
title_string = sprintf("Vizualization of error of the numerical
solution\nNumber of points in x - %d, Number of points in t - %d", 5, 5);
title(title_string);
```




```
%As we can see the error is quite dramatic, we will try to fix this issue
%in the next segment
```

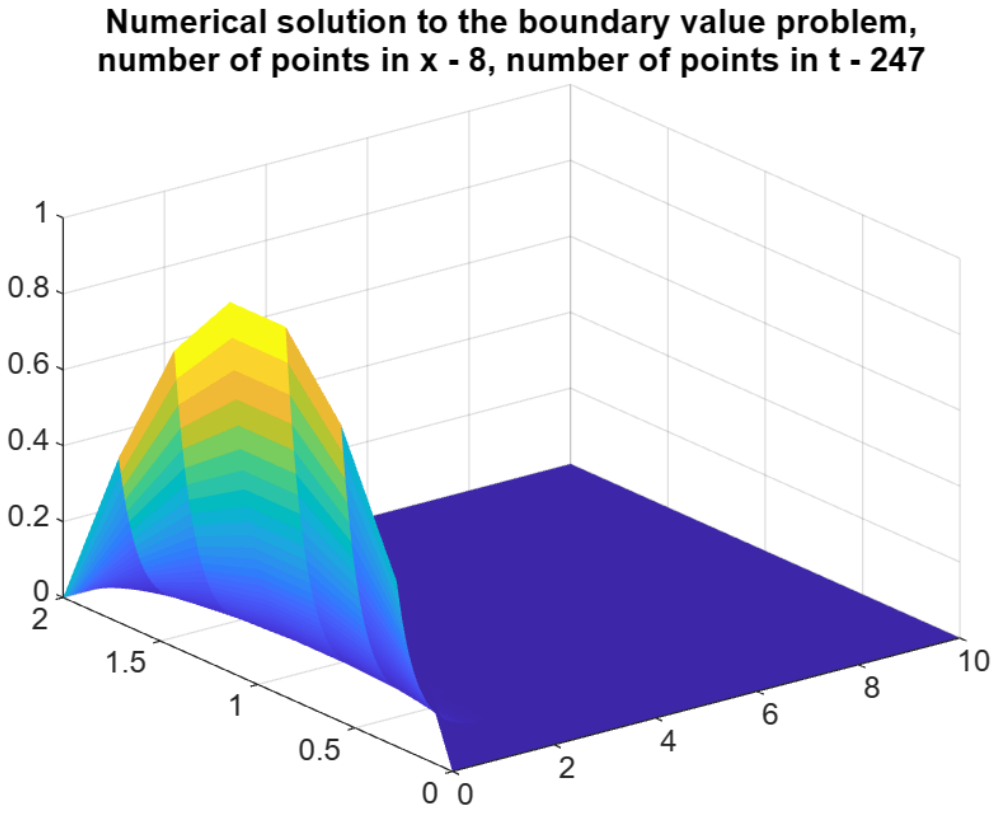
5. Numerical solution of PDE

```
format long;
%du/dt=const*d^2u/dx^2
%u(0,t) = u(2,t) = 0, t>0
%u(x,0) = sin(pi*x/2), 0<=x<=2
% 0<x<2 & 0<t<10

%Step 1) Discretize

%(t_0,...,t_N), where
t_1 = 0;
t_N = 10;
%and
%(x_0,...,x_J), where
x_1 = 0;
x_J = 2;

%Set boundary values
left_boundary = 0;
```

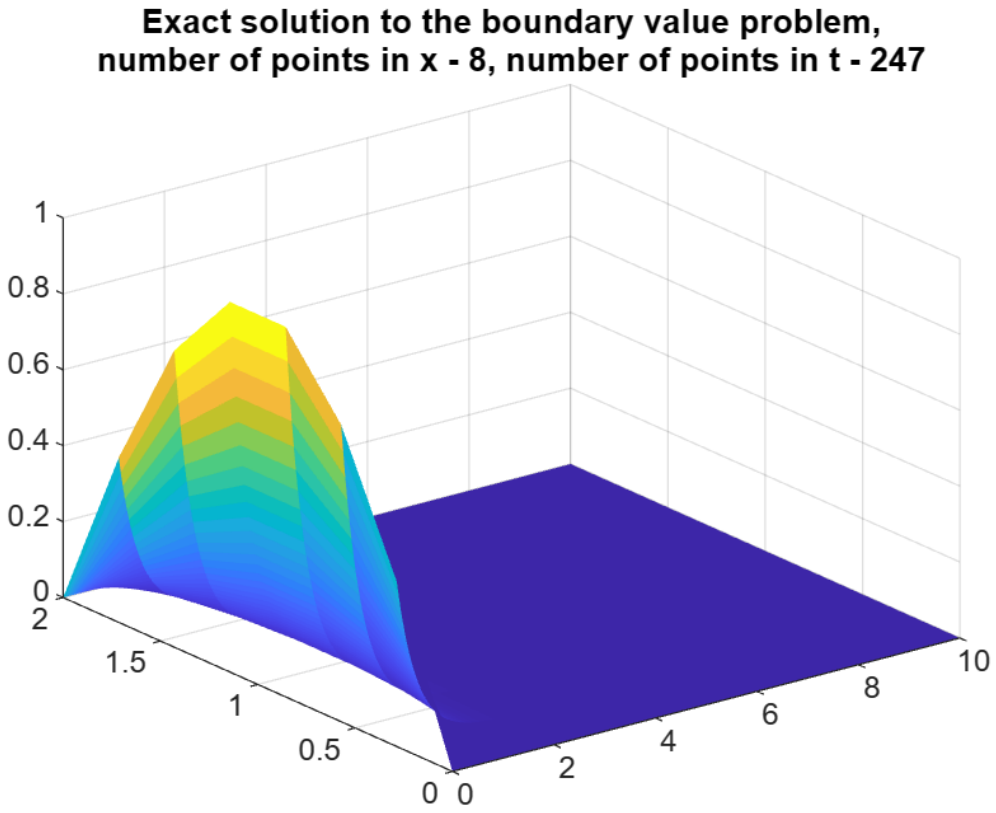
```
right_boundary = 0;


%Create initial condition funciton
f = @(x) sin(pi*x/2);
%These are different initial conditioins to experiment with
f1 = @(x) cos(pi*x/2);
f2 = @(x) -(x-1).^2+1;

initial_condition = f;
const = 1;

% Set number of points for x - J
J1 = 8;


%Call solve_pde_2 and store the solution in solution1_matrix
%solve_pde_2 sets s = 0.5 and calculates N1 such that s <= 0
[numerical_solution_matrix1,x1,t1] = solve_pde_2(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J1);
```

```
s value is 0.497967
Number of points in time = 247
Number of points in x = 8
```

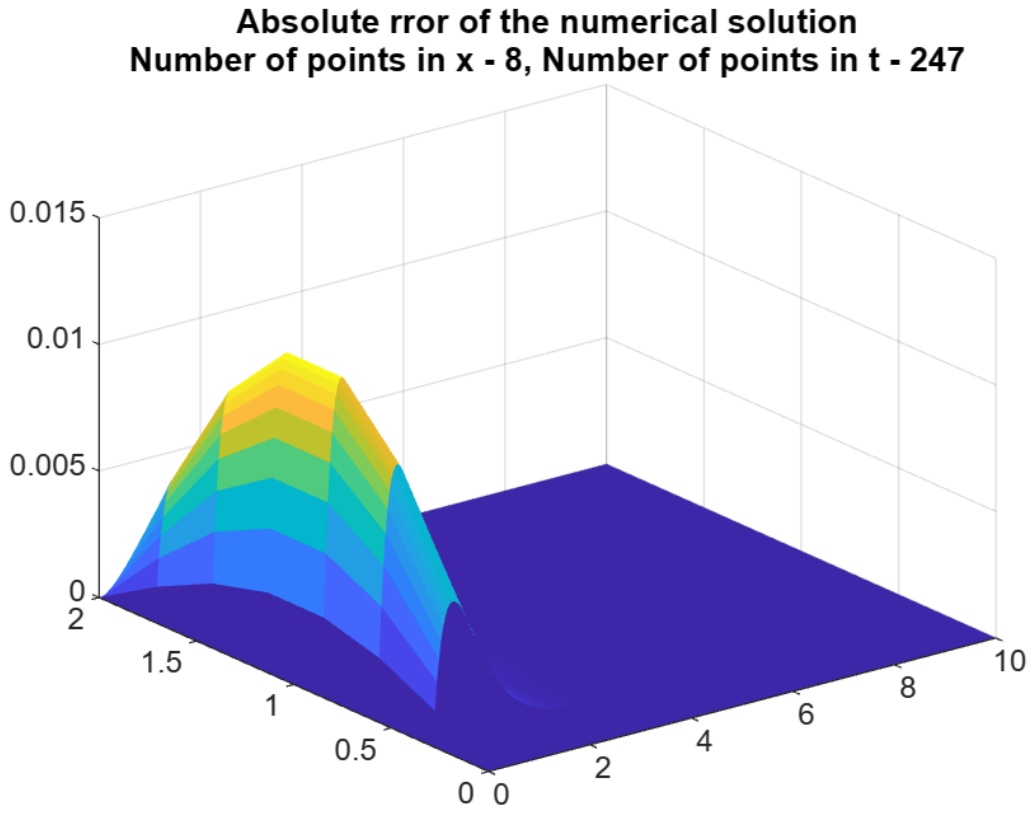
```
N1 = length(t1);

%Plotting
surf(t1,x1,numerical_solution_matrix1, 'EdgeColor', 'none')
title_string = sprintf("Numerical solution to the boundary value
problem,\nnumber of points in x - %d, number of points in t - %d", J1, N1);
title(title_string);
```




```
%Exact solution
exact_solution_function = @(x,t) exp(-1*(pi^2/4)*t)*sin(pi*x/2);
[exact_solution_matrix1,x1,t1] =
exact_solution(exact_solution_function,x_1,x_J,t_1,t_N,J1,N1);

%Plotting
surf(t1,x1,exact_solution_matrix1, 'EdgeColor', 'none')
title_string = sprintf("Exact solution to the boundary value
problem,\nnumber of points in x - %d, number of points in t - %d", J1, N1);
title(title_string);
```




```
%Error
%Calculate error for each numerical_solution_matrix1(j,n) and
exact_solution_matrix(j,n) element
error_matrix1 = abs(numerical_solution_matrix1 - exact_solution_matrix1);

%Plotting
surf(t1,x1,error_matrix1, 'EdgeColor', 'none')
title_string = sprintf("Absolute rror of the numerical solution\nNumber of
points in x - %d, Number of points in t - %d", J1, N1);
title(title_string);
```

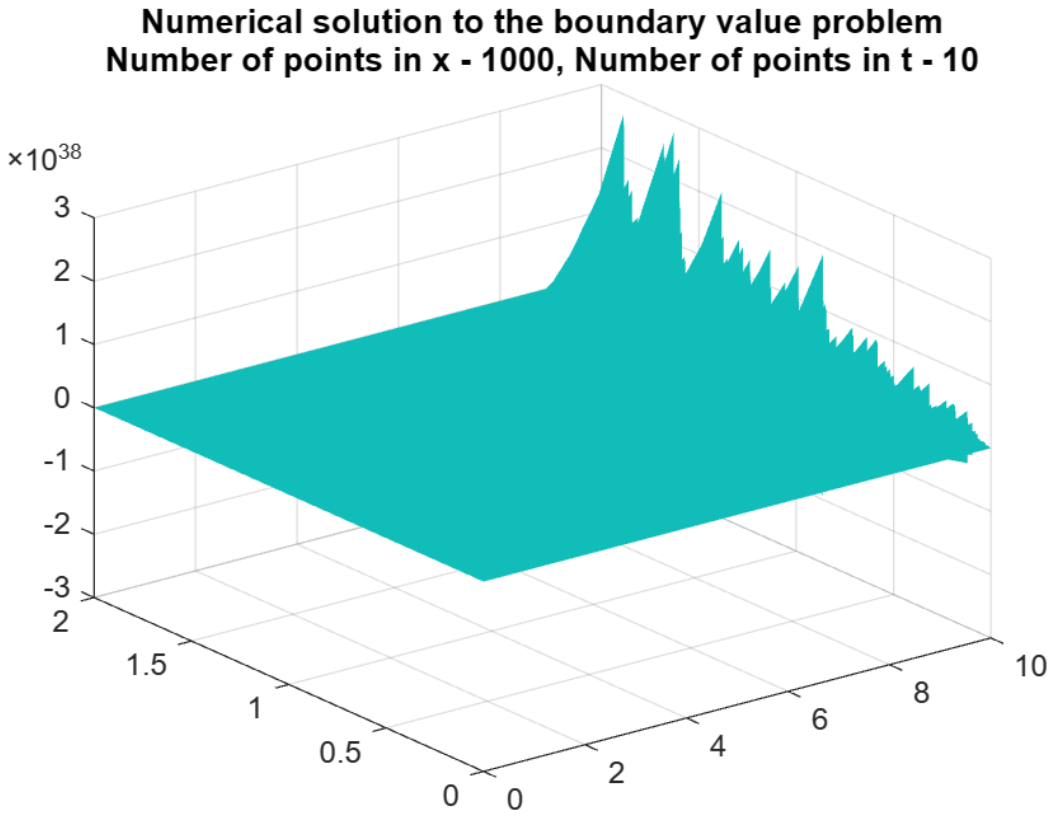


6. Plotting exact and numerical solutions when x-step is J = 1000 and t-step N = 10

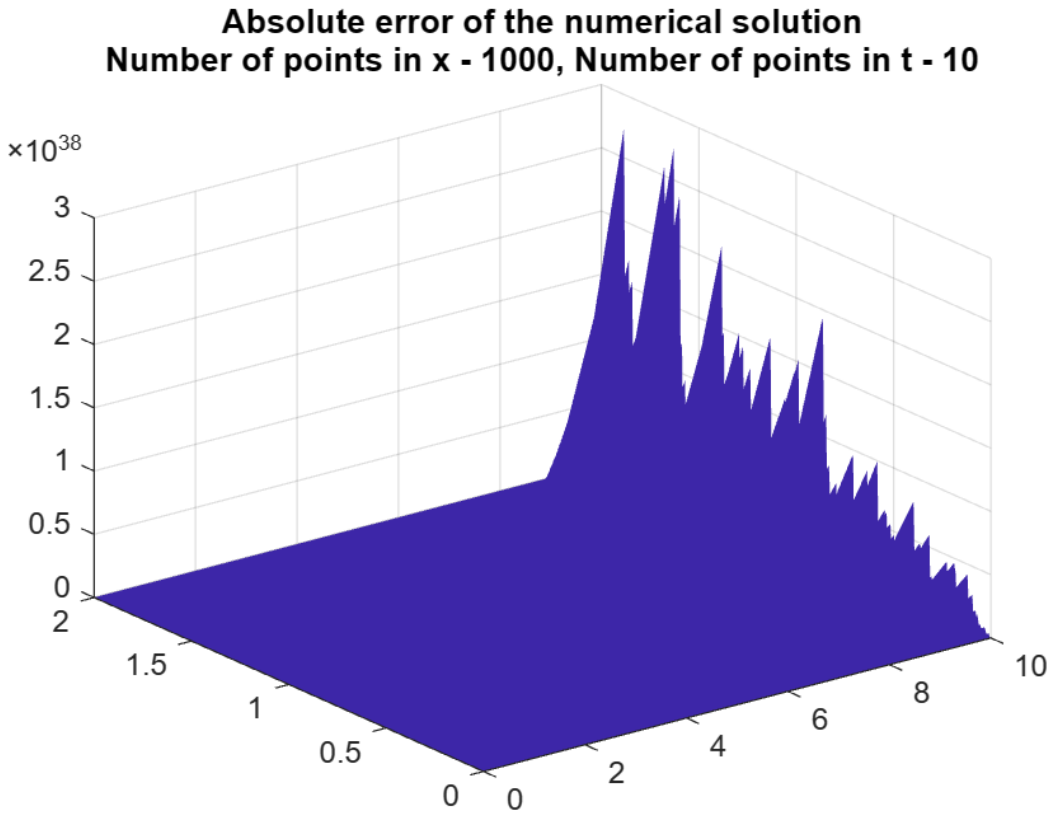
```
% Note - in the assignment
%   N is used for points in x
%   T is used for points in t
% In this code
%   J is used for points in x
%   N is used for points in t
J2 = 1000;
N2 = 10;
[numerical_solution_matrix2,x2,t2] = solve_pde_1(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J2,N2);
```

```
s value is 277222.500000
Number of points in time = 10
Number of points in x = 1000
```

```
%Plotting
surf(t2,x2,numerical_solution_matrix2, 'EdgeColor', 'none')
title_string = sprintf("Numerical solution to the boundary value
problem\nNumber of points in x - %d, Number of points in t - %d", J2, N2);
title(title_string);
```

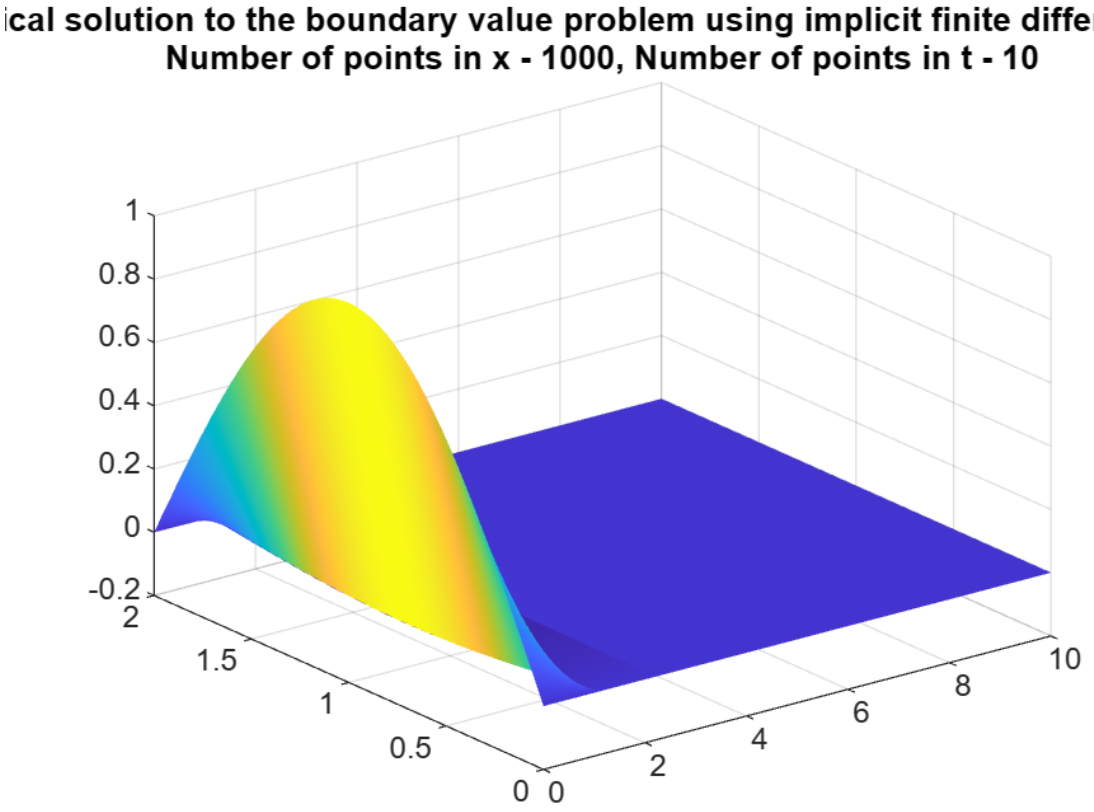



```
%Exact solution
[exact_solution_matrix2,x2,t2] =
exact_solution(exact_solution_function,x_1,x_J,t_1,t_N,J2,N2);

%Absolute error
error_matrix2 = abs(numerical_solution_matrix2 - exact_solution_matrix2);

%Plotting
surf(t2,x2,error_matrix2, 'EdgeColor', 'none')
title_string = sprintf("Absolute error of the numerical solution\nNumber of
points in x - %d, Number of points in t - %d", J2, N2);
title(title_string);
```

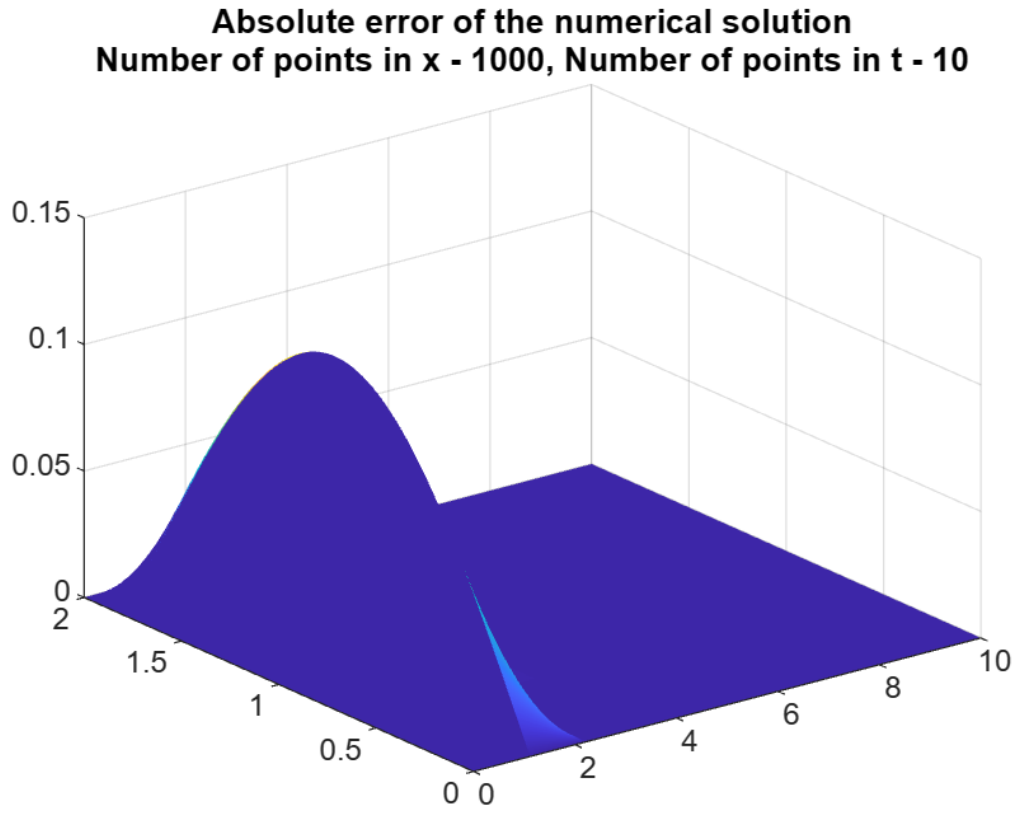



```
delx = (x_J-x_1)/(J2-1);
delt = (t_N-t_1)/(N2-1);

%Numerical solution is highly unstable

% r  = 277222.500000 > 1/2 -> algorithm is highly unstable
% To reduce instability and error we can either make J smaller, and N
% larger so that r would be smaller than 1/2, or we could modify our
% function that solves PDE to use implicit time steps instead of explicit.

% In explicit time steps the future values of the solution are computed
% explicitly in terms of previous time steps.

% In implicit time steps the future values of the solution are computed
% using a system of equations,and they often require solving a linear or
% nonlinear system of equations at each time step in other words, the "now"
% value of solution depends both on "past" and "fututre".

%Solve using implicit time steps
[implicit_time_solution, t2, x2] = solve_pde_implt(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J2,N2);

%Plotting
surf(x2,t2,implicit_time_solution, 'EdgeColor', 'none')
```

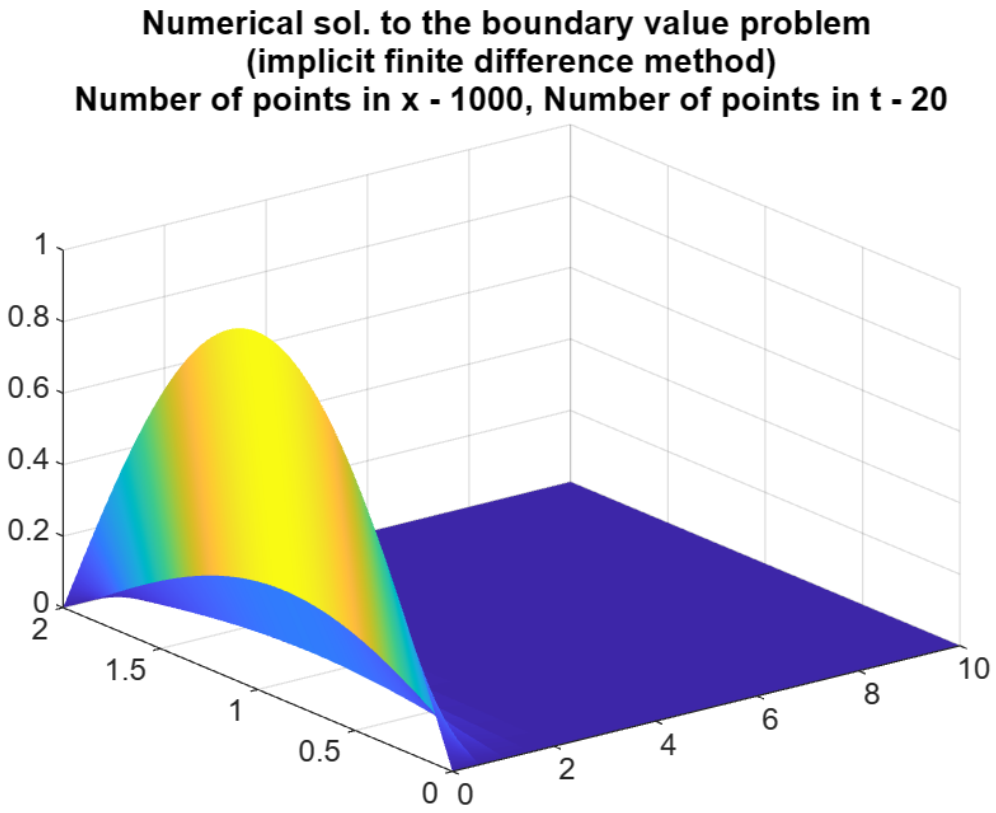
```
title_string = sprintf("Numerical solution to the boundary value problem
using implicit finite difference method\nNumber of points in x - %d, Number
of points in t - %d", J2, N2);
title(title_string);
```




```
%Absolute error
error_matrix3 = abs(implicit_time_solution - exact_solution_matrix2);

%Plotting the error of implicit finite difference method solution
surf(x2,t2,error_matrix3, 'EdgeColor', 'none')
title_string = sprintf("Absolute error of the numerical solution\nNumber of
points in x - %d, Number of points in t - %d", J2, N2);
title(title_string);
```




```
% As we can see the solution is much more stable although it is still
% inacurate, from physics and common sence we understand that it is
% impossible for the temperture to be negative at any x and any t, yet
% this solution gives us such result.
% To get more accurate results, lets try increasing N

N2 = 20;

[implicit_time_solution, t2, x2] = solve_pde_implt(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J2,N2);



%Plotting
surf(x2,t2,implicit_time_solution, 'EdgeColor', 'none')
title_string = sprintf("Numerical sol. to the boundary value problem
\n(implicit finite difference method)\nNumber of points in x - %d, Number
of points in t - %d", J2, N2);
title(title_string);
```




```
%Increasing J to J = 20 seems to fix this problem.
```

7. Function definitions

```
function [numerical_solution_matrix,x,t] = solve_pde_1(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J,N)
    %This function numericly solves Boundary value problem for PDE given all
required parameters and J, N

    %Calculate the length of time step and space step
    delx = (x_J-x_1)/(J-1);
    delt = (t_N-t_1)/(N-1);

    s = const*delt/((delx)^2);
    fprintf("\ns value is %f\n", s)
    fprintf("Number of points in time = %d\nNumber of points in x =
%d\n",N,J)

    %Create vectors for x and t
    x = x_1:delx:x_J; %vector of discretized x values
    t = t_1:delt:t_N; %vector of discretized t values
```

```
    %Derived by hand recurent formula
    %u(x,t + delt) = r( u(x+delx,t) + u(x-delx,t) ) + u(x,t)(1-2r);
    %where s = delt/((delx)^2)
    %Change notation
    %u(xj,tn+1) = s( u(xj+1,tn) + u(xj-1,tn) ) + u(xj,tn)(1-2s)
    %u_j,n = u(xj,tn)
    %Then
    %u_j,n+1 = s( u_j+1,n + u_j-1,n ) + u_j,n(1-2s)


    %Declaring a matrix
    numerical_solution_matrix = zeros(J,N);

    %Using initial condition u(x,0) = f(x) calculate u_j,0 for all j
    numerical_solution_matrix(:,1) = initial_condition(x);

    %Using boundary values
    numerical_solution_matrix(1,:) = left_boundary;
    numerical_solution_matrix(J,:) = right_boundary;

    %Using recurance relation calculate the rest of data points
    for n = 2:(N)
        for j = 2:(J-1)
            numerical_solution_matrix(j,n)
= s*( numerical_solution_matrix(j+1,n-1)
+ numerical_solution_matrix(j-1,n-1) ) +
numerical_solution_matrix(j,n-1)*(1-2*s);
        end
    end
end


function [numerical_solution_matrix,x,t] = solve_pde_2(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J)
    %This function solves Boundary value problem given all the required
    %parameters and J. N is calculted inside the function using
    %Courant–Friedrichs–Lewy condition
    %s = const*delt/((delx)^2) < 1/2 otherwise numerical solution will be
    %highly unstable - so in this variant of finite difference method
    %we take N to be as little as possible, but such that s < 1/2
    %Because of that we use the number of steps in x - N and s,
    %and using those calculate number of steps in time - J
    %assign 0.5 to s because when s>0.5 finite difference method can become
unstable
    %lower s result in more points in t,
    %higher s results in less points in t.
    s = 0.5;

    %calculate delx and N from r value
    delx = (x_J-x_1)/(J-1);
```

```
    N = ceil((t_N - t_1)/(s*(delx^2))/const + 1);
    %Calling a function that solves the given Boundary value problem
    [numerical_solution_matrix,x,t] = solve_pde_1(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J,N);

end



%This function was partly generated by ChatGPT
function [numerical_solution_matrix,x,t] = solve_pde_implt(left_boundary,
right_boundary,x_1,x_J,t_1,t_N,initial_condition,const,J,N)
    %This function numericly solves Boundary value problem for heat equation
given all required parameters and J, N

    % Parameters
    const = 1;   % Thermal diffusivity
    dx = (x_J-x_1) / (J+1);
    dt = (t_N-t_1) / (N+1);

    % Initialize the spatial grid
    x = linspace(x_1, x_J, J);
    t = linspace(t_1, t_N, N);

    % Initialize the solution grid
    numerical_solution_matrix = zeros(J, N);

    % Set initial condition
    numerical_solution_matrix(:, 1) = initial_condition(x);

     %Using boundary values
    numerical_solution_matrix(1,:) = left_boundary;
    numerical_solution_matrix(J,:) = right_boundary;

    % Create tridiagonal matrices A and B
    s = const * dt / (2 * dx^2);
    A = diag(1 + 2 * s * ones(1, J - 2)) - diag(s * ones(1, J - 3), 1) -
diag(s * ones(1, J - 3), -1);
    B = diag(1 - 2 * s * ones(1, J -2)) + diag(s * ones(1, J - 3), 1) +
diag(s * ones(1, J - 3), -1);

    % Time-stepping loop
    for n = 1:(N-1)
        % Compute the right-hand side using B matrix and the previous time
step
        rhs = B * numerical_solution_matrix(2:(J-1), n);

        % Enforce boundary conditions
        rhs(1) = rhs(1) + s * numerical_solution_matrix(1, n + 1);
        rhs(end) = rhs(end) + s * numerical_solution_matrix(J, n + 1);
```

```
        % Solve the linear system using the A matrix and the right-hand side
        numerical_solution_matrix(2:(J-1), n + 1) = A \ rhs;
    end
end

function [exact_solution_matrix,x,t] =
exact_solution(exact_solution_function, x_1,x_J,t_1,t_N,J,N)
    delx = (x_J-x_1)/(J-1);
    delt = (t_N-t_1)/(N-1);
    x = x_1:delx:x_J; %vector of discretized x values
    t = t_1:delt:t_N; %vector of discretized t values
    exact_solution_matrix = zeros(J,N);
    for n = 1:(N)
        for j = 1:J
            exact_solution_matrix(j,n) = exact_solution_function(x(j),t(n));
        end
    end
end
```