Eigenvalues by Power Method and Inverse Power Method

Elvis Rodas, Patrik Wall, Jamel Hudson

I. Introduction

The goal of this lab is to find the largest and smallest eigenvalue of a given unit-length stationary wave function with a lose end. The system,

$$\begin{cases} -u'' = \lambda u \\ u(0) = 0 \\ u'(1) = 0 \end{cases}$$

is transformed with Finite Element Method to an ordinary eigenvalue problem,

$$Ac = \lambda c \tag{1}$$

where λ is the eigenvalue and $A = M^{-1}K$, n-by-n matrix[1] and c n-by-1 vector. The eigenvalues is analytically determined by finding the roots for the characteristic polynomial, derived from (1) as,

$$p(\lambda) = det(A - \lambda I) = 0 \tag{2}$$

where I is the identity matrix. With known eigenvalues, the corresponding eigenvectors can be determined by solving

$$\det(A - \lambda_j I)\hat{x}_j = 0, \quad j = 1, 2, ...n. \tag{3}$$

For $n >> 1$ it is close to impossible to solve (2) analytically and one has to resort to computational methods.

II. Eigenvalues

A. The Power Method

The Power Method is used in this laboratory which is an iterative method to find the Dominant Eigenvalue, which is defined as

$$|\lambda_{max}| > |\lambda_j|, \quad j = 1, 2, ...n. \tag{4}$$

Let $\hat{z}_0 \in \mathbb{R}^n$ a random non-zero vector, in this laboratory it is instructed to initialize $\hat{z}_0$ with all entries equal. There exists a linear combination of the eigenvetors such that

$$\hat{z}_0 = \sum_{j=1}^{n} c_j \hat{x}_j \tag{5}$$

since eigenvectors are, a priori, linearly independent.

By continuous left-side multiplication of A on (5) the following expression is derived,

$$A\hat{z}_0 = c_1 A\hat{x}_1 + c_2 A\hat{x}_2 + ... + c_n \hat{x}_n \overset{(1)}{\iff}$$

[1]M and K are defined in the instructions

$$A\hat{z}_0 = c_1\lambda_1\hat{x}_1 + c_2\lambda_2\hat{x}_2 + ... + c_n\lambda_n\hat{x}_n \implies ... \implies$$

$$A^k\hat{z}_0 = \lambda_1^k\left(c_1\hat{x}_1 + c_2\left(\frac{\lambda_2}{\lambda_1}\right)^k\hat{x}_2 + ... + c_n\left(\frac{\lambda_n}{\lambda_1}\right)^k\hat{x}_n\right). \tag{6}$$

Let λ_1 denote The Dominant Eigenvalue, thus by the limit function,

$$\lim_{k\to\infty} c_p\left(\frac{\lambda_p}{\lambda_1}\right)^k\hat{x}_p \longrightarrow 0, \quad p = 2, 3, ...n,$$

expression (6) converges geometrically to

$$A^k\hat{z}_0 \approx \lambda_1^k c_1\hat{x}_1 \overset{(1)}{\iff}$$

$$\therefore \quad \hat{z}_0 \approx c_1\hat{x}_1, \tag{7}$$

with ratio

$$\left|\frac{\lambda_1}{\lambda_2}\right| \tag{8}$$

where λ_2 denotes the second Dominant Eigenvalue. For computational reasons $\hat{z_k}$ is normalised with the eucleadian norm. The power method algorithm

Algorithm 1 Power Method

```
1: $z_0$ = initial guess
2: while $|\mu_{k+1} - \mu_1| < tolerance$ do
3:     $y_k = z_k/||z_k||_2$
4:     $z_{k+1} = Ay_k$
5:     $\mu_k = y_k^H z_{k+1}$
6: end while
```

where $\lambda_1 \approx \mu_k$ when convergence is reached.

B. Inverse Power Iteration

Inverse power iteration is can be used to approximate the smallest eigenvalue of the matrix. From (1),

$$A^{-1}\hat{x} = \frac{1}{\lambda}\hat{x} \tag{9}$$

is formed. Let $A^{-1} = A - sI$, a shifted version. Expression (9)

$$(A - sI)\hat{x} = (\lambda - s)\hat{x} \tag{10}$$

$\therefore$ A^{-1} has the same eigenvalues as A. By changing row 5 in Algorithm 1 to its inverse μ_k converges to the smallest eigenvalue. The Inverse Power algorithm

Algorithm 2 Inverse Power Method

```
1: $z_0$ = initial guess
2: while $|\mu_{k+1} - \mu_1| < tolerance$ do
3:     $y_k = z_k/||z_k||_2$
4:     solve $Az_{k+1} = y_k$
5:     $\mu_k = 1/(y_k^H z_{k+1})$
6: end while
```

where $\mu_k \approx \lambda_1$ when convergence is reached. λ_1 is now the smallest eigenvalue.

The speed of convergence for the inverse iteration method for a given matrix is proportional to

$$\mathcal{O}\left(\left|\frac{\mu - \lambda_{\text{closest to }\mu}}{\mu - \lambda_{\text{second closest to }\mu}}\right|\right) \tag{11}$$

where μ is the smallest eigenvalue by magnitude.

III. Result

Figure 1 shows a visual solution of the random matrix

$$A = \begin{bmatrix} 5 & 3 \\ 1 & 4 \end{bmatrix}$$

which has the eigenvalues $\lambda = [6.3028, 2.6972]$ and the eigenvetors $[0.9172, 0.3983]^T$ and $[-0.7933, 0.6089]^T$ respektively. The iterative metohds shows a clear convergence to the real eigenvalues.

A. Power Method

A MATLAB function for computing the largest eigenvalue and the corresponding eigenvector using the power method was constructed. The code is presented in Appendix 3.2 Task 1: code.

The largest eigenvalues for matrix dimensions $n = 20, 40, 80$ are presented in Table 1. The number of computations required increased with with higher dimensions whereas the error in largest eigenvalue approximation decreased.

TABLE I
APPROXIMATED LARGEST EIGENVALUES

n	eigenvalue	error	number of iterations
20	4.7780e+03	0.0973	134
40	1.9178e+04	0.0254	581
80	7.6778e+04	0.0057	2307

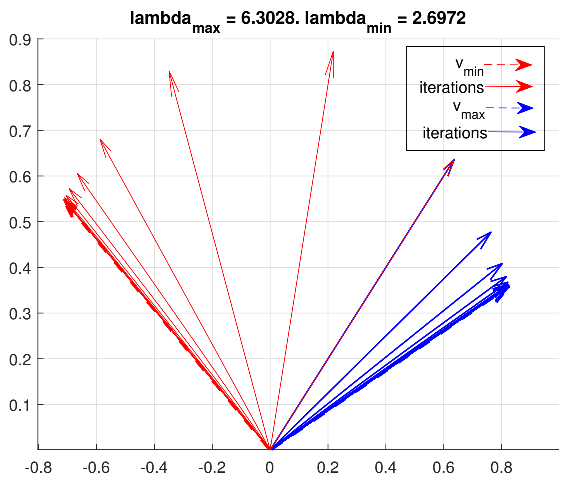


Fig. 1. Approximation of eigenvectors for each iteration. The real eigenvetors are $[0.9172, 0.3983]^T$ and $[-0.7933, 0.6089]^T$. The power method and inverse power method were used to compute the largest and smallest eigenvlaues and its respective eigenvectors.

B. Inverse Power Method

A MATLAB function for computing the smallest eigenvalue and the corresponding eigenvector using the Inverse iteration method was constructed. The code is presented in Appendix 3.2 Task 2: code.

The smallest eigenvalues for dimension n=20, 40, 80 are presented in Table II. The number of computations required decreased with higher dimensions whereas the error in smallest eigenvalue approximation increased.

TABLE II
APPROXIMATED SMALLEST EIGENVALUES

n	eigenvalue	error	number of iterations
20	2.4687	5.1460e-06	5
40	2.4677	2.6711e-06	5
80	2.4675	1.1569e-05	4

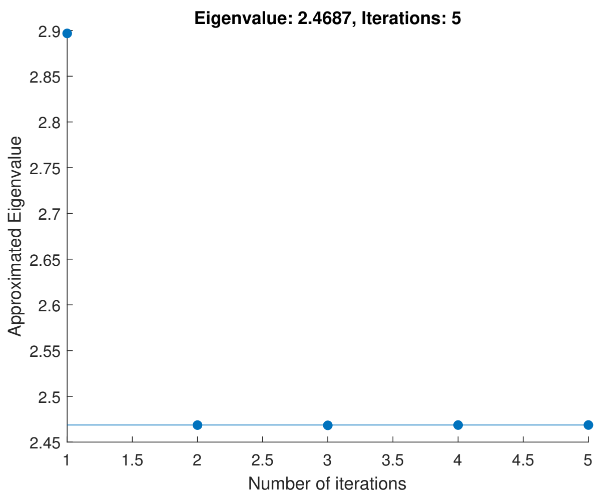


Fig. 2. Approximation of eigenvalues for the 20x20 matrix

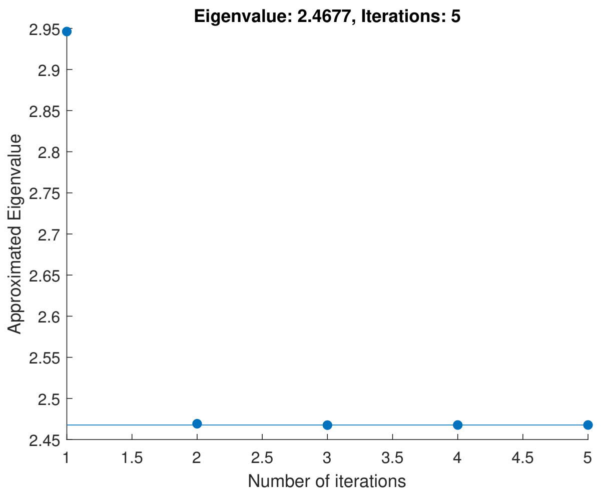


Fig. 3. Approximation of eigenvalues for the 40x40 matrix, error=

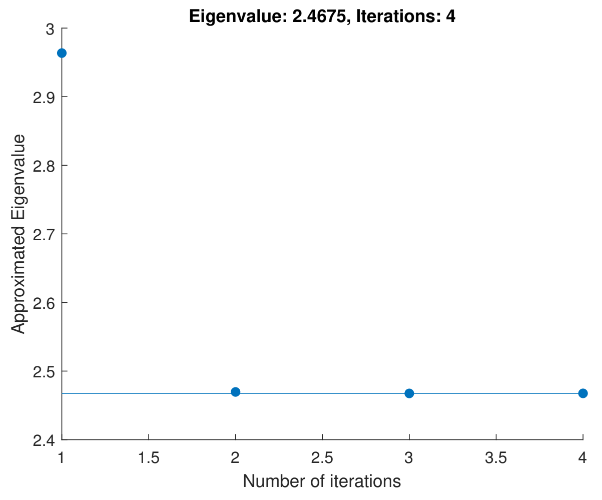


Fig. 4. Approximation of eigenvalues for the 60x60 matrix

IV. Discussion

The number of iterations required for the power method with is seen to increase with increasing matrix dimension n (table 1). The convergence speed for the power method, for this case, is then inversely proportional to matrix dimension, n. However, for the inverse power method the opposite relationship is found; as the matrix dimension is increased the number of iterations required to reach the error tolerance decreases and it can be said that the convergence speed of the inverse power method is proportional to the matrix dimension.

V. Appendix

A. Main code

```
% ========================================== %
% MiniProject 1: Eigenvalues by Power Method %
% ==========================================%

close all; clear all;

n = 20; % Number of dimensions.
h = 1/n;
e =ones(n,1);
K = spdiags([-1*e 2*e -1*e],-1:1,n,n)*1/h;
M = spdiags([e 4.*e e],-1:1,n,n)*h/6;

A = M\K;

% Calculating eig values and vectors.
[eigVal, eigVect,numiter] = eig_power(A, false);
[eigVal2, eigVect2,numiter2]=eig_ipower(A, false);
```

B. Task 1: Power method function

```
function [eigVal, eigVect,numiter] = eig_power(A, plot)
% ================================================
% The function computes the eigenvectors of the
% square matrix A using the Power Method.
%
% input:    A - Square matrix
%           plot - boolean.
%
% output:   v   - The approx eigenvectors of A
%           numiter - Number of iterations.
% ============================================

tol = 1e-4; %   tolernace: 10^-4.
err = 1;    %   |delta lambda| < tol
z = ones(length(A),1); %    Initial vector
eigVal = 0; eigValPrev = 0; numiter = 0;
vectArray = []; % Array of eigenvetors iterated.


while err >= tol
    y = z/norm(z) ;
    z = A*y;
    eigVal = y'*z;
    err = abs(abs(eigVal) - abs(eigValPrev));
    eigValPrev = eigVal;
    numiter = numiter +1;
    vectArray(:,end+1) = y;
end
eigVect = z/norm(z);


% MATLABs solution
[eigVectsM, eigValsM] = eigs(A);
[~, idx]= max(max(abs(eigValsM)));
maxEigVect = eigVectsM(:,idx);
% vect of max eig. value.


if plot && size(A) <= 3
    % Plotting: Eigenvector given by MATLAB.
    hold on;
    if (length(A) == 2) % 2D
    quiver(0, 0, maxEigVect(1), maxEigVect(2),...
    '--blue', 'LineWidth',2); grid on;
    elseif(length(A) == 3) % 3D
    quiver3(0,0,0,maxEigVect(1), maxEigVect(2), ...
    maxEigVect(3),'--blue', 'LineWidth',2);
    end
```

```matlab

    % Plotting: Eigenvectors for each iteration.
    if (length(A) == 2) % 2D
        for i = 1:length(vectArray)
        v = vectArray(:,i);
        quiver(0, 0,v(1), v(2),'b','LineWidth',1);
        end

    elseif (length(A) == 3) %3D
        for i = 1:length(vectArray)
        v = vectArray(:,i);
quiver3(0,0,0,v(1),v(2),v(3),'b',LineWidth',1);
        end
    end
    hold off;
end
end
```

C. Task 2: Invere power method function

```matlab
function [eigVal, eigVect,numiter] = eig_ipower(A, plot)
% ==============================================
% The function computes the eigenvectors of the
% square matrix A using the Power Method.
%
% input: A       - Square matrix
%        plot    - boolean
% output: eigVal - The minimal eigenvalue of A
%         eigVect - The corresponding eigenvector.
%         numiter - Number of iterations.
% ==============================================

tol = 1e-4; %   tolernace: 10^-4.
err = 1;    %   |delta lambda| < tol
z = ones(length(A),1); %    Initial vector
eigVal = 0; eigValPrev = 0; numiter = 0;
vectArray = []; % Array of eigenvetors iterated.


while err >= tol
    y = z/norm(z);
    z = A\y;
    eigVal = 1/(y'*z);
    err = abs(abs(eigVal) - abs(eigValPrev));
    eigValPrev = eigVal;
    numiter = numiter +1;
    vectArray(:,end+1) = y;
end
eigVect = z/norm(z);


% MATLABs solution
[eigVectsM, eigValsM] = eigs(A);
[~, idx]= min(max(abs(eigValsM)));
maxEigVect = eigVectsM(:,idx);

% Plotting only 2D or 3D.
if plot && length(A) <= 3
    % Plotting: Eigenvector given by MATLAB.
    hold on;
    if (length(A) == 2)
    quiver(0, 0, maxEigVect(1), maxEigVect(2),...
    '--red', 'LineWidth',2); grid on;
    elseif(length(A) == 3)
    quiver3(0, 0, 0, abs(maxEigVect(1)),...
    abs(maxEigVect(2)),abs(maxEigVect(3)),...
        '--r', 'LineWidth',2);
    end

    % Plotting: Eigenvector of iterations
    if (length(A) == 2 )
        for v = vectArray
        quiver(0, 0, v(1), v(2),'r');
        end

    elseif (length(A) == 3)
        for v = vectArray
        quiver3(0, 0, 0, v(1), v(2), v(3), 'r');
        end
    end
    hold off;
end
end
```

Miniprojekt 2

Elvis Rodas, Patrik Wall, Jamel Hudson

***Abstract*—workout 2 in Scientific Computing III.**

I. Task 1: Steady state 1D heat equation

The stationary state heat equation is given by

$$-k\frac{d^2T}{dx^2} = Q + h(T_{ext} - T) \tag{1}$$

where k is the coefficient of heat conduction, h is the convective heat transfer, Q is the heat source and T_{ext} is the external temperature.

In this we assume $k = 1, Q = 0, h = 1$ and $T_{ext} = 20$. Equation (1) becomes

$$-\frac{d^2T}{dx^2} = T_{ext} - T, \quad T(0) = 40, T(10) = 200 \tag{2}$$

A. Analytical solution

Equation (2) can be solved analytical by separation of homogen and inhomogen solutions. The homogen oden is given by

$$-\frac{d^2T}{dx^2} + T = 0 \tag{3}$$

The characteristic equation of (3) gives that the solution is

$$T_h(x) = Ae^x + Be^{-x} \tag{4}$$

The inhomogen solution is $T_p(x) = 20$.

The boundary conditions yields that $A = (20-180e^{10})/(1-e^{20})$ and $B = (180e^{10} - 20e^{20})/(1 - e^{20})$. The analytical solution to (2) is

$$T(x) = \left(\frac{20 - 180e^{10}}{1 - e^{20}}\right)e^x + \left(\frac{180e^{10} - 20e^{20}}{1 - e^{20}}\right)e^{-x} + 20 \tag{5}$$

B. FEM solution

The weak formulation leads; find $u \in V$ such that

$$\int_0^{10} -u''vdx = \int_0^{10} (T_{ext} - u)vdx \quad \forall v \in V_o \tag{6}$$

where V_g is the trial space and V_o is the test space defined as

$$V_g = \left\{ \begin{array}{rr} v: & \int_0^{10} v^2dx < \infty, \int_0^{10}(v')^2dx < \infty \\ & v(0) = 40, v(10) = 200 \end{array} \right\} \tag{7}$$

$$V_o = \left\{ \begin{array}{rr} v: & \int_0^{10} v^2dx < \infty, \int_0^{10}(v')^2dx < \infty \\ & v(0) = 0, v(10) = 0 \end{array} \right\} \tag{8}$$

The left-hand side of the weak form can be simplified to

$$\begin{aligned}\int_0^{10} -u''vdx &= -\overbrace{u'v'\Big|_{x=0}^{x=10}}^{=0} + \int_0^{10} u'v'dx \\ &= \int_0^{10} u'v'dx\end{aligned}$$

The weak form can be instead formulated as
Find $u \in V_g$ such that

$$\int_0^{10} u'v'dx = \int_0^{10} (T_{ext} - u)vdx \quad \forall v \in V_o \tag{9}$$

Let $0 = x_0 < x_1 < \cdots < x_{n-1} < x_n = 10$ and define $h_j = x_j - x_{j-1}$, $j = 1, \ldots, n$ to be a local mesh size. The FEM formulation leads; find $u_h \in V_{g,h}$ such that

$$\int_0^{10} u_h'v'dx = \int_0^{10} (T_{ext} - u_h)vdx \quad \forall v \in V_{o,h} \tag{10}$$

where

$$V_{g,h} = \left\{ \begin{array}{rr} v: & v \in C^0(0,10), v \text{ linear function in} \\ & \text{every } I_j = (x_{j-1}, x_j), j = \overline{0,n} \\ & v(0) = 40, v(10) = 200 \end{array} \right\} \tag{11}$$

$$V_{o,h} = \left\{ \begin{array}{rr} v: & v \in C^0(0,10), v \text{ linear function in} \\ & \text{every } I_j = (x_{j-1}, x_j), j = \overline{0,n} \\ & v(0) = 0, v(10) = 0 \end{array} \right\} \tag{12}$$

We have that $v = \{\phi_1, \phi_2, \ldots, \phi_n\}$ since $v \in V_{o,h}$. u_h is the linear combinations

$$v_h = \sum_{j=0}^{n} c_j\phi_j(x) = c_0\phi_0 + c_n\phi_n + \sum_{j=1}^{n-1} c_j\phi_j \tag{13}$$

where $c_0 = 40$ and $c_n = 200$ according to the boundary condtions.

The left-hand side of (10) is

$$\begin{aligned}\int_0^{10} u_hv'dx &= \int_0^{10}\left(c_0\phi_0' + c_n\phi_n' + \sum_{j=1}^{n-1} c_j\phi_j'\right)\phi_i'dx \\ &= c_0\int_0^{10} \phi_0'\phi_i'dx + c_n\int \phi_n'\phi_i'dx + \sum_{j=1}^{n-1} c_j\int_0^{10} \phi_j'\phi_i'dx\end{aligned}$$

and the right-hand side of (10) is

$$\begin{aligned}&\int_0^{10}(T_{ext}-u_h)vdx\\&=\int_0^{10}\left(T_{ext}-c_0\phi_0-c_n\phi_n-\sum_{j=1}^{n-1}c_j\phi_j\right)\phi_i dx\\&=T_{ext}\int_0^{10}\phi_i dx-c_0\int_0^{10}\phi_0\phi_i dx\\&-c_n\int_0^{10}\phi_n\phi_i dx-\sum_{j=1}^{n-1}c_j\int_0^{10}\phi_j\phi_i dx\end{aligned}$$

The FEM formulation can then be written as

$$\begin{aligned}&\sum_{j=1}^{n-1}c_j\int_0^{10}\left(\phi_j'\phi_i'+\phi_j\phi_i\right)dx=T_{ext}\int_0^{10}\phi_i dx\\&-c_0\int_0^{10}(\phi_0'\phi_i'+\phi_0\phi_i)dx-c_n\int_0^{10}(\phi_n'\phi_i'+\phi_n\phi_i)dx\end{aligned}$$

We let the left-hand side be represented by Kc where $K=\int_0^{10}(\phi_j'\phi_i'+\phi_j\phi_i)dx$ and $c=\sum_{j=1}^{n-1}c_j$. the matrix K can be represented as

$$K=\begin{bmatrix}K_{i,i}&K_{i+1,i}&0&0&\dots&0\\K_{i,i-1}&K_{i,i}&K_{i+1,i}&0&\dots&0\\0&K_{i,i-1}&K_{i,i}&K_{i+1,i}&0&\vdots\\\vdots&0&\ddots&\ddots&\ddots&0\\0&\dots&0&K_{i,i-1}&K_{i,i}&K_{i+1,i}\\0&\dots&0&0&K_{i,i-1}&K_{i,i}\end{bmatrix}$$

where

$$\begin{cases}K_{i,i}=\dfrac{20}{3n}+\dfrac{n}{5}\\K_{i,i-1}=K_{i+1,i}=\dfrac{5}{3n}-\dfrac{n}{10}\end{cases}$$

The right-hand side of the FEM formulation can be represented by b as a $n\times 1$ vector where

$$b=\begin{pmatrix}\frac{10}{n}T_{ext}-c_0\left(\frac{5}{3n}-\frac{n}{10}\right)\\\frac{10}{n}T_{ext}\\\vdots\\\frac{10}{n}T_{ext}\\\frac{10}{n}T_{ext}-c_n\left(\frac{5}{3n}-\frac{n}{10}\right)\end{pmatrix}$$

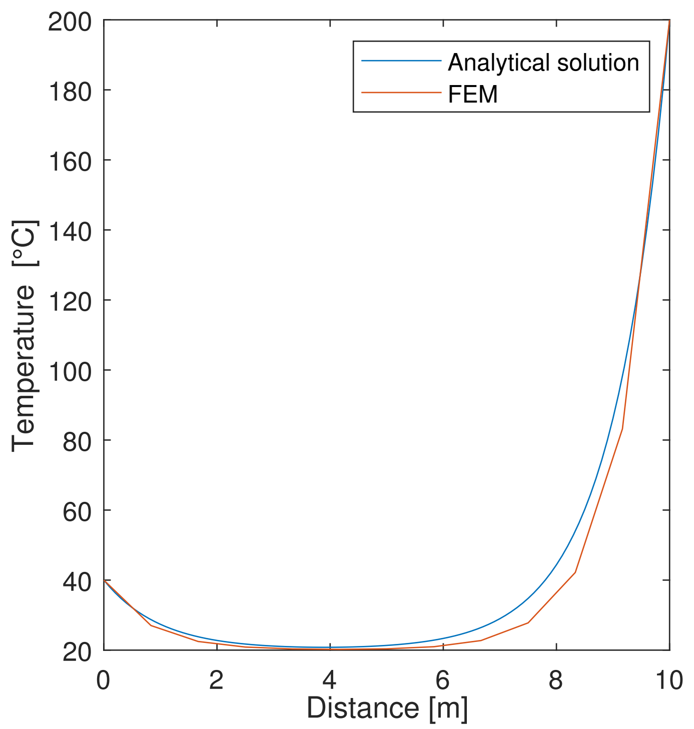
The solution $c=K^{-1}b$ gives the solutions $c_1,c_2,\dots,c_{n-1}$. The full solution is thus

$$c=[c_0,c_1,\dots,c_{n-1},c_n]^T \tag{14}$$

where c_0 and c_n are known to be 40 and 200 respectively.

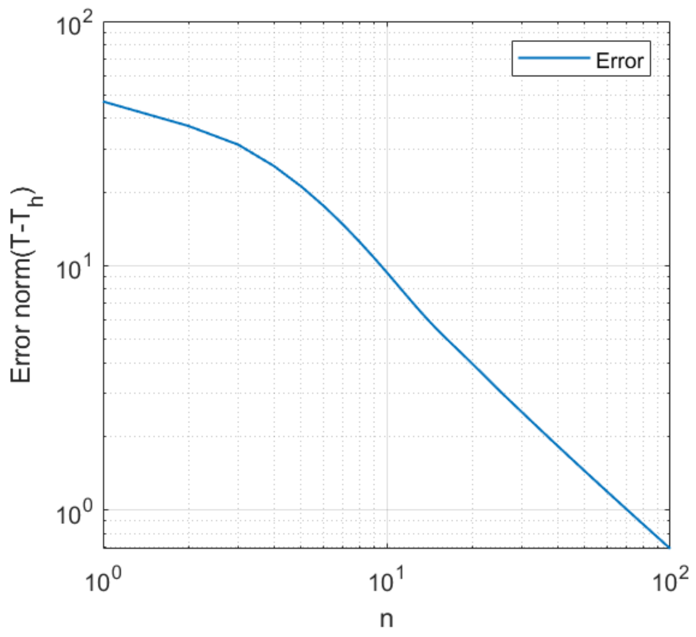
C. Result

Figure 1 shows the analytical and the FEM solution for the case where $n=100$ and $h_i=h_{i+1}=1/n$.



h

Fig. 1. Analytical and FEM solution to the 1D steady state heat equation with $N=10$



h

Fig. 2. Logarithmic normalized L_2 error

D. task 2: Convergence study

Figure 2 shows the error convergence for increasing n between $n=1$ and $n=100$. The error converges to zero for increasing step size.

II. Task 3 and 4

The stationary solution of the given FEM problem is solved in MATLAB's PDEtoolbox. The result is viewed in Figure 3 where the dimensions is normalized. The parameters $k = 3$ and $h = 2$ were used.

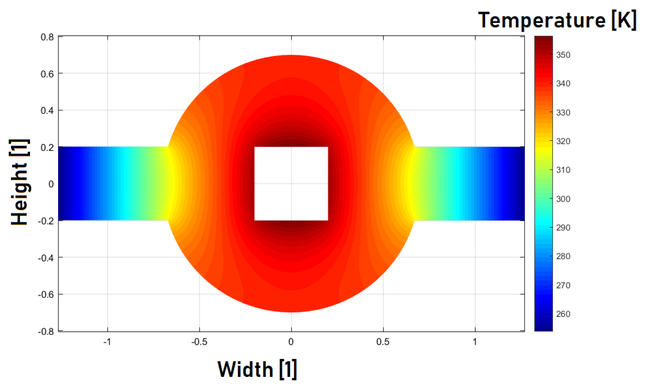


Fig. 3. Solution of

By ocular inspection of the figure, the warmest temperature is found to be around 360 K.

With $\rho = 1$ and $C = 1$ steady state is reached after 2-3 seconds.

III. Task 5: Hollow cylinder on hot surface

The steady-state heat conduction of an hollow aluminium cylinder placed on a hot surface is studied. The heat flux of the hot surface is modeled by a constant heat flux entering the bottom-most surface of the cylinder. Ambient temperature boundary conditions exists on all other faces of the cylinder. Furthermore forced convection by a low speed air flow over all but the bottom-most surface of the is assumed to exist.

$$\begin{cases} T(ext) = 20°C & (15) \\ \Phi_q = 5000\dfrac{W}{m^3} & (16) \end{cases}$$

where T_{ext} denotes the ambient temperature and Φ_q the heat flux into the bottom face of the cylinder.
The thermal properties of the material are specified as

$$\begin{cases} \epsilon_{al} = 0.07 & (17) \\ h = 10\dfrac{W}{m^2K} & (18) \\ k = 237\dfrac{W}{mK} & (19) \end{cases}$$

where the ϵ is the emissivity of the material, h is the heat transer coefficient of the aluminum-air interface and k is the thermal conductivity of the material.

The geometry of the model is showed in figure 2. The height is 10 cm and the outer and inner radius are 13 cm and 10 cm respectively. The heat flux enters into the bottom face of the cylinder.

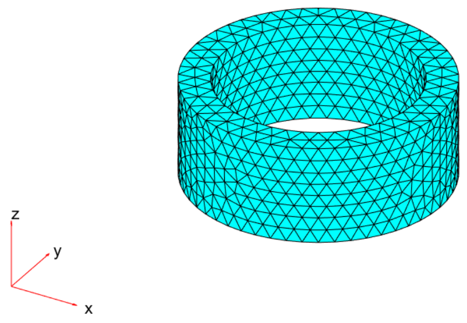


Fig. 4. Hollow Cylinder with Mesh

[]

A. Result

The thermal results show that the bottom-most part reaches 280 °C and the top-most part 100 °C in steady-state.

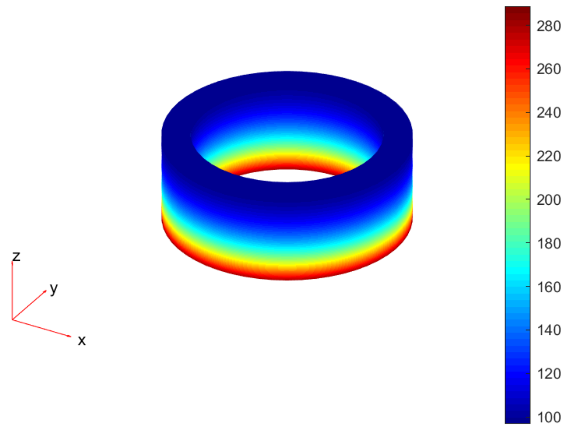


Fig. 5. Thermal results

B. Discussion

The heat conduction model is fairly realistic, however the constant heat flux would heat up the surrounding air of the cylinder affecting the ambient temperature boundary conditions on the surfaces of the cylinder. Thus the simulated temperature values are too low.

IV. Appendix

A. Task 1: FEM solution

```
% ==============================================
% task 1: Analytical and Numerical solution
%                 of the heat equation.
% ==============================================

% ANALYTICAL SOLUTION
x = 0:0.1:10; % Boundaries form 0 to 10m .

y = @(x) exp(x).*(20-180.*exp(10))./(1-exp(20)) + ...
exp(-x).*(180*exp(10)-20*exp(20))/(1-exp(20)) +20;

plot(0:0.1:10, y(0:0.1:10))


% Numerical solution.
n=10;
Text = 20; c0 = 40; cn = 200;
e=ones(n+1,1);
K=spdiags([e*(5/(3*n)-n/10) e*(20/(3*n)+n/5) ...
e*(5/(3*n)-n/10)],-1:1,n+1,n+1);
b=10*Text/n.*ones(n+1,1);
b(1) = b(1) - c0*(5/(3*n)-n/10);
b(end) = b(end) - cn*(5/(3*n)-n/10);
c=mldivide(K,b);

% adding boundaries
T = [c0; c; cn];
x2 = 0:10/(n+2):10;

%plotting FEM sol
hold on;
plot(x2,T);
```

B. Task2: Error estimation

```
% =================================
%   TASK2: Convergence test
% =================================

close all; clear all; clc;
y = @(x) exp(x).*(20-180.*exp(10))./(1-exp(20))+ ...
exp(-x).*(180*exp(10)-20*exp(20))/(1-exp(20)) +20;


N = 100;
Text = 20; c0 = 40; cn = 200;
errs = zeros(1,N);

for n = 1:1:N

    % Numerical solution for n.
    e = ones(n+1, 1);
    K = spdiags([e*(5/(3*n)-n/10) ...
    e*(20/(3*n)+n/5) e*(5/(3*n)-n/10)], ...
    -1:1, n+1, n+1);
    b = 10*Text/n.*ones(n+1, 1);
    b(1) = b(1) - c0*(5/(3*n)-n/10);
    b(end) = b(end) - cn*(5/(3*n)-n/10);
    c=mldivide(K,b);

    % adding boundaries
    x2 = 0:10/(n+2):10; % x axis
    Tfem = [c0; c; cn];    % FEM solution
    Treal = y(x2);
    err = Treal-Tfem';
    errs(n) =  sqrt(sum(abs(err).^2)/n);
end


plot(1:1:N,errs./max(errs))
xlabel('Degree of freedom');
ylabel('Normalized L_2 error')
```

C. Task 5: code

```
    %Hollow cylinder on hot plate
model=createpde('thermal');
gm = multicylinder([10 13],[10 10],'void',[1,0,])
model.Geometry=gm;
generateMesh(model);

figure(1)
pdegplot(model,'FaceLabels','on', 'CellLabels', 'on');

%Boundary conditions
thermalBC(model,'Face',1,'HeatFlux', 10000);
%Hot plate heat source Wm^3
thermalBC(model,'Face',2,'ConvectionCoefficient',10, ...
'AmbientTemperature', 20,'Emissivity', 0.07);
thermalBC(model,'Face',3,'ConvectionCoefficient',10, ...
'AmbientTemperature', 20,'Emissivity', 0.07);
thermalBC(model,'Face',4,'ConvectionCoefficient',10, ...
'AmbientTemperature', 20,'Emissivity', 0.07);

%Thermal Properties
thermalProperties(model,'cell',1,'ThermalConductivity',237);
model.StefanBoltzmannConstant=5.670367e-8;

%Solving
results=solve(model);
figure(2)
pdeplot3D(model,'ColorMapData',results.Temperature)
```

Vibration of a string

Elvis Rodas, Patrik Wall, Jamel Hudson

I. Problem description

The pattern of vibration of a steel string is modelled. The string is lifted a distance $\hat{u}$ from a distance of $\hat{x}$ from one end. The equation describing the vibration of the string is the one-dimensional wave equation

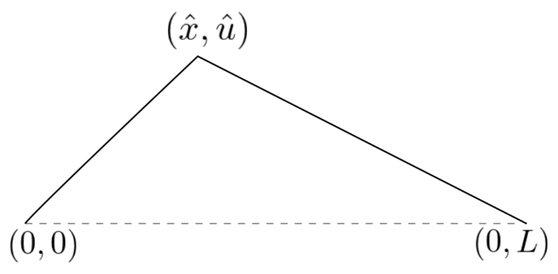


Fig. 1. Initial condition.

$$\begin{cases} u_{tt} = c^2 \cdot u_{xx}, \quad x \in [0,L] & (1) \\ u(x,0) = \hat{u} \cdot \dfrac{x}{\hat{x}}, \quad x \in [0,\hat{x}] & (2) \\ u(x,0) = \hat{u} \cdot \dfrac{x-L}{\hat{x}-L}, \quad x \in [\hat{x},L] & (3) \\ u_t = 0, \quad x \in [0,L] & (4) \\ u(0,t) = 0, \quad t \geqslant 0 & (5) \\ u(L,t) = 0, \quad t \geqslant 0 & (6) \end{cases}$$

where L is the length of the string, u is the displacement of the string in y-direction (See figure 1). The wave speed, c is computed as

$$c = \sqrt{\frac{T}{\rho}},$$

where T is the tension and ρ is the linear mass density (mass/length). The relevant parameters for the steel string being modeled are

Parameter	Typical value
Diameter d	0.5-1 mm
Density of steel ρ	7800 kg/m^3
Tension T	440 N

The linear mass density, ρ, is calculated as

$$\text{Density} \cdot \text{Area} = 7800 \cdot \pi r^2$$

yielding $\rho = 0.0245$ kg/m

The wave speed is then

$$c = 133.98 \text{ m/s}$$

II. Solution method

The problem is solved by use of finite differences method.

A. Discretizing the domain

A finite difference mesh is introduced, replacing the temporal and spatial domain with a finite number of meshpoints $u_n^k = u(x_n, t_k)$, rows t_k and columns x_n.

$$\begin{cases} x_n = n\Delta x & (7) \\ t_k = k\Delta t & (8) \\ u_n^k = u(x_n, t_k) & (9) \\ & (10) \end{cases}$$

where $n, k \in \mathbb{Z}^+$

B. Approximating derivatives with finite differences

Equation (1) is approximated by central differences and an algebraic expression of the PDE is formed as

$$\frac{u_n^{k+1} - 2u_n^k + u_n^{k-1}}{\Delta t^2} = c^2 \Big(\frac{u_{n+1}^k - 2u_n^k + u_{n-1}^k}{\Delta x^2} \Big) \qquad (11)$$

or

$$u_n^{k+1} = r^2 u_{n-1}^k + 2(1-r^2) u_n^k + r^2 u_{n-1}^k - u_n^{k-1} \qquad (12)$$

where $r = c\Delta t / \Delta x$ and the case where u_n^0 is defined by the initial conditions (2) and (3).

Equation (4) is in the same manner approximated by central differences and an algebraic version is formed as

$$\frac{u_n^1 - u_n^{-1}}{2\Delta t} = 0 \qquad (13)$$

yieldning

$$u_n^1 = u_n^{-1} \qquad (14)$$

The first time step is given by (see figure 2)

$$\begin{aligned} u_n^1 &= r^2 u_{n-1}^0 + 2(1-r^2) u_n^0 + r^2 u_{n-1}^0 - u_n^{-1} \\ u_n^1 &= \frac{1}{2} r^2 u_{n-1}^0 + 2(1-r^2) u_n^0 + r^2 u_{n-1}^0 \qquad (15) \end{aligned}$$

where the equation (14) was used.

In order to run a simulation, u_n^0 is computed by the initial condtions (2) and (3), u_n^1 is computed by (15) and further steps are computed by (12).

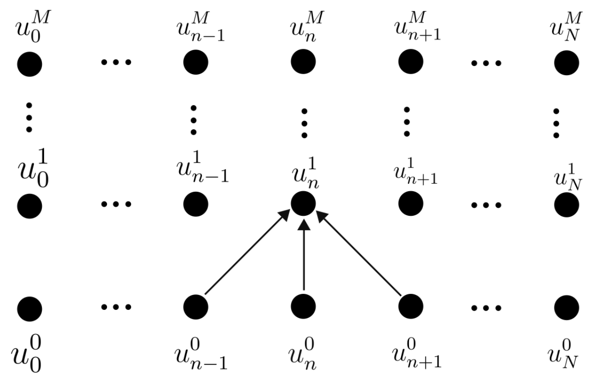


Fig. 2. Graphical visualization of finite differential method.

C. Von Neumann Stability Analysis

Let $u_n^j = q^n e^{-i\omega jk}$. The method (11) the becomes

$$\frac{q^n e^{-i\omega(j+1)k} - 2q^n e^{-i\omega jk} + q^n e^{-i\omega(j-1)k}}{\Delta t^2} \tag{16}$$

$$= c^2 \left(\frac{q^{(n+1)} e^{-i\omega jk} - 2q^n e^{-i\omega jk} + q^{(n-1)} e^{-i\omega jk}}{\Delta x^2} \right) \tag{17}$$

or

$$e^{-i\omega k} - 2 + e^{+i\omega k} = r^2(q - 2 + q^{-1}) \tag{18}$$

by factorizing $q^n e^{-i\omega jk}$. The lefthand side can be writen as $-4\sin^2(\omega k/2)$. The equation then becomes

$$-4\sin^2(\omega k/2) = r^2(q - 2 + q^{-1}) \tag{19}$$

$$-4\sin^2(\omega k/2) = r^2 \left(\frac{q^2 - 2q + 1}{q} \right) \tag{20}$$

$$-\frac{4\sin^2(\omega k/2)}{r^2} q = q^2 - 2q + 1 \tag{21}$$

$$0 = q^2 - \underbrace{(2 - \frac{4\sin^2(\omega k/2)}{r^2})}_{b} q + 1 \tag{22}$$

We know that an equation of the form $0 = q^2 - bq + 1$ has the solution $q = b/2 \pm \sqrt{b^2/4 - 1}$ where

- $\frac{b^2}{4} > 1$ unstable.
- $\frac{b^2}{4} = 1$ weakly unstable.
- $\frac{b^2}{4} < 1$ stable.

Thus, in order to have stabiblity we need

$$\frac{\left(2 - \frac{4\sin^2(\omega k/2)}{r^2}\right)^2}{4} < 1 \tag{23}$$

yielding

$$|1 - 2r^2 \sin^2(\omega k)| < 1 \tag{24}$$

$$0 \geq 2r^2 \sin^2(\omega k) < 2 \tag{25}$$

which is valid for $r^2 < 1$ since $0 \leq \sin^2(\omega k) \leq 1$.

Thus

$$\frac{c\Delta t}{\Delta x} < 1 \tag{26}$$

D. Local Truncation Error and Order of Accuracy

The local truncation error, τ_j, of the approximation of the PDE, eq. (11), can be found by Taylor expanding the terms of (11) and subtracting the analytical PDE, eq. (1).

Taylor expanding u_n^{k+1} around u_n^k we have that

$$\begin{cases} u_n^{k+1} \approx u + \Delta t \cdot u_t + \frac{\Delta t^2}{2} \cdot u_{tt} + \mathcal{O}(\Delta t^3) & (27) \\ u_n^{k-1} \approx u - \Delta t \cdot u_t + \frac{\Delta t^2}{2} \cdot u_{tt} - \mathcal{O}(\Delta t^3) & (28) \\ u_{n+1}^{k} \approx u + \Delta x \cdot u_x + \frac{\Delta x^2}{2} \cdot u_{xx} + \mathcal{O}(\Delta x^3) & (29) \\ u_{n-1}^{k} \approx u - \Delta x \cdot u_x + \frac{\Delta x^2}{2} \cdot u_{xx} - \mathcal{O}(\Delta x^3) & (30) \end{cases}$$

Inputting Taylor expansion approximations yields

$$u_{tt} + \mathcal{O}(\Delta t^2) = c^2 \cdot u_{xx} + \mathcal{O}(\Delta x^2) \tag{31}$$

Subtracting the analytical PDE from (27) it can be seen that the local trunction error is of order $\mathcal{O}(\Delta t^2, \Delta x^2)$, that is of second order in both time and space. The order of accuracy of the proposed finite difference method, eq. (12), is thus of order 2 in both time and space. Furthermore the method is consistent since the local trunction error, $|\tau_j|$, $\to 0$ when Δt and $\Delta x \to 0$.

III. RESULT

A. Test of theoretical stability condition

The theoretical stability condition $\frac{c\Delta t}{\Delta x} < 1$ is tested. Plotting the vibration of the string for $\frac{c\Delta t}{\Delta x} = 1.02 > 1$ results in a unstable solution, as shown in figure 4.

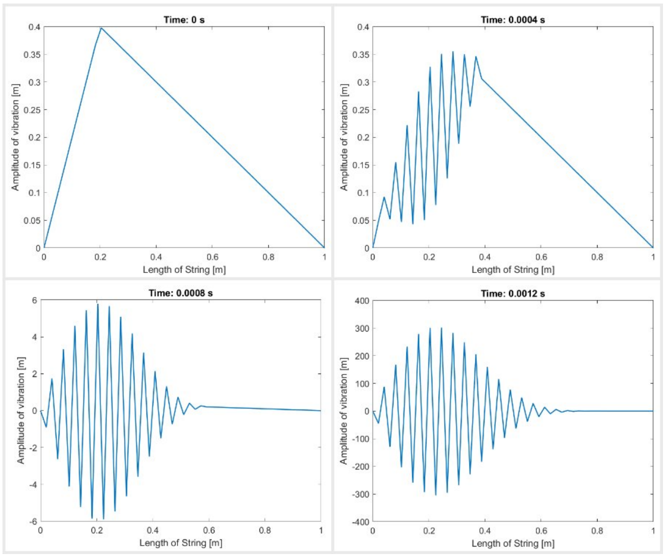


Fig. 4. Vibration of the string for $c\frac{\Delta t}{\Delta x} = 1.022 > 1$

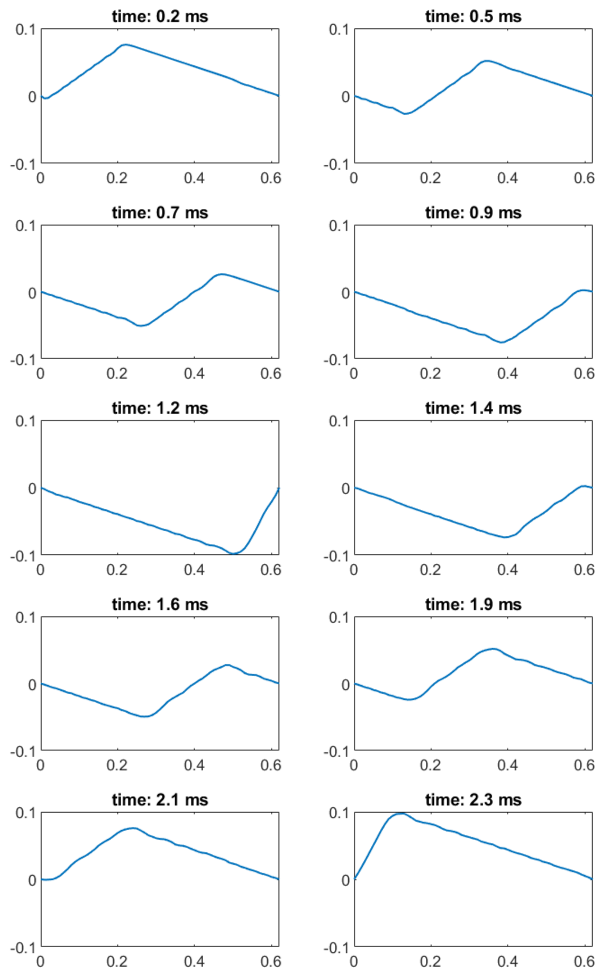


Fig. 3. Simulation over one period.

B. *Global Error Convergence-Experiment With Known Solution*

In this section the accuracy of the algorithm is analyzed. The problem is simplified by replacing equation (2) and (3) with with a standing wave,

$$u(x,0)^{(*)} = A\sin(2\pi x), \quad x \in [0,L],$$

which has the analytical solution, **??**

$$u(x,t)^{(*)} = Asin(kx)cos(\omega t), \quad x \in [0,L], \quad t = 0,1,...$$

where $k = \frac{2\pi n}{2L}$ and n is the n^{th} overtone, which is set to 2 in this experiment. The frequency and material parameters are chooses as suggested in the laboratory instructions.

With the numerical solution and the analytical solution known, the error is calculated with a global error (GE) formula,

$$GE = \sum_{t=0}^{t_{max}} \sum_{x=0}^{L} \Big(||u(x,t) - u^{(*)}(x,t)|| \Big),$$

where $t_{max} = 0.0095s$ and $\Delta x = 0.02$. The GE value is saved for 100 different valued time steps in the interval $[0.19,\ 0.0002]$. The result is seen in Figure (5).

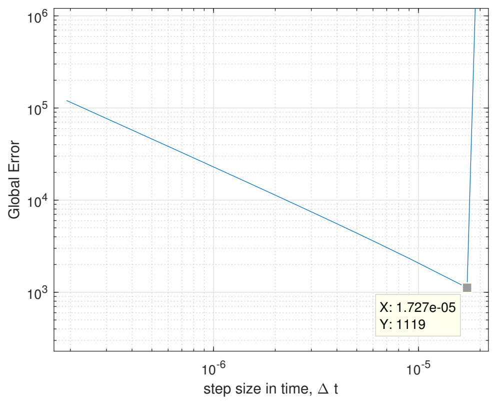


Fig. 5. The global error over 100 simulations with different time step sizes.

IV. Discussion

The global error was expected to decrease as the step size decreased as shown in Eq (31). As seen in Figure 5 the global error is the smallest just before divergence. We suspect that by adding more time steps the netto error grows. We also suspect that Eq(5) is not the true global error as defined in the lectures.

V. Conclusion

The pattern of vibration of a steel string is modelled by use of finite differences. The partial differential equation (1) is approximated with second order central differences in both time and space. The method is found to be second order accurate in time and space. An equation modelling of the position of the string one step ahead in time, u_n^{k+1}, is then formed and used in a loop to compute the next values. From the results it can be seen that the pattern of vibration after the string has been plucked is indeed initially ”V-shaped”. Some limitations of the method is found with Von Neumann Analysis. A condition for stability is theoretically derived as $\frac{c\Delta t}{\Delta x} < 1$ and also verified numerically.

The results from Figure 5 suggest that best accuracy over time is achieved when r is as close to as possible.

VI. Appendix

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%            Finite difference method of wave equation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% The finite  difference matrix :
% % ---------------------------------------------------------------
% a) For k = 0:
%    U(0,n) = inverted V shape where highest point at xhat & hat.
%
% b) For k = 1:
%    u(1,n) = 0.5*r^2*u(0,n-1) + (1-r^2)*u(0,n) + 0.5*r^2*U(0,n+1)
%    where we used the fact that u(-1,n) = u(1,n).
%
% c) For k > 1:
%    u(k+1,n) = r^2*u(k,n-1) + 2*(1-r^2)*u(k,n) + r^2*u(k,n+1) - u(k-1,n)
% ---------------------------------------------------------------

% Spacial and temporal limits
xlen = 63e-2;    % spacial length [m]
time = 40;       % temporal length [s]

% Discretisation of domamins
dx = 0.01; M = round(xlen/dx);
dt = 0.01; N = round(time/dt);
U = zeros(N,M);      % M x N matrix. x(0) = x(L) = 0 included.

% Initial spacial conditions. (Lifted xhat at distance uhat).
xhat = 10e-2;    % [m] (mitten)
uhat = 10e-2;    % Lyfted 10cm (godtyckligt).

% constants
T = 440;                            % tension [N]
rho = 7800;                         % Density [kg/m^3]
diam = 0.5e-3;                      % Diameter [m]
c = sqrt(T/(pi*diam^2/4*rho));      % wave speed [m/s]
r = c*dt/dx;          % variable defined in (7)

% a) Computing u(0,n)
xv =0:dx:xlen-dx;
U(1,:) =  [uhat/xhat.*xv(xv < xhat), uhat/(xhat - xlen).*(xv(xv>=xhat) - xlen)];

% Computing u(1,n)
U(2,2:end-1) = [0.5*r^2, 1-r^2, 0.5*r^2] * [U(1,1:end-2); U(1,2:end-1); U(1,3:end)];
% Computing u(k,n) for k > 1
for k = 2:N
    U(k+1,2:end-1) = [r^2, 2*(1-r^2), r^2] * [U(k,1:end-2); U(k,2:end-1); U(k,3:end)] ...
    - U(k-1,2:end-1);
end

% Plotting:
for n = 1:N
plot(U(n,:));
%ylim([-uhat, uhat]);
pause(0.01);
end
```