MATH-6840
Vignesh Ramakrishnan
Due: Monday January 31, 2022

Problem Set 2

1. (15 pts.) *Adopted from NPDE exercise 1.2.1:*
Write a code to approximately solve

$$\begin{aligned} v_t &= \nu v_{xx}, \qquad x \in (0,1), \qquad t > 0 \\ v(x,0) &= f(x), \qquad x \in (0,1) \\ v(0,t) &= a(t), \qquad v(1,t) = b(t), \qquad t \geq 0. \end{aligned}$$

Use the grid $x_j = j\Delta x$, with $j = 1, 2, \ldots, N$, and $\Delta x = 1/N$ (as described in the text), and apply the usual second-order centered spatial discretization with forward Euler time integration, i.e.

$$D_{+t}v_j^n = \nu D_{+x}D_{-x}v_j^n.$$

Use $f(x) = \sin(2\pi x)$, $a = b = 0$, $\nu = 1/6$, and $N = 10$. Find solutions at $t = 0.06$, $t = 0.1$, $t = 0.9$, and $t = 50.0$. For the first three values of t,use $\Delta t = 0.02$. To speed the solution to the last value of t, you might use a larger value for Δt. Determine how large you can choose Δt and still get results that might be correct. Compare and contrast your solution to the exact solution.

The second-order centered spatial discretization with forward Euler time integration is given by the set of equations for all the nodes ranging from $j = 0, 1, 2, 3..., N-1, N$:

$$\begin{aligned} D_{+t}v_j^n &= \nu D_{+x}D_{-x}v_j^n \\ v_j^{n+1} &= v_j^n + \frac{\nu\Delta t}{\Delta x^2}\left(v_{j+1}^n - 2v_j^n + v_{j-1}^n\right) \end{aligned}$$

One ghost node is added to the left side of the boundary node $x_j,\ j = 0$ and it is called x_{-1}. Similarly one ghost node is added to the right side of the boundary node $x_j,\ j = N$ and it is named as x_{N+1}. Compatibility boundary conditions are applied to figure out the values at these ghost nodes and this technique is used to apply the boundary conditions $v(0,t) = a(t)$, and $v(1,t) = b(t)$, $t \geq 0$. Compatibility boundary conditions specify that a time derivative is obtainable at both the boundary conditions and hence the following formulation comes about:

$$\begin{aligned} v(0,t) &= a(t), \\ v_t(0,t) &= a_t(t) \\ \nu v_{xx}(0,t) &= a_t(t) \\ v_{-1}^n &= \frac{\Delta x^2}{\nu}a_t(t^n) + 2v_0^n - v_1^n \end{aligned}$$

Similarly

$$v_{N+1}^n = \frac{\Delta x^2}{\nu}b_t(t^n) + 2v_N^n - v_{N-1}^n$$

Listing 1: Euler Forward-Step time marching - Heat Equation (Q.1)

```
function [] = HeatEqn1D(N,dt,xlim1,xlim2,tlim1,tlim2)
% $Author: Vignesh Ramakrishnan$
% $RIN: 662028006$
% v_t = \nu v_xx, x \in (0,1), t > 0
% v(x,t=0) = sin(2\pi*x), v(0,t) = v(1,t) = 0
% \nu = 1/6;
% This function intends to use Finite Difference method to get a numerical
% solution to the heat equation described above. The compatability Boundary
% condtions are utilized to enforce Boundary conditions at the Boundary
% points. Euler forward stepping is utilized to march in time.

% Inputs: N     - Number of elements - describes spatial discretization
%         dt    - temporal discretization
%         xlim1 - left end of the spatial boundary
%         xlim2 - right end of the spatial boundary
%         tlim1 - start time (initial condition time)
%         tlim2 - end time of simulation
% Output: Prints the time evolution across the spatial grid for the heat
%         equation described above

%% Trial runs

% HeatEqn1D(10,0.02,0,1,0,0.1); - stable solution
% HeatEqn1D(40,0.02,0,1,0,0.1); - unstable solution

%% code


nu = 1/6;         % coefficient of heat transfer
a  = 0   ;        % time varying function at left boundary
b  = 0   ;        % time varying function at right boundary
dx = 1/N;         % dx -spatial discretization

r  = nu*dt/dx^2;% r = CFL number

ng = 1   ;        % number of ghost points at one boundary
NP = N+1+2*ng;    % Total number of spatial points
ja = ng+1;        % xlim1's index number
jb = NP-ng;       % xlim2's index number

x  = (xlim1:dx:xlim2);
t  = (tlim1:dt:tlim2);

u  = zeros(length(t),NP);

% set initial conditions for the spatial grid
```

```
u(1, ja:jb)     = sin(2*pi*x);
u(1,ja)         = a;
u(1,jb)         = b;
u(1,1)          = 2*u(1,ja) - u(1,ja+1);
u(1,NP)         = 2*u(1,jb) - u(1,jb-1);

for i=2:length(t)
    for j = ja:jb
        u(i,j) = u(i-1,j) + r*(u(i-1,j+1)-2*u(i-1,j)+ u(i-1,j-1));
    end

    u(i,ng) = 2*u(i,ja) - u(i,ja+1);
    u(i,NP) = 2*u(i,jb) - u(i,jb-1);

end

Legend   = strcat('t = ',num2str(tlim2));
str1     = strcat('N = ', num2str(N));
str2     = strcat('\nu = ',num2str(nu));

str      = append('$',str1,',\ ',str2,'$');

figure
plot(x,u(end,ja:jb));
hold on;
grid on;
ylim([-0.8  0.8]);
xlabel('$x$','FontSize',16,'Interpreter','latex');
ylabel('$v(x,t)$','FontSize',16,'Interpreter','latex');
legend(Legend,'FontSize',16,'Interpreter','latex');
title('Euler Forward-Step time marching',str,'Interpreter','latex','FontSize',

[X,T] = meshgrid(x,t);

figure
surf(X,T,u(:,ja:jb));
xlabel('$x$','FontSize',16,'Interpreter','latex');
ylabel('$t$','FontSize',16,'Interpreter','latex');
zlabel('$v(x,t)$','FontSize',16,'Interpreter','latex');
title('Euler Forward-Step time marching',str,'Interpreter','latex');

end
```

The solutions at time $t = 0.06$, $t = 0.1$, $t = 0.9$, and at $t = 50$ are displayed in Fig 1. To speed up the solution, Δt was raised from 0.02 to 0.03 and the solution still looked feasible. $\Delta t = 0.03$ yields a CFL number $r = 0.5$ and if Δt is increased to any value over 0.03, CFL number, $r > 0.5$ and this makes the solution look implausible and at $t = 50$, the values of the solutions blow up.

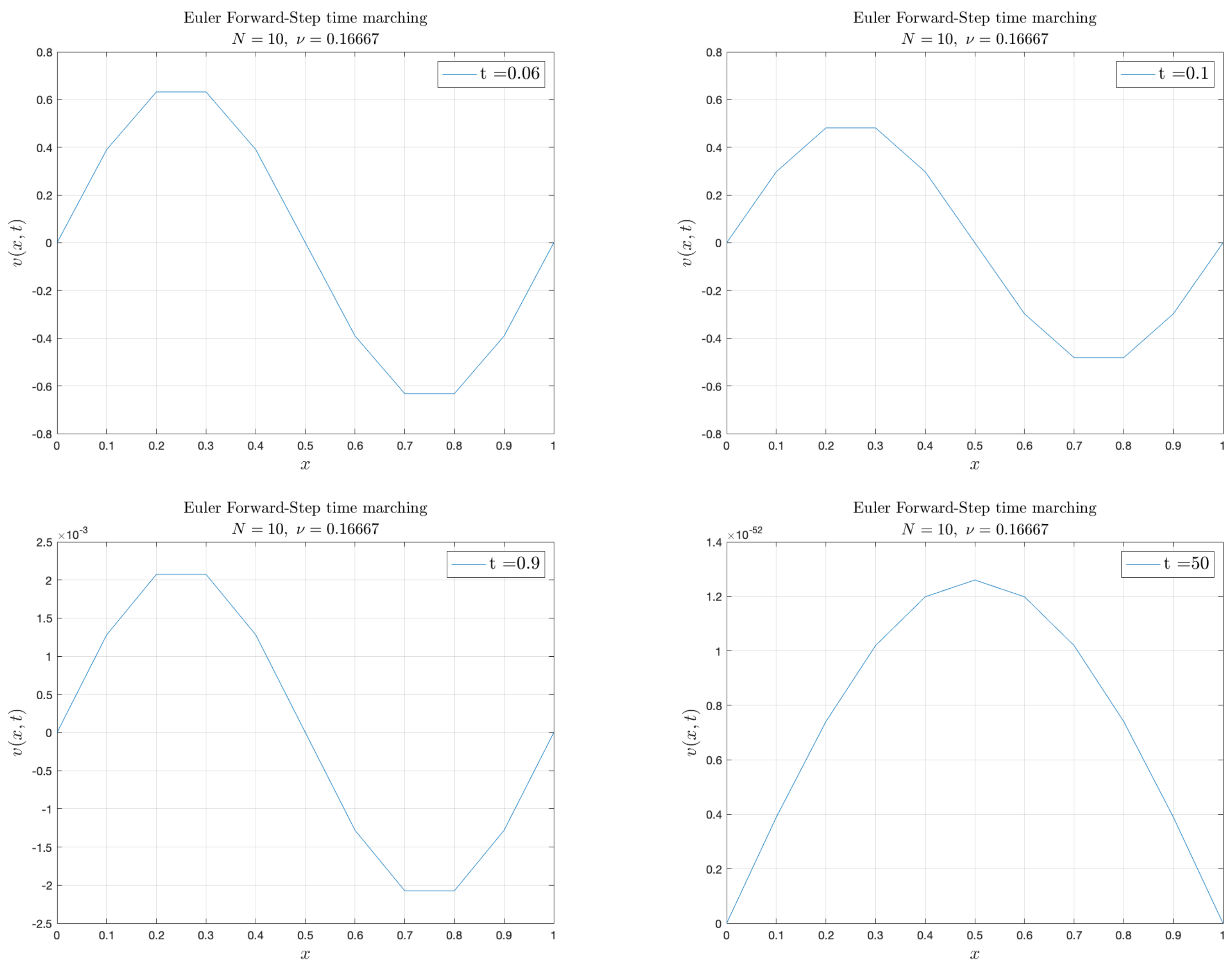

Figure 1: Solution for $N = 10,\ \Delta t = 0.02$, at $t = 0.06, t = 0.1, t = 0.9, t = 50$

2. (10 pts.) *Adopted from NPDE exercise 1.3.1:*
Solve the IBVP from problem (1) using leapfrog temporal integration, i.e.

$$D_{0t}v_j^n = \nu D_{+x}D_{-x}v_j^n.$$

Recall that $D_{0t}v_j^n = \frac{1}{2}(D_{+t} + D_{-t})v_j^n = \frac{v_j^{n+1} - v_j^{n-1}}{2\Delta t}$. For convenience, use the values form the exact solution at $t = \Delta t$ to get the leapfrog scheme started. Do only the part with $\Delta t = 0.02$. It is also suggested that if the results of this problem are not nice, do note spend the rest of your life on in.
To implement the leapfrog time scheme, it is important to know the solution at the second time-step along with the initial conditions. The exact solution to this PDE is found using

the dispersion relationship.

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

Applying the differences,

$$v_j^{n+1} = v_j^{n-1} + \frac{2\nu\Delta t}{\Delta x^2}\left(v_{j+1}^n - 2v_j^n + v_{j-1}^n\right)$$
$$v_j^{n+1} = v_j^{n-1} + 2r\left(v_{j+1}^n - 2v_j^n + v_{j-1}^n\right)$$

For the solution at first time-step, $\Delta t = 0.02$, the solution is

$$v(x,\Delta t) = \sin(2\pi x)e^{-4\pi^2 \nu \Delta t}$$

The code is attached for the leapfrog time-step method below.

Listing 2: Leap Frog time marching - Heat Equation (Q.2)

```
function [] = LeapFrog1D(N,dt,xlim1,xlim2,tlim1,tlim2)
% $Author: Vignesh Ramakrishnan$
% $RIN: 662028006$
% v_t = \nu v_xx, x \in (0,1), t > 0
% v(x,t=0) = sin(2\pi*x), v(0,t) = v(1,t) = 0
% \nu = 1/6;
% This function intends to use Finite Difference method to get a numerical
% solution to the heat equation described above. The compatability Boundary
% condtions are utilized to enforce Boundary conditions at the Boundary
% points. Leap Frog time stepping is used for time marching.
% Inputs: N     - Number of elements - describes spatial discretization
%         dt    - temporal discretization
%         xlim1 - left end of the spatial boundary
%         xlim2 - right end of the spatial boundary
%         tlim1 - start time (initial condition time)
%         tlim2 - end time of simulation
% Output: Prints the time evolution across the spatial grid for the heat
%         equation described above

%% Trial runs

% LeapFrog1D(10,0.02,0,1,0,0.1);

%% code


nu = 1/6;           % coefficient of heat transfer
a  = 0   ;          % time varying function at left boundary
b  = 0   ;          % time varying function at right boundary
dx = 1/N;           % dx -spatial discretization

r  = 2*nu*dt/dx^2; % r = CFL number

```

```
ng = 1   ;              % number of ghost points at one boundary
NP = N+1+2*ng;          % Total number of spatial points
ja = ng+1;              % xlim1's index number
jb = NP-ng;             % xlim2's index number

x  = (xlim1:dx:xlim2);
t  = (tlim1:dt:tlim2);

u  = zeros(length(t),NP);

% set initial conditions for the spatial grid
u(1, ja:jb)    = sin(2*pi*x);
u(1,ja)        = a;
u(1,jb)        = b;
u(1,1)         = 2*u(1,ja) - u(1,ja+1);
u(1,NP)        = 2*u(1,jb) - u(1,jb-1);

% set solution values at time = dt for the spatial grid
u(2, ja:jb)    = sin(2*pi*x)*exp(-nu*4*pi^2*dt);
u(2,ja)        = a;
u(2,jb)        = b;
u(2,1)         = 2*u(2,ja) - u(2,ja+1);
u(2,NP)        = 2*u(2,jb) - u(2,jb-1);

for i=3:length(t)
    for j = ja:jb
        u(i,j) = u(i-2,j) + r*(u(i-1,j+1)-2*u(i-1,j)+ u(i-1,j-1));
    end

    u(i,ng) = 2*u(i,ja) - u(i,ja+1);
    u(i,NP) = 2*u(i,jb) - u(i,jb-1);

end

Legend  = strcat('t = ',num2str(tlim2));
str1    = strcat('N = ', num2str(N));
str2    = strcat('\nu = ',num2str(nu));

str     = append('$',str1,',\ ',str2,'$');

figure
plot(x,u(end,ja:jb));
hold on;
grid on;
ylim([-1 1]);
xlabel('$x$','FontSize',16,'Interpreter','latex');
ylabel('$v(x,t)$','FontSize',16,'Interpreter','latex');
legend(Legend,'FontSize',16,'Interpreter','latex');
```

```
title('LeapFrog time marching',str,'Interpreter','latex','FontSize',14);


[X,T] = meshgrid(x,t);

figure
surf(X,T,u(:,ja:jb));
xlabel('$x$','FontSize',16,'Interpreter','latex');
ylabel('$t$','FontSize',16,'Interpreter','latex');
zlabel('$v(x,t)$','FontSize',16,'Interpreter','latex');
title('LeapFrog time marching','Surface Plot');
end
```

The solutions for this time marching approach is attached in Fig 2. The solutions for $t = 0.9$, and $t = 50$ have blown up and are unstable. Hence, those plots have not been attached.

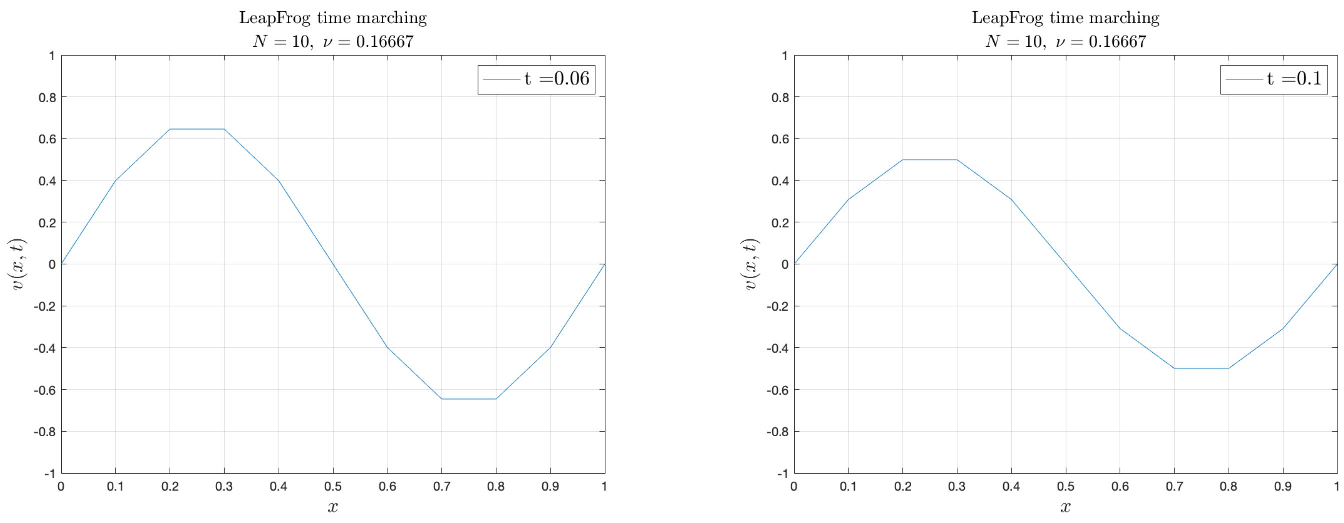

Figure 2: Solution for $N = 10, \Delta t = 0.02$, at $t = 0.06, t = 0.1, t = 0.9$, and $t = 50$

3. (15 pts.) Consider the heat equation

$$u_t = \nu u_{xx}, \qquad x \in (0,1), \qquad t > 0$$

with initial conditions

$$u(x, t = 0) = \sin\left(\frac{5\pi}{2}x\right),$$

and boundary conditions

$$u(x = 0, t) = 0$$
$$u_x(x = 1, t) = 0.$$

(a) Write a code to solve this problem using the usual second-order centered spatial discretization with forward Euler time integration, i.e.

$$D_{+t}v_j^n = \nu D_{+x}D_{-x}v_j^n$$

on the grid defined by $x_j = j\Delta x,\ j = 0, 1, \ldots, N,\ \Delta x = 1/N$, and the parameter $r = \nu\Delta t/\Delta x^2 = 0.4$. Ensure that your treatment of the boundary conditions are at least second-order accurate. Note that you may use a ghost cell to implement boundary conditions, but you are not required to do so.

There is one Initial condition, one Dirichlet Boundary condition and one Neumann Boundary condition present for this PDE. It is well posed and the method to handle these boundary conditions is described below.

$$u(x = 0, t) = 0 \tag{1}$$
$$u_x(x = 1, t) = 0 \tag{2}$$

Two ghost node points are added, one to the left of the left boundary (x_{-1}) and one to the right of the right boundary (x_{N+1}). Compatibility boundary condition is used for (1) and a central difference approach on the last boundary node is computed to approximate (2).

$$\begin{aligned} v_{-1}^n &= 2v_0^n - v_1^n \\ v_{N+1}^n &= v_{N-1}^n \\ v_j^{n+1} &= v_j^n + r\left(v_{j+1}^n - 2v_j^n + v_{j-1}^n\right),\ j = 0, 1, 2, 3, ..., N-1, N \\ r &= \frac{\nu\Delta t}{\Delta x^2} \end{aligned}$$

The code is modified to accept CFL number, r as an input and the time-step for time marching is calculated using r. Code is attached below in Listing 3.

Listing 3: Heat Equation (Q.3)

```
function [max_err,tEND] = HeatEqn2_1D(N,r,xlim1,xlim2,tspan)
% $Author: Vignesh Ramakrishnan$
% $RIN: 662028006$
% v_t = \nu v_xx, x \in (0,1), t > 0
% v(x,t=0) = sin(5\pi*x/2), v(0,t) = 0, v_x(1,t) = 0
% \nu = 1;
% This function intends to use Finite Difference method to get a
% numerical solution to the heat equation described above.
% Euler forward stepping is utilized to march in time.

% Inputs: N        - Number of elements
%         r        - CFL number
%         xlim1    - left end of the spatial boundary
%         xlim2    - right end of the spatial boundary
%         tspan    - total time period of integration
% Output: Prints the time evolution across the spatial grid for the heat
%         equation described above
%         max_err - maximum error between exact and numerical solution
%         tEND     - total time taken for simulation

```

```
%% Trial runs

% HeatEqn2_1D(40,0.4,0,1,1); - stable solution
% HeatEqn2_1D(40,0.6,0,1,1); - unstable solution

%% code

nu = 1;            % coefficient of heat transfer
a  = 0   ;         % time varying function at left boundary
dx = 1/N;          % dx -spatial discretization

dt  = r*dx^2/nu; % r = CFL number

ng = 1   ;         % number of ghost points at one boundary
NP = N+1+2*ng;     % Total number of spatial points
ja = ng+1;         % xlim1's index number
jb = NP-ng;        % xlim2's index number

x  = (xlim1:dx:xlim2);
t  = (0:dt:tspan);

u  = zeros(length(t),NP);

% set initial conditions for the spatial grid
u(1, ja:jb)    = sin(5*pi*x/2);
u(1,ja)        = a;
u(1,1)         = 2*u(1,ja) - u(1,ja+1); % compatability on left BC
u(1,NP)        = u(1,jb-1);             % Neumann on right BC

tSTART = tic;
for i=2:length(t)
    for j = ja:jb
        u(i,j) = u(i-1,j) + r*(u(i-1,j+1)-2*u(i-1,j)+ u(i-1,j-1));
    end

    u(i,ng) = 2*u(i,ja) - u(i,ja+1);
    u(i,NP) = u(i,jb-1);

end
tEND = toc(tSTART);

[X,T] = meshgrid(x,t);

figure
surf(X,T,u(:,ja:jb));
xlabel('$x$','FontSize',16,'Interpreter','latex');
ylabel('$t$','FontSize',16,'Interpreter','latex');
zlabel('$v(x,t)$','FontSize',16,'Interpreter','latex');
```

```
title('Numerical Solution','Surface Plot');

u_ex     = sin(5*pi*x/2).*exp...
               (-25*nu*pi^2*tspan/4); % exact solution at t = tspan

max_err  = max(u(end,ja:jb)-u_ex); % maximum error at t = tspan


end
```

(b) Setting $\nu = 1$, perform a grid refinement study using $N = 20, 40, 80, 160$ by computing the maximum errors in the approximation at a final time $t_f = .1$ for grid points $j = 0, 1, \ldots, N$ (i.e. do not compute errors in ghost cells if you use them). Produce a log-log plot showing the error as a function of Δx, and include a reference line indicating the expected rate of convergence.

The code attached above was simulated at multiple suggested values of $N = 20, 40, 80, 160$. The objective of this experiment is to verify the drop in the error between the actual solution and the numerical solution when the refinement of the spatial gird is increased. Actual solution is obtained using the dispersion analysis technique. The exact solution is given by:

$$u_{ex} = \sin\left(\frac{5\pi}{2}\right) e^{-\frac{25\nu\pi^2 t}{4}}$$
$$e_j^n = u_j^n - u_{ex}(x_j, t_n)$$

The maximum error among all the spatial points at final time $t_f = 0.1$ sec is plotted in Fig 3. From Fig 3, we can deduce that $\max \text{Error} \to 0$ as $\Delta x \to 0$.

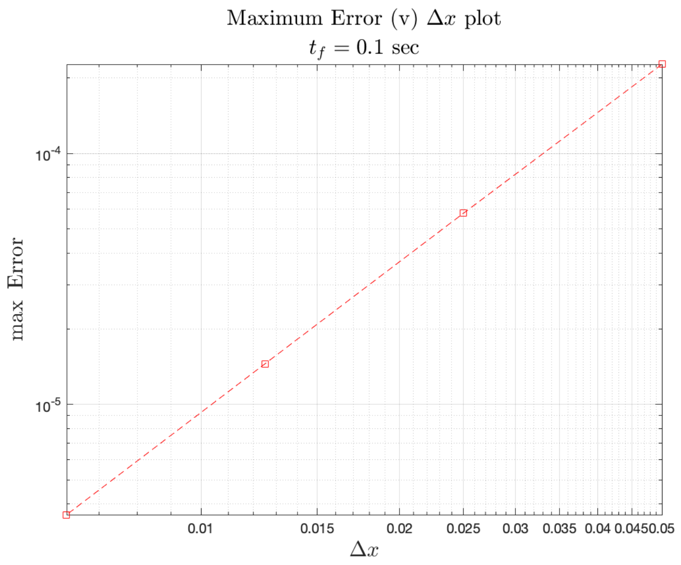

Figure 3: Maximum error (v) Grid refinement

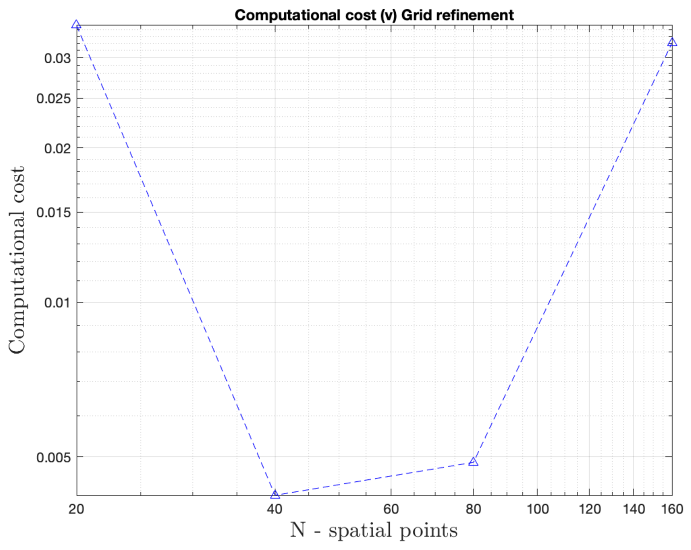

Figure 4: Computational cost (v) Grid refinement

(c) For each of the runs in part (b) determine the computational cost to obtain the approximation at the final time. For example you may use the MATLAB `tic` and `toc` commands. Discuss the scaling of the computational time with respect to the size of the grid. Produce a log-log plot showing the relation of the computational cost to the number of grid points, and include a reference line indicating the predicted growth in cost.

In the code attached in Listing 3., line 50. and line 60. documents the time taken to calculate the value of u at all the spatial points. The relationship between computational cost with increasing grid refinement is shown in Fig 4. The behavior is pretty strange for $N = 20$. It makes sense as to why the computational time is increasing as N is increased. With $N \uparrow$, there are more points on the spatial grid, hence the **for** loop runs for more number of iterations and hence, the rise in computational time is justifiable. But, the reasoning behind why the time taken for $N = 20$ is higher than $N = 40$ is strange and I am unable to figure it out.