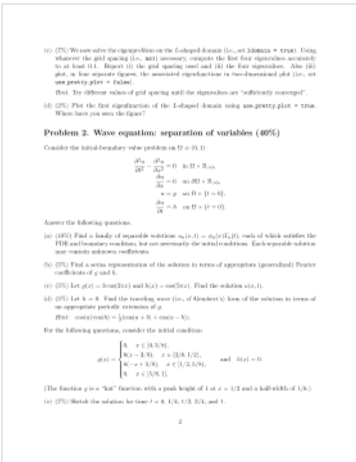
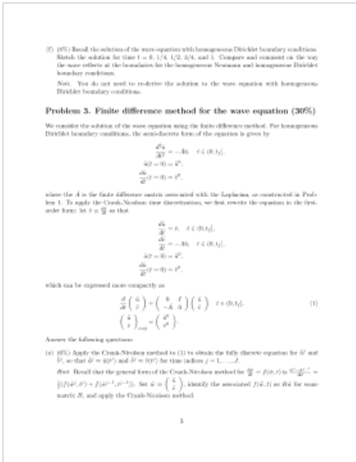
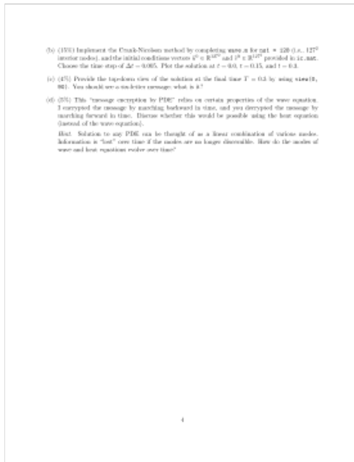
SOLUTIONS BELOW ↓
QUESTIONS







Aaryan Nagpal
1007792546
nagpalaa

ESC384 PDE's Assignment 4 Solutions

Q1)

a) The make_matrix.m, ldrum.m Functions completed. The code is on the right. I also have a ldrum_c.m code which was used to get the errors calculated in table below for part (b).

b) Error for eigenvalues 1, 4 for various grid spacing values (see table)

grid spacing

ns	hs	|λ_1-λ̄_1|	|λ_4-λ̄_4|
8	0.125	0.028493	0.22663
16	0.0625	0.0071335	0.056986
32	0.03125	0.001784	0.014267

used ldrum_c.m (rightmost code)

It appears that as grid spacing (hs) decreases by a factor of 2, the error reduces by a factor of 4 for both eigenvalues making the system second-order accurate.

c) i) hs = 1/256 → grid spacing

Chosen grid spacing = 1/256

used ldrum_m.

pretty_plot = false.

on L domain = true.

ii) First four eigenvalues

Number	Eigenvalue
1	6.21057471e+00
2	7.79653246e+00
3	8.88554285e+00
4	10.8662181e+00

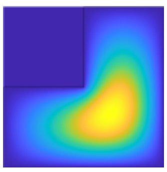
→ $\bar{\lambda}_1$ → $\bar{\lambda}_2$ → $\bar{\lambda}_3$ → $\bar{\lambda}_4$

First 4 eigenvalues

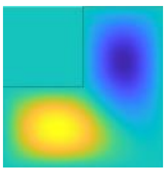
Plots of eigenfunctions for each eigenvalue.

iii)

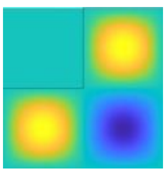
Eigenfucntion for eigenvalue 1



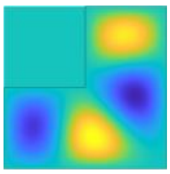
Eigenfucntion for eigenvalue 2



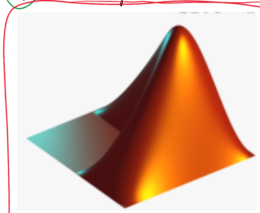
Eigenfucntion for eigenvalue 3



Eigenfucntion for eigenvalue 4



d) The first eigenfunction looks like the MATLAB (MathWorks) Logo.



Used ldrum.m

→ Mathworks (MATLAB) logo from Pretty Plot on L domain = true.

Q1a. Make_matrix code

```
function [Aint,interior] = make_matrix(xx,ldomain)
n1 = sqrt(size(xx,1));
% get grid spacing
%getting the spacing between 1d points assuming square grid by getting the
%recirpocal obtained by subtracting the last point from the first
h = 1/(n1-1);
% construct a matrix for the n1 x n1 grid
n2 = n1*n1;
A = sparse([],[],[],n2,n2);
for i = 2:n1-1 % interior nodes only
    for j = 2:n1-1 % interior nodes only
        % fill the FD equation for node (i,j)

        % find single-index value of self and neighbors
        %using matlab indexing to get the index value of the required
        %variables listed below
        indc = i  + n1*(j-1);  % center node: (i,j)
        indl = i-1 + n1*(j-1);  % left node: (i-1,j)
        indr = i+1 + n1*(j-1);  % right node: (i+1,j)
        indb = i  + n1*(j-2);  % bottom node: (i,j-1)
        indt = i  + n1*(j);      % top node: (i,j+1)

        % fill values
        %setting the values for specific indices
        A(indc,indc) = 4/h^2;     % center value
        A(indc,indl) = -1/h^2;    % left value
        A(indc,indr) = -1/h^2;    % right value
        A(indc,indb) = -1/h^2;    % bottom value
        A(indc,indt) = -1/h^2;    % top value
    end
end
% we now find the interior nodes; these are the unknowns of the problem
if (ldomain)
    % the following line needs to be completed to extract the interior nodes of
    % the L-shaped domain
    interior = find(((xx(:,1) > 0) & (xx(:,1) < 1) & (xx(:,2) > 0) & (xx(:,2) < 1)) ...
        & ((xx(:,1) > 1/2) | (xx(:,2) < 1/2)));
else
    % the following line extracts the interior nodes of a square
    interior = find((xx(:,1) > 0) & (xx(:,1) < 1) & (xx(:,2) > 0) & (xx(:,2) < 1));
end
% extract the equations and unknowns associated with interior nodes
Aint = A(interior,interior);
end
```

Q1c/d ldrum code for eigenvalues and pretty plot

```
function ldrum
% number of grid spacings in each direction (i.e., 1/h)
ns1 = 256;  %changed to 256
% specify the domain shape
ldomain = true;
% number of grid points in each direction
n1 = ns1+1;
n2 = n1*n1;
% generates 2d grid points
x1 = linspace(0,1,n1)';
[xm,ym] = ndgrid(x1,x1);
% concaternate grid points as single-index array
xx = [xm(:),ym(:)];
% construct a matrix for the n1 x n1 grid
[Aint,interior] = make_matrix(xx,ldomain);
% find the 10 smallest eigenvalues
neig = 10;
[V,D] = eigs(Aint,neig,'sm');
% display the sqrt of the neig eigenvalues
fprintf('%.8e\n',sqrt(diag(D)));
% eigenfunction to be plotted; change this to plot different eigenfunctions
ieig = 1;  %plotted 1,2,3,4 as needed
% plotting style
use_pretty_plot = true;
% plot the specified eigenfunctions
U = zeros(n2,1);
U(interior) = V(:,ieig);
figure(1), clf,
if (use_pretty_plot)
    pretty_plot(xm,ym,U);
else
    regular_plot(xm,ym,U);
end
end
function regular_plot(xm,ym,U)
% regular plotting
if (sum(U) < 0)
    U = -U;
end
surf(xm,ym,reshape(U,size(xm)));  hold on;
shading interp;
view(0,90);
axis equal;
axis off;
axis([0,1,0,1]);
% plot the border
if (sum(abs(U(xm < 1/2 & ym > 1/2))) < 0.1)
   plot([0,1,1,1/2,1/2,0,0],[0,0,1,1,1/2,1/2,0],'k-');
end
end
function pretty_plot(xm,ym,U)
% Pretty plot for the L-shaped domain problem.
% Copyright 1984-2007 The MathWorks, Inc.
% Modified by Masayuki Yano 2023
xm = 50*xm;
ym = 50*ym;
if (sum(U) < 0)
    U = -U;
end
U = U/max(abs(U));
if (min(U) < -0.01)
    Ushift = -0.4;
else
    Ushift = 0;
end
U = 40*(1/(max(U)-min(U))*(U-min(U)+Ushift));

figh = figure(1);
clf;
h = surf(xm,ym,reshape(U,size(xm))');
axis off;
logoax = axes('CameraPosition', [-193.4013 -265.1546  220.4819],...
    'CameraTarget',[26 26 10], ...
    'CameraUpVector',[0 0 1], ...
    'CameraViewAngle',9.5, ...
    'DataAspectRatio', [1 1 .9],...
    'Position',[0 0 1 1], ...
    'Visible','off', ...
    'XLim',[1 51], ...
    'YLim',[1 51], ...
    'ZLim',[-13 40], ...
    'parent',figh);
s = set(h, ...
    'EdgeColor','none', ...
    'FaceColor',[0.9 0.2 0.2], ...
    'FaceLighting','phong', ...
    'AmbientStrength',0.3, ...
    'DiffuseStrength',0.6, ...
    'Clipping','off',...
    'BackFaceLighting','lit', ...
    'SpecularStrength',1, ...
    'SpecularColorReflectance',1, ...
    'SpecularExponent',7, ...
    'parent',logoax);
l1 = light('Position',[40 100 20], ...
    'Style','local', ...
    'Color',[0 0.8 0.8], ...
    'parent',logoax);
l2 = light('Position',[.5 -1 .4], ...
    'Color',[0.8 0.8 0], ...
    'parent',logoax);
%z = zoom(figh);
%z.setAxes3DPanAndZoomStyle(logoax,'camera');
end
```

Q1b. Ldrum_c code for errors

```
function ldrum_c
ns = [1/8,1/16,1/32];
lambda1 = sqrt(2*pi^2);
lambda4 = sqrt(8*pi^2);
lambda1_error_values = zeros(length(ns),size(ns,1));
lambda4_error_values = zeros(length(ns),size(ns,1));

% number of grid spacings in each direction (i.e., 1/h)
hs = [8,16,32];
for i = 1:length(hs)
    ns1 = hs(i);
    disp(ns1)
    % specify the domain shape
    ldomain = false;

    % number of grid points in each direction
    n1 = ns1+1;
    n2 = n1*n1;

    % generates 2d grid points
    x1 = linspace(0,1,n1)';
    [xm,ym] = ndgrid(x1,x1);

    % concaternate grid points as single-index array
    xx = [xm(:),ym(:)];

    % construct a matrix for the n1 x n1 grid
    [Aint,interior] = make_matrix(xx,ldomain);

    % find the 10 smallest eigenvalues
    neig = 10;
    [V,D] = eigs(Aint,neig,'sm');

    % display the sqrt of the neig eigenvalues
    %fprintf('%.8e\n',sqrt(diag(D)));

    lambda1_error_values(i)=abs(lambda1-sqrt(D(1,1)));
    lambda4_error_values(i)=abs(lambda4-sqrt(D(4,4)));
end
T=table(hs',ns',lambda1_error_values, lambda4_error_values, 'VariableNames', {'ns','hs','|λ_1-λ̄_1|','|λ_4-λ̄_4|'});
disp(T)
% eigenfunction to be plotted; change this to plot different eigenfunctions
ieig = 4;
% plotting style
use_pretty_plot = false;
% plot the specified eigenfunctions
U = zeros(n2,1);
U(interior) = V(:,ieig);
figure(1), clf,
if (use_pretty_plot)
    pretty_plot(xm,ym,U);
else
    regular_plot(xm,ym,U);
end
end
function regular_plot(xm,ym,U)
% regular plotting
if (sum(U) < 0)
    U = -U;
end
surf(xm,ym,reshape(U,size(xm)));  hold on;
shading interp;
view(0,90);
axis equal;
axis off;
axis([0,1,0,1]);
% plot the border
if (sum(abs(U(xm < 1/2 & ym > 1/2))) < 0.1)
   plot([0,1,1,1/2,1/2,0,0],[0,0,1,1,1/2,1/2,0],'k-');
end
end
function pretty_plot(xm,ym,U)
% Pretty plot for the L-shaped domain problem.
% Copyright 1984-2007 The MathWorks, Inc.
% Modified by Masayuki Yano 2023
xm = 50*xm;
ym = 50*ym;
if (sum(U) < 0)
    U = -U;
end
U = U/max(abs(U));
if (min(U) < -0.01)
    Ushift = -0.4;
else
    Ushift = 0;
end
U = 40*(1/(max(U)-min(U))*(U-min(U)+Ushift));

figh = figure(1);
clf;
h = surf(xm,ym,reshape(U,size(xm))');
axis off;
logoax = axes('CameraPosition', [-193.4013 -265.1546  220.4819],...
    'CameraTarget',[26 26 10], ...
    'CameraUpVector',[0 0 1], ...
    'CameraViewAngle',9.5, ...
    'DataAspectRatio', [1 1 .9],...
    'Position',[0 0 1 1], ...
    'Visible','off', ...
    'XLim',[1 51], ...
    'YLim',[1 51], ...
    'ZLim',[-13 40], ...
    'parent',figh);
s = set(h, ...
    'EdgeColor','none', ...
    'FaceColor',[0.9 0.2 0.2], ...
    'FaceLighting','phong', ...
    'AmbientStrength',0.3, ...
    'DiffuseStrength',0.6, ...
    'Clipping','off',...
    'BackFaceLighting','lit', ...
    'SpecularStrength',1, ...
    'SpecularColorReflectance',1, ...
    'SpecularExponent',7, ...
    'parent',logoax);
l1 = light('Position',[40 100 20], ...
    'Style','local', ...
    'Color',[0 0.8 0.8], ...
    'parent',logoax);
l2 = light('Position',[.5 -1 .4], ...
    'Color',[0.8 0.8 0], ...
    'parent',logoax);
%z = zoom(figh);
%z.setAxes3DPanAndZoomStyle(logoax,'camera');
end
```

Q2) a) We want Family of sols: $u_n(x,t) = \phi_n(x)\,T_n(t)$ on $\Omega \equiv (0,1)$

Sub in PDE $\frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0$ to obtain

$$\Rightarrow \phi_n T_n'' - \phi_n'' T_n = 0 \quad \Rightarrow \quad \frac{T_n''}{T_n} = \frac{\phi_n''}{\phi_n} = -\alpha_n$$

Now, imposing the B.C, we have $(\Omega \equiv (0,1))$

$$\frac{\partial u_n(x=0,t)}{\partial x} = \phi_n'(x=0)\,T_n(t) \overset{\text{want}}{=} 0 \Rightarrow \phi_n'(0) = 0$$

$$\frac{\partial u_n(x=1,t)}{\partial x} = \phi_n'(x=1)\,T_n(t) \overset{\text{want}}{=} 0 \Rightarrow \phi_n(1) = 0$$

So, using the results above we can setup our spatial eigenproblem

i.e., $-\phi_n'' = \alpha_n \phi_n$ in $(0,1)$

∂x

So, using the results above we can setup our spatial eigenproblem

i.e., $-\phi_n'' = \alpha_n \phi_n$ in $(0,1)$

$\phi_n'(0) = \phi_n'(1) = 0$

This represents a strom-Liouville problem with

$p=1,\ q=0,\ w=\alpha_n^2,\ \alpha_1, \beta_1 = 0,\ \alpha_2 = -1, \beta_2 = 1$

Since its an S-L problem, eigenvalues ≥ 0, $\therefore \alpha_n \geq 0$

case: $\alpha_n = 0 \Rightarrow$ General Solution : $\phi_n(x) = A_n + B_n x$

checking B.C's : $\phi_n'(0) = B_n \overset{want}{=} 0 \Rightarrow B_n = 0$

$\phi_n'(1) = B_n \overset{want}{=} 0$ $\Big] \rightarrow \phi_n(x) = B_n$ (non-zero constant)

case: $\alpha_n > 0 \Rightarrow$ General solution : $\phi_n(x) = A_n\cos(\sqrt{\alpha_n}x) + B_n\sin(\sqrt{\alpha_n}x)$

Checking B.C.s : $\phi_n'(0) = -A_n\sqrt{\alpha_n}\,\cancel{\sin}^{0}(0) + B_n\sqrt{\alpha_n}\,\cancel{\cos}^{1}(0) = 0 \Rightarrow B_n\sqrt{\alpha_n} \overset{want}{=} 0$

$\Rightarrow B_n = 0$

$\phi_n'(1) = -A_n\sqrt{\alpha_n}\sin(\sqrt{\alpha_n}) + \cancel{B_n}^{0}\sqrt{\alpha_n}\cos(\sqrt{\alpha_n}) = 0$

$\Rightarrow -A_n\sqrt{\alpha_n}\sin(\sqrt{\alpha_n}) \overset{want}{=} 0$, $A_n \neq 0$ (trivial soln)

$\therefore \sin(\sqrt{\alpha_n}) = 0 \Rightarrow \sqrt{\alpha_n} = n\pi$

$\alpha_n = n^2\pi^2$, $n = 1,2,3\cdots$

$\Rightarrow$ solution ϕ_n given by $\phi_n(x) = \cos(n\pi x)$ as $\frac{d}{dx}u_n(x)$ at $x = 0,1 = 0$.

Also we have $\alpha_n = n^2\pi^2$ for $n = 1,2,3\ldots$

Now, solving the temporal problem we have,

$-T_n'' = \alpha_n T_n \Rightarrow -T_n'' = n^2\pi^2 T_n$

The solution to this expression exists in the form of

$$T_n(t) = \begin{cases} A_n + B_n t & , n = 0 \\ A_n\cos(n\pi t) + B_n\sin(n\pi t) & , n = 1,2,3\cdots \end{cases}$$

Thus, we can represent the seperable solution as follows

$$\boxed{u_n(x,t) = \begin{cases} A_n + B_n t & , n = 0 \\ \underbrace{A_n\cos(n\pi t)}_{\text{temporal}}\,\underbrace{\cos(n\pi x)}_{\text{spatial}} + B_n\sin(n\pi t)\cos(n\pi x) & , n = 1,2,3\cdots \end{cases}}$$

$\rightarrow$ solution for PDE, BC but not the IC's

b) To find series representation of solution with Fourier coefficients g, h we will impose initial conditions.

$$u(x,t=0) = A_0 + \sum_{n=1}^{\infty} A_n\cos(n\pi x) \overset{want}{=} g(x)$$

$$\frac{\partial u}{\partial t}(x,t=0) = B_0 + \sum_{n=1}^{\infty} B_n n\pi\cos(n\pi x) \overset{want}{=} h(x)$$

Now, we compare the expressions obtained to generalized Fourier series coefficients.

Fourier cosine series.

$$A_0 = \frac{\hat{g}_0}{2} = \int_0^1 g(x)\,dx$$

$$A_n = \hat{g}_n = 2\int_0^1 g(x)\cos(n\pi x)\,dx \quad , n = 1,2,3\cdots$$

$$B_0 = \frac{\hat{h}_0}{2} = \int_0^1 h(x)\,dx$$

$$B_n n\pi = \hat{h}_n \Rightarrow B_n = \frac{\hat{h}_n}{n\pi} \quad , \hat{h}_n = 2\int \sin(n\pi x)h(x)\,dx \quad , n = 1,2,3\cdots$$

Thus, we obtain the solution

$$\boxed{u(x,t) = \underbrace{\underbrace{\frac{\hat{g}_0}{2}}_{A_0} + \underbrace{\frac{\hat{h}_0 t}{2}}_{B_0 t}}_{n=0} + \sum_{n=1}^{\infty}\left(\underbrace{\hat{g}_n}_{A_n}\cos(n\pi t)\cos(n\pi x) + \underbrace{\frac{\hat{h}_n}{n\pi}}_{B_n}\sin(n\pi t)\cos(n\pi x)\right)}$$

$\rightsquigarrow$ series representation of the solution

c) Solution $u(x,t)$ for $g(x) = 3\cos(2\pi x)$, $h(x) = \cos(5\pi x)$

$$A_0 = \frac{\hat{g}_0}{2} = \int_0^1 3\cos(2\pi x)\,dx \Rightarrow A_0 = \frac{1}{2}\Big[6\pi\sin(2\pi x)\Big]_0^1 = 0$$

$$A_n = \hat{g}_n = 2\int_0^1 3\cos(2\pi x)\cos(n\pi x)\,dx$$

[invoking the orthogonality relationship]

$$\Rightarrow A_n = \begin{cases} 3, & n = 2 \\ 0, & n \neq 2 \end{cases}$$

$$B_0 = \frac{\hat{h}_0}{2} = \int_0^1 \cos(5\pi x)\,dx = B_0 = \frac{1}{2}\Big[5\pi\sin(5\pi x)\Big]_0^1 = 0$$

$$B_n = \frac{\hat{h}_n}{n\pi} = 2\int_0^1 \cos(5\pi x)\cos(n\pi x)\,dx$$

[Again, invoking orthogonality]

$$\Rightarrow B_n = \begin{cases} 1, & n=5 \\ 0, & n \neq 5 \end{cases}$$

Substituting these coefficients into $u(x,t)$ we found in part (b), we get

$$\boxed{u(x,t) = 3\cos(2\pi t)\cos(2\pi x) + \frac{1}{5\pi}\sin(5\pi t)\cos(5\pi x)}$$ → solution $u(x,t)$ for $g(x)$, $f(x)$ given

$\hat{g}_n = A_n \rightarrow n=2$ $\qquad$ $B_n = \frac{\hat{h}_n}{n\pi}$, $n=5$

(d) For $h=0$, Finding the travelling wave D'alemberts form of solution with appropriate periodic extension.

$g \neq, h=0 \Rightarrow \hat{h}_n = 0$

$$u(x,t) = \frac{1}{2}\hat{g}_0 + \sum_{n=1}^{\infty}\left(\hat{g}_n\cos(n\pi t)\cos(n\pi x)\right)$$ (from pt (b), setting $\hat{h}_n = 0$)

$$= \frac{1}{2}\hat{g}_0 + \frac{1}{2}\sum_{n=1}^{\infty}\hat{g}_n\left(\cos(n\pi(x-t)) + \cos(n\pi(x+t))\right)$$ [using trig identities as explained in lecture and hint]

Since $\hat{g}_n$ are Fourier cosine coefficients of g,

$$\sum_{n=1}^{\infty}\hat{g}_n\cos(n\pi\xi) = \underbrace{g^{e,p}(\xi)}_{\text{even-periodic extension of } g} \quad \forall \xi \in \mathbb{R}$$

$$u(x,t) = \frac{1}{2}\hat{g}_0 + \frac{1}{2}\sum_{n=1}^{\infty}\hat{g}_n\left(\cos(n\pi(x-t)) + \cos(n\pi(x+t))\right)$$

$$= \boxed{\frac{1}{2}\left(g^{e,p}(x-t) + g^{e,p}(x+t)\right)}$$ → solution in terms of periodic extension of g, $h=0$.

(e) Sketches for varying time values for given initial condition.

given: $g(x)$ is a hat function, peak height = 1 at $x=\frac{1}{2}$, width = $\frac{1}{4}$

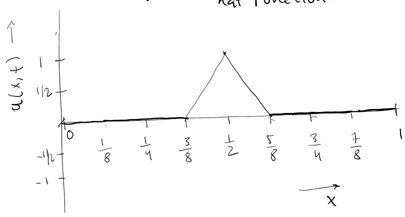
$h(x) = 0$.

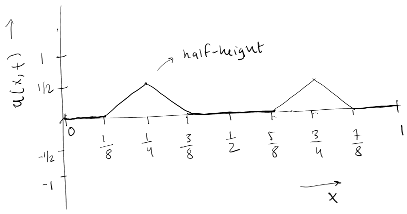
Sketching $u(x,t)$ (Neumann B.C), reflection without sign change

at $t=0$

using initial conditions to sketch hat function.

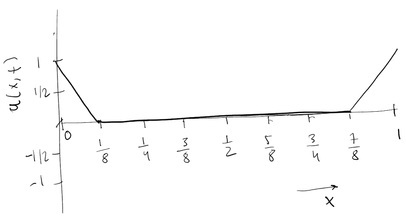
at $t = 1/4$

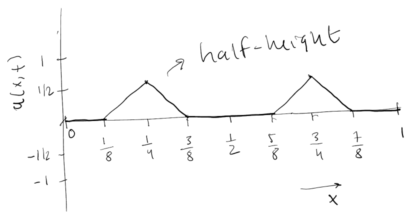




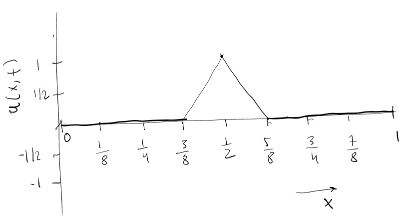
at $t = 1/2$

at $t = 3/4$





at $t = 1$



(f) Sketching the solution with Dirichlet B.C. at varying times

given: $g(x)$ is a hat function, peak height = 1 at $x=\frac{1}{2}$, width = $\frac{1}{4}$

(f) Sketching the solution with Dirichlet B.C. at varying times

given: $g(x)$ is a hat function, peak height $=1$ at $x=\frac{1}{2}$, width $=\frac{1}{4}$

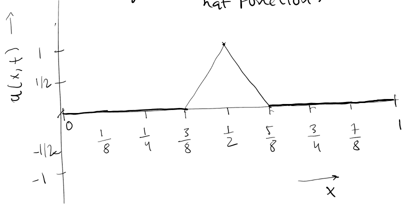
$h(x)=0$.

The main difference between wave reflection between (Neumann pt(e)) and Dirichlet B.C. (pt (f)) is the sign-change after reflection at the boundary in Dirichlet case. This is prominently seen at $t=1/2$, $t=3/4$, $t=1$ in the sketches below as compared to pt (e).

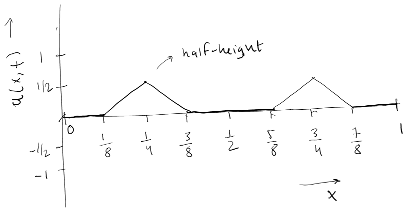
Sketching $u(x,t)$ (reflection on boundary with sign change) (Dirichlet B.C.)

at $t=0$

using initial conditions to sketch hat function.

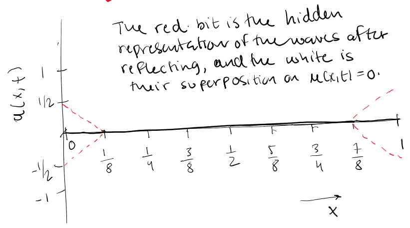


at $t=1/4$

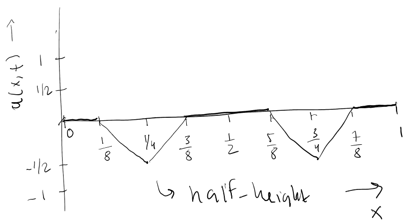


at $t=1/2$ (zero-everywhere)

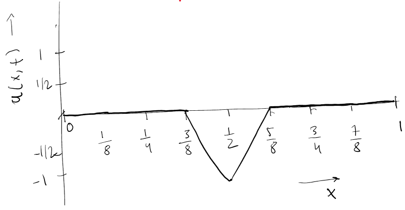
The red bit is the hidden representation of the waves after reflecting, and the white is their superposition as $u(x,t)=0$.



at $t=3/4$



at $t=1$



Q3 (a) Applying Crank-Nicolson on ① to get fully discrete $\hat{\tilde{w}}, \hat{\tilde{v}}^j$

$$\frac{d}{dt}\begin{pmatrix}\hat{u}\\ \hat{v}\end{pmatrix} = \begin{pmatrix}0 & I\\ -\hat{A} & 0\end{pmatrix}\begin{pmatrix}\hat{u}\\ \hat{v}\end{pmatrix} \quad - ①$$

$\Rightarrow \frac{d}{dt}\hat{\tilde{w}} = B\hat{\tilde{w}}$ to obtain $\hat{\tilde{w}}^j$, B.

$\Rightarrow$ given in hint: $\frac{d\hat{\tilde{w}}}{dt} = F(\tilde{w},t) = \frac{\hat{\tilde{w}}^j - \hat{\tilde{w}}^{j-1}}{\Delta t} = \frac{1}{2}\left(F(\hat{\tilde{w}}^j, t^j) + F(\hat{\tilde{w}}^{j-1}, t^{j-1})\right)$

→ Crank Nicolson

Also from lecture, we know

$$\frac{d\hat{u}}{dt} + \hat{A}\hat{u}(t) = \underbrace{\hat{f}(t) + \hat{h}(t)}_{=0}$$

expressing in our case as

$$\frac{d\hat{\tilde{w}}}{dt} = B\hat{\tilde{w}} \text{ as } \hat{A}\hat{u}(t) = -B\hat{\tilde{w}}$$

setting $\tilde{w} = \begin{pmatrix}\hat{u}\\ \hat{v}\end{pmatrix}$, and using $B\tilde{w}$ to find associated $F(\tilde{w},t)$.

$$\Rightarrow \frac{\hat{\tilde{w}}^j - \hat{\tilde{w}}^{j-1}}{\Delta t} = \frac{1}{2}\left(B\hat{\tilde{w}}^j + B\hat{\tilde{w}}^{j-1}\right) \Rightarrow$$

isolating $\hat{\tilde{w}}^j$ we end up with

$$\hat{\tilde{w}}^j = \frac{\Delta t}{2}B\hat{\tilde{w}}^j + \frac{\Delta t}{2}B\hat{\tilde{w}}^{j-1} + \hat{\tilde{w}}^{j-1}$$

, now we can use the properties of identity matrices to express this as

$$\Rightarrow \hat{\tilde{w}}^j - \frac{\Delta t}{2}B\hat{\tilde{w}}^j = \frac{\Delta t}{2}B\hat{\tilde{w}}^{j-1} + \hat{\tilde{w}}^{j-1} \Rightarrow \hat{\tilde{w}}^j\underbrace{\left(I - \frac{\Delta t}{2}B\right)}_{\text{C matrix}} = \hat{\tilde{w}}^{j-1}\underbrace{\left(\frac{\Delta t}{2}B + I\right)}_{\text{D matrix}}$$

$\Rightarrow$ getting the update value

$$\boxed{\hat{\tilde{w}}^j = \left(I - \frac{\Delta t}{2}B\right)^{-1}\left(\frac{\Delta t}{2}B + I\right)\hat{\tilde{w}}^{j-1} \quad \text{where } B\hat{\tilde{w}} = \frac{d\hat{\tilde{w}}}{dt}}$$

From ①, we can therefore say by comparison that $B = \begin{pmatrix}0 & I\\ -\hat{A} & 0\end{pmatrix}$

using in pt(b) code

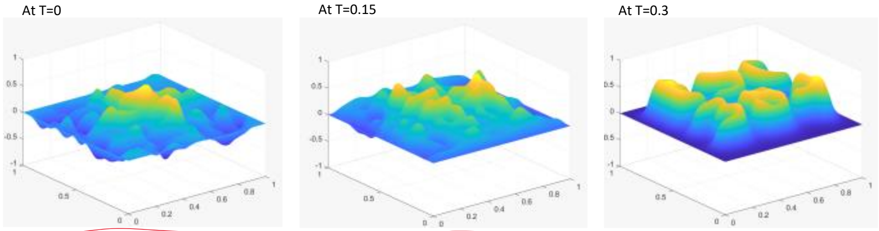
↳ fully discrete eqn for $\hat{\tilde{w}}, \hat{\tilde{v}}^j$ in terms of $\hat{\tilde{w}}^j$ and the B matrix.

(b) Plots for solution to wave-eqn at $t=0, 0.15, 0.3$ after implementing Crank-Nicolson as obtained in part (a). Using I.C. $\hat{u}^0, \hat{v}^0$ as given and $\Delta t = 0.005$. Plotted using wave.m implementation shown in the code on the right.

```
Q3b/c wave code to generate plots of solutions at timesteps and get top-view plot
function wave
% number of grid spacings in each direction (i.e., 1/h)
ns1 = 128;
% number of grid points in each direction
n1 = ns1 + 1;
n2 = n1 * n1;
% generates 2d grid points
x1 = linspace(0, 1, n1)';
[xm, ym] = ndgrid(x1, x1);
% concaternate grid points as single-index array
xx = [xm(:), ym(:)];
% construct the Laplacian matrix for the n1 x n1 square grid
[Aint, interior] = make_matrix(xx, false);
n = size(Aint, 1);
```

b) Plots for solution to wave-eqn at $t = 0, 0.15, 0.3$ after implementing Crank-Nicolson as obtained in part (a). Using I.C. $\vec{u}^0, \vec{v}^0$ as given and $\Delta t = 0.005$. Plotted using wave.m implementation shown in the code on the right.

At T=0   At T=0.15   At T=0.3



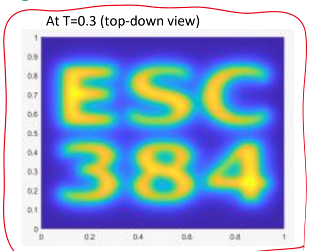
```
ns1 = 128;
% number of grid points in each direction
n1 = ns1 + 1;
n2 = n1 * n1;
% generates 2d grid points
x1 = linspace(0, 1, n1)';
[xm, ym] = ndgrid(x1, x1);
% concaternate grid points as single-index array
xx = [xm(:), ym(:)];
% construct the Laplacian matrix for the n1 x n1 square grid
[Aint, interior] = make_matrix(xx, false);
n = size(Aint, 1);
% load initial condition
d = load('ic.mat');
W = [d.U0; d.V0]; % concatenate displacement and velocity as a single vector
% useful matrices
Iint = speye(size(Aint)); % sparse identity matrix for interior points
Zint = sparse([], [], [], n, n); % sparse zero matrix for interior points
% set time step
dt = 0.005;
%creating the identity matrix with dimensions to match W (a vector with
%u,v)
Iint2 = speye(size([Zint Iint; -Aint Zint]));
%creating the B matrix using the expression found for it in Q3 part (a)
B = [Zint Iint; -Aint Zint];
%creating the timestamps at which we want to observe the plotted solution
timestamps =[0,0.15,0.3];
% time marching
for it = 1:n1
    % Crank-Nicolson update
    W = (Iint2-dt/2.*B)\((dt/2.*B+Iint2)*W); %the update the uses the identity matrix with
    %dimensions equivalent to W and the B matrix whihc was setup as
    %earlier, the expression for W was also obtained in Q3 part (a)

    Up = zeros(n2, 1); % full vector including boundary nodes
    Up(interior) = W(1:n); % copy displacement part of the solution

    % plot the solution
    %running through the selected timestamps using a for loop and plotting
    %a figure for the solution at each timestamp
    for i = 1:length(timestamps)
        if ((it-1)*dt == timestamps(i))
            %creating a condition where it also plots the top-view incase the
            %timestamp is 0.3 for Q3 part (c)
            if (timestamps(i)==0.3)
                figure;
                plot_wave(xm, ym, Up);
                drawnow();
                figure;
                plot_wave(xm, ym, Up);
                view(0,90);
                drawnow();
            else
                figure;
                plot_wave(xm, ym, Up);
                drawnow();
            end
        end
    end
end
end
function plot_wave(xm, ym, U)
    clf,
    surf(xm, ym, reshape(U, size(xm)));  hold on;
    shading interp;
    axis([0, 1, 0, 1, -1, 1]);
end
```

c) The six letter message as seen below is ESC384, the course code for this class (PDEs)

At T=0.3 (top-down view)



used wave.m to obtain where if condition was run to get top view using view(0,90)

d) No, it would not be possible to use a heat equation to do the "message encryption as it depends on the linear combination of modes as explained in the hint. As the modes of the heat equation decay exponentially, and they get smoothened out as time passes, it makes the modes less discernable and thus the message would get faded out after a few timesteps, making it not readable.

In the wave equation, the modes have a lack of smoothness and no decay, allowing the message to be encrypted/decrypted even after the passage of time steps. The lack of smoothness is dependant on the initial condition, but it is never completely smooth (infinitely differentiable) like the heat equation, further explaining why a heat eqn can not be used.

→ due to $\cos(n\pi t)$ term