FLAT EARTH SEISMIC WAVE

Ashvat Bansal
University of Illinois Urbana Champaign

May 13, 2020

1 Introduction

The wave equation is an important second-order linear partial differential equation for the description of waves. This wave equation is used to model many waves, such as mechanical waves (e.g. water waves, sound waves and seismic waves) or light waves. It arises in fields like acoustics, electromagnetics, and fluid dynamics[1].

One such important wave are the seismic waves. Earthquakes and tsunamis are natural disasters that occur very frequently. Natural events such as volcanic eruptions and meteor impacts can cause earthquakes, but the majority of earthquakes occur due to the movement of the earth's tectonic plates. In fact, we don't feel most of the earthquakes that occur. According to the National Earthquake Information Center (NEIC), an average of 20,000 earthquakes are recorded every year (50 per day) around the world. There are, however, millions of earthquakes estimated to occur every year that are too weak to be recorded[2].

To go on further, it is important to understand how and why earthquakes occur. The motion of the tectonic plates causes a steady increase in stress. When this stress sufficiently large, it is released in form of energy along the faults in Earth's crust[3]. This produces waves known as seismic waves, which propagate through the Earth's crust, transferring energy which results in the bending and buckling of Earth's crust. If this energy is released on continental land, we get earthquakes, and if its released under the oceans (ocean floor), the transfer of energy produces tsunamis.

With all the talk and theories floating around about Earth being flat, we thought it might be a good and fun idea to model the propagation of a seismic wave along the Earth's surface, and how it would interact with the Earth's boundary when it reaches the "Edge of the Earth". For this project, we will be considering the S-Waves (Secondary Waves), which arrive after the faster moving P-waves (Primary waves) and displace the ground perpendicular to direction of propagation

2 The Wave Equation

The general wave equation is a 3 dimensional partial differential equation, but for the purpose of this project, we're only considering the reduced simple 1-D wave equation given by:

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2} + g(x,t)$$

where c is a constant referred to as the wave speed, and $g(x,t)$ is the prescribed forcing function. This equation will be used to model the motion of seismic wave along the crust of our flat Earth, but first we to determine some parameters before we can start modelling the wave propagation.

[1] https://en.wikipedia.org/wiki/Wave_equation
[2] https://www.dosomething.org/us/facts/11-facts-about-earthquakes
[3] http://www.seismo.ethz.ch/en/knowledge/things-to-know/causes-of-earthquakes/general/

2.1 Modification of Wave Equation for Seismic Waves

The given wave equation is a general wave equation, and needs to be modified to fit the seismic waves. For the purpose of this project, we will consider that the wave is propagating through the Earth's crust. Since the most abundant element in Earth's crust is silicon, we assume that the crust is homogeneous and made of just silicon, for simplicity.

Further, there are 2 types of main seismic waves that propagate through the crust, P-waves and S-waves. P-waves, also known as Primary Waves, are pressure waves, and are the fastest travelling seismic waves[4]. P-waves are presure waves and propagate by alternating compressions and rarefactions[5]. The Secondary waves are known as S-waves, and are shear waves. S-waves are slower thatn P-waves, and propagate by displacing the medium perpendicular to the direction of propagation. Hence, using this knowledge, and a combination of Newton's second law with a relationship relating stress and strain, the wave speed is given as

$$c = \sqrt{\frac{\lambda + 2\mu}{\rho}} \quad \text{for P-wave}$$

$$c = \sqrt{\frac{\mu}{\rho}} \quad \text{for S-wave}$$

where λ (Elastic Modulus) and μ (Shear Modulus) are known as Lamé parameters, and ρ is the density of medium. Since we're only focusing on S-waves, we just need to know ρ and μ. After doing some research, we found that silicon density is 2.329 g/cm^3[6] and the shear modulus for silicon ranges from 51-80 Gpa [7]. Using $\rho = 2329\ kg/m^3$ and $\mu = 80 \cdot 10^9 pa$, the wave speed was found to be 5.86 km/s

2.2 Initial and Boundary Conditions

We chose L (the total domain) to be the circumference of the Earth, since that would be the length of the Earth if the Earth was flat. Based on this, the initial and boundary conditions were chosen as follows:

Initial Conditions

The initial and boundary conditions were determined trying to emulate how a seismic wave is generated, and how it should interact with the boundary in our very hypothetical situation[8].

To emulate the sudden burst of energy and wave amplitude, the Gaussian function was used, which produces a bell curve which was used as the model for a representation of the seismic wave. The center of this curve was chosen as the rough estimation of where Japan would lie

[4] https://en.wikipedia.org/wiki/P-wave
[5] https://en.wikipedia.org/wiki/P-wave
[6] https://en.wikipedia.org/wiki/Silicon
[7] https://en.wikipedia.org/wiki/Silicon
[8] I AM NOT A FLAT EARTHER

on flat Earth (since it's one of the post Earthquake prone countries), based on the distance between Japan and Greenwich. Hence the initial position was given by

$$u(x, t = 0) = A \cdot \exp\left(- \frac{(x - x_0)^2}{(2\sigma)^2} \right)$$

Where A is the amplitude of the wave, x_0 is the location where the wave starts propagation (measured from left boundary), and σ controls the sharpness of the waveform. Since we're modelling the system that has been scaled down by a factor of 10, A was chosen as 0.0002 (km), since the amplitude of an earthquake of magnitude 3 on Richter Scale is 1 meter, so the wave amplitude at the epicenter would be twice that amount due to superposition. σ was chosen as 87.5 and x_0 was chosen as 3082.5 km, as that should be the rough location of japan if we take International date line as the center with USA being on left.

Similarly, the initial velocity was as

$$\frac{\partial u}{\partial t} = 0 \quad \text{at t=0}$$

since the epicenter is the point of origin of the wave, and should not have an initial velocity.

Boundary Conditions

Since we're using a Flat Earth system, the boundaries were considered as the "Edge of the Earth", and hence the 0 Dirichlet Boundary Conditions were used.

$$u(x = 0, t) = 0$$
$$u(x = L, t) = 0$$

Since our theory is that if the Earth was indeed flat, the waves would just continue propagating and transfer over to the lower surface of the Earth (Assuming that Earth has at least a certain thickness). It was assumed that the wave does not dampen and lose amplitude (for simplicity) and hence the 0 Dirichlet Boundary Conditions were perfect as they are perfectly reflective boundary conditions.

Forcing Function

In an Earthquake, all of the energy is released at once, at the epicenter. Because of this, the forcing function was chosen as

$$g(x, t) = 0$$

since we do not want the wave to be produced continuously and only want a single waveform travelling through.

3 The Numerical Method

3.1 Description/Overview

Finite Difference Derivation

[9] Our project used the finite difference method to discretize wave differential equation. The method divides the domain into individual interpolation points, which can then be used to approximate the solution to the PDE using using basis functions. Finite Difference Methods convert a linear ordinary differential equations (ODE) or non-linear partial differential equations (PDE) into a system of equations that can be solved by matrix algebra techniques.

First, we discretize the spatial domain by using Method of Lines. To do this, we break up the domain into a set of uniformly distributed interpolation points over the interval as:

$$x_j = a + \frac{(b-a)(j-1)}{n}, \quad j = 1, \ldots, n+1 \tag{1}$$

We can approximate the spatial dependence of our function $u(x)$ by local interpolation. For each interpolation point x_j, a p^{th} order polynomial is used to approximate $u(x)$. For this, we need $p+1$ points to define the polynomial (p points in addition to x_j):

$$\{x_{j-p/2}, \ldots, x_j, \ldots, x_{j+p/2}\} \tag{2}$$

We will use p = 2 to get our points, which gives us the points

$$\{x_{j-1}, x_j, x_{j+1}\} \tag{3}$$

We now approximate the solution variable $u(x,t)$ in terms of the Lagrange polynomials as follows,

$$u(x,t) \approx \sum_{i=j-p/2}^{j+p/2} b_i(t) L_i^{(j)}(x_j) = \sum_{i=j-1}^{j+1} b_i(t) L_i^{(j)}(x_j) \tag{4}$$

To simplify the above into an initial value problem, we represent the spatial variable x by the n+1 points,$\{x_1, x_2, , x+n+1\}$, and restrict the spatial dependence of the function u to be a linear combination of the Lagrange polynomials. Now, plugging the approximation for $u(x,t)$ as shown in (4), into the wave equation gives,

$$\sum_{i=j-1}^{j+1} \ddot{u}_i(t) L_i^{(j)}(x_j) = c^2 \sum_{i=j-1}^{j+1} u_i(t) \left.\frac{d^2 L_i^{(j)}}{dx^2}\right|_{x=x_j} + g(x_j,t) \quad (j = 2, ..., n) \tag{5}$$

Noticing that

$$L_i^{(j)}(x_j) = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases} \tag{6}$$

[9] HW8 and Week 13 Typed Notes

$$u(x_j,t) \approx \sum_{i=j-1}^{j+1} b_i(t) L_i^{(j)}(x_j) = b_j(t) \tag{7}$$

Equation (5) can be further simplified into

$$\ddot{u}_j(t) = c^2 \sum_{i=j-1}^{j+1} u_i(t) \frac{d^2 L_i^{(j)}}{dx^2}\bigg|_{x=x_j} + g(x_j,t) \quad (j=2,...,n) \tag{8}$$

Focusing on the right hand side of equation (8), we simplify the Lagrangian terms into

$$\begin{aligned} L_{j-1}^{(j)}(x) &= \frac{(x-x_j)(x-x_{j+1})}{(x_{j-1}-x_j)(x_{j-1}-x_{j+1})} = \frac{1}{2\Delta x^2}(x-x_j)(x-x_{j+1}) \\ L_j^{(j)}(x) &= \frac{(x-x_{j-1})(x-x_{j+1})}{(x_j-x_{j-1})(x_j-x_{j+1})} = -\frac{1}{\Delta x^2}(x-x_{j-1})(x-x_{j+1}) \\ L_{j+1}^{(j)}(x) &= \frac{(x-x_{j-1})(x-x_j)}{(x_{j+1}-x_{j-1})(x_{j+1}-x_j)} = \frac{1}{2\Delta x^2}(x-x_{j-1})(x-x_j) \end{aligned} \tag{9}$$

Substituting the Lagrange polynomials from (9) into (8) gives

$$\ddot{u}_j(t) = c^2 \left[u_{j-1} \frac{d^2 L_{j-1}^{(j)}}{dx^2}_{x=x_j} + u_j \frac{d^2 L_j^{(j)}}{dx^2}_{x=xj} + u_{j+1} \frac{d^2 L_{j+1}^{(j)}}{dx^2} \right] + g(x_j,t) \quad (j=2,...,n)$$

$$\implies \ddot{u}_j(t) = \frac{c^2}{\Delta x^2} \left[u_{j-1} - 2u_j + u_{j+1} \right] + g(x_j,t) \quad (j=2,...,n) \tag{10}$$

Using the boundary conditions, u_1 and u_{n+1} can be expressed as $g(x_1,t)$ and $g(x_{n+1},t)$ respectively, where x_1 and x_{n+1} correspond to the end points of the domain. Thus, $\ddot{u}_2$ and $\ddot{u}_n$ as

$$\ddot{u}_2(t) = \frac{c^2}{\Delta x^2}[-2u_2(t) + u_3(t)] + g(x_2,t) + \frac{c^2}{\Delta x^2} g(a,t) \tag{11}$$

$$\ddot{u}_n(t) = \frac{c^2}{\Delta x^2}[u_{n-1} - 2u_n(t)] + g(x_n,t) + \frac{c^2}{\Delta x^2} g(b,t) \tag{12}$$

where a is the left boundary of domain and b is the right boundary of domain.

To get our system into a state space of the form $\dot{u} = Au + g$, we need to manipulate the matrices to get it in the format wanted. Currently we have $\ddot{u} = A_{temp} \cdot u_{temp} + g_{temp}$

$$\begin{bmatrix} \ddot{u}_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} = \frac{c^2}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} g(x_2,t) + \frac{c^2}{\Delta x^2} g_a(t) \\ g(x_3,t) \\ \vdots \\ g(x_{n-1},t) \\ g(x_n,t) + \frac{c^2}{\Delta x^2} g_b(t) \end{bmatrix} \tag{13}$$

where $j = 2, \ldots, n$.

Now, our system can be converted into the form of $\dot{u} = Au + g$. Let

$$\dot{u_j} = v_j \tag{14}$$
$$\dot{v_j} = Au_j + g_j \tag{15}$$

The above can be represented as

$$\begin{bmatrix} \dot{u_j} \\ v_j \end{bmatrix} = \begin{bmatrix} 0 & I \\ A_{temp} & 0 \end{bmatrix} \begin{bmatrix} u_j \\ v_j \end{bmatrix} + \begin{bmatrix} 0 \\ g_j \end{bmatrix} \tag{16}$$

Hence we have obtained our system in the form of $\dot{u} = Au + g$, where

$$u = \begin{bmatrix} u_{temp} \\ \dfrac{d(u_{temp})}{dt} \end{bmatrix} \qquad A = \begin{bmatrix} 0 & I \\ A_{temp} & 0 \end{bmatrix} \qquad g = \begin{bmatrix} 0 \\ g_{temp} \end{bmatrix}$$

Trapezoid Method Derivation

[10]

Having an initial value problem as

$$\dot{u} = f(u, t) \tag{17}$$
$$u(t_0) = u_0 \tag{18}$$

We iterate (17) from t_k to t_{k+1}:

$$\int_{t_k}^{t_{k+1}} \dot{u}\, dt = \int_{t_k}^{t_{k+1}} f(u(t), t)\, dt \tag{19}$$

We can evaluate the left hand side without any error by $u(t_{k+1}) - u(t_k)$. To evaluate the right hand side, we approximate it as a line over the interval $t \in [t_k, t_{k+1}]$ and interpolate. Using the Lagrange Basis:

$$f(u(t), t) \approx f(u(t_k), t_k)\Big(\frac{t - t_{k+1}}{t_k - t_{k+1}}\Big) + f(u(t_{k+1}), t_{k+1})\Big(\frac{t - t_k}{t_{k+1} - t_k}\Big) \tag{20}$$

Plugging this expression into (17) and denoting $\triangle t = t_{k+1} - t_k$, we get

$$u(t_{k+1}) - u(t_k) \approx \int_{t_k}^{t_{k+1}} f(u(t_k), t_k)\Big(\frac{t - t_{k+1}}{-\triangle t}\Big) + f(u(t_{k+1}), t_{k+1})\Big(\frac{t - t_k}{\triangle t}\Big) dt \tag{21}$$

Since the $f(u(t_k), t_k)$ and $f(u(t_{k+1}), t_{k+1})$ terms are constant (f is being evaluated at specific values of u and t), as is $\triangle t$, we can pull these terms out of the integral to get

[10] HW5 In Class Solutions

$$u(t_{k+1}) - u(t_k) \approx \frac{f(u(t_k), t_k)}{-\triangle t} \int_{t_k}^{t_{k+1}} (t - t_{k+1})dt + \frac{f(u(t_{k+1}), t_{k+1})}{\triangle t} \int_{t_k}^{t_{k+1}} (t - t_{k+1})dt \quad (22)$$

$$= \frac{\triangle t}{2}(f(u(t_k), t_k) + f(u(t_{k+1}), t_{k+1})) \quad (23)$$

The form (23) is written in terms of the exact solution, u(t), at t_k and t_{k+1}. In general, we do not have this quantity; we start from an approximate solution at t_k and the result of applying the trapezoid method will again give us an approximate answer for u at t_{k+1}. We therefore replace the form (23) with

$$u_{k+1} - u_k = \frac{\triangle t}{2}(f(u_k), t_k) + f(u_{k+1}, t_{k+1})) \quad (24)$$

to make it clear that we are obtaining an approximate solution to the IVP.

We can also describe our linear system $f(u, t)$ as:

$$f(u, t) = \dot{u} = Au + g(t) \quad (25)$$

Substituting $f(u_k, t_k)$ back in gives the final form

$$u_{k+1} = \left(I - \frac{A\Delta t}{2}A\right)^{-1} \left[u_k \left(I + \frac{A\Delta t}{2}\right) + \frac{\Delta t}{2}(g(t_k) + g(t_{k+1}))\right] \quad (26)$$

which can be used to approximate the IVP at each time step.

3.2 Justification for Choice of Method

The Trapezoidal method was chosen for this numerical analysis since this method has good accuracy as well as great stability, which is important in this case since we're dealing with large numbers that emulate the real world parameters. Given below is the stability plot for Trapezoidal method

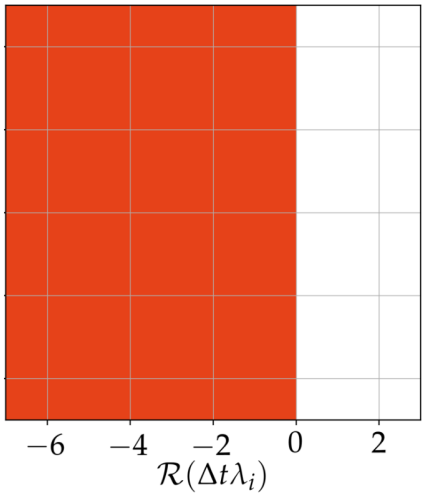


Figure 1: Stability Plot for Trapezoidal Numerical Method

As it can be seen, the stability region for Trapezoidal Method is quite large, and due to this reason Trapezoidal Method was used in the Numerical Analysis.

3.3 Demonstration of Correct Implementation

To show that our method was implemented correctly, the Method of Manufactured Solutions (MMS) was used. For this, an arbitrary wave function was chosen for which we know the solution for. Hence, $u_{pick} = A \cdot sin(wt - kx)$ was chosen as our function, since this is the solution to the harmonic wave equation. Here, w is the frequency and k is the wave number, while c is the wave speed and A is the amplitude. By putting this function into the wave equation

$$\frac{\partial^2 u_{\text{pick}}}{\partial t^2} = c^2 \frac{\partial^2 u_{\text{pick}}}{\partial x^2} + g(x,t)$$

with $\frac{\partial^2 u_{\text{pick}}}{\partial t^2} = -c^2 w^2 sin(wt - kx)$ and $c^2 \frac{\partial^2 u_{\text{pick}}}{\partial x^2} = -c^2 A k^2 sin(wt - kx)$, the forcing function g(x, t) was calculated to be:

$$g(x,t) = c^2 A k^2 sin(wt - kx) - c^2 w^2 sin(wt - kx), since$$

With initial conditions being

$$u_{pick}(x, t = 0) = A \cdot sin(-kx)$$

$$\frac{\partial u_{pick}}{\partial t} = w \cdot A \cdot cos(-kx) \quad \text{at t=0}$$

and the dirichlet boundary conditions as

$$u_{pick}(x = 0, t) = A \cdot sin(wt)$$

$$u_{pick}(x = L, t) = A \cdot cos(wt - kL)$$

Using our method implementation and state space from (16), the above system of equation for the harmonic wave was implemented for the following values

$$a = 0 \quad b = 1$$

$$w = 3.25 \quad \text{Frequency}$$

$$k = 7.5 \quad \text{Wave Number}$$

$$c = w/k \quad \text{Wave Speed (km/s)}$$

$$Amp = 15 \quad \text{Amplitude (km)}$$

$$ln = b - a \quad \text{The domain of x (km)}$$

Using the above values, the following plots were obtained

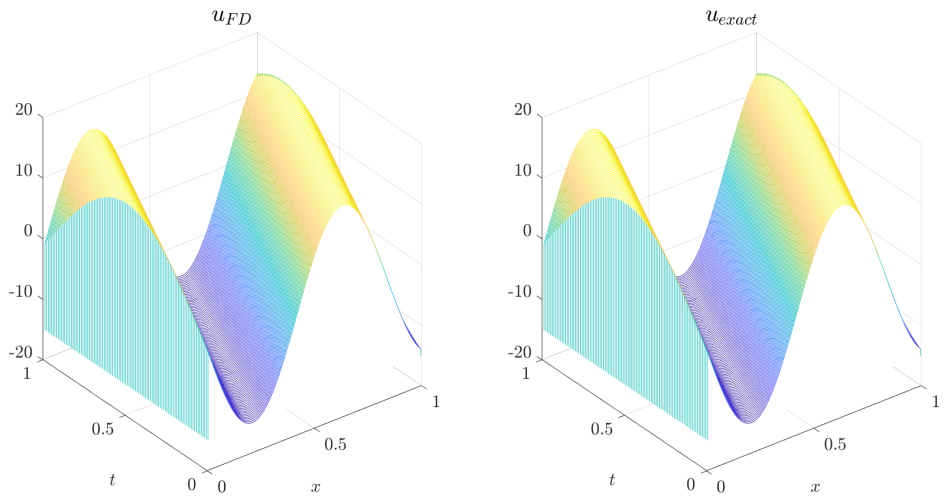


Figure 2: Waterfall Plot for Harmonic Wave

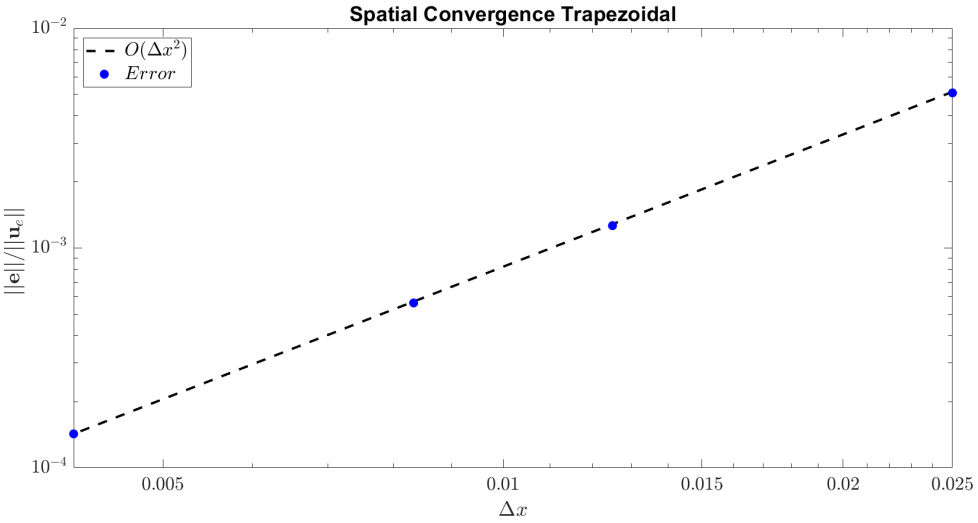


Figure 3: Spatial Convergence for Harmonic Wave

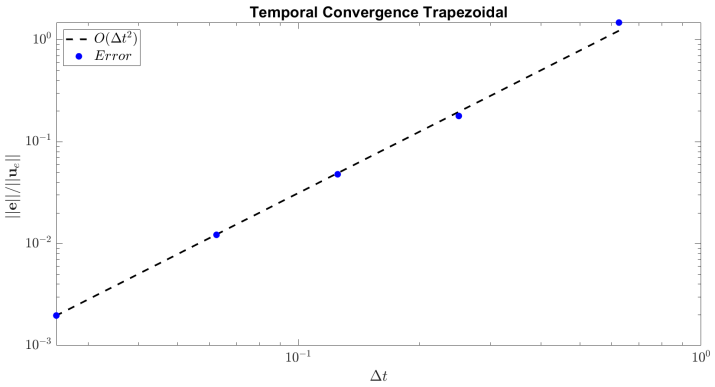


Figure 4: Temporal Convergence for Harmonic Wave

As it can seen in the plots above, we get the same waterfall plot using the exact solution and our finite difference solution. Further, it can be seen that both the spatial convergence and the Temporal convergence are of 2nd Order, which is what we wanted for confirmation. Hence, we can safely say that the numerical method was implemented correctly.

4 Results

Once we confirmed our implementation of the method, we can move ahead with confidence and run our own simulation. The plots obtained from running the simulation are shown below

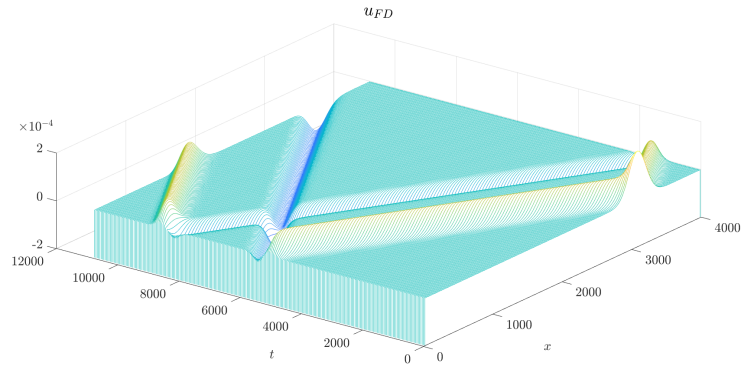
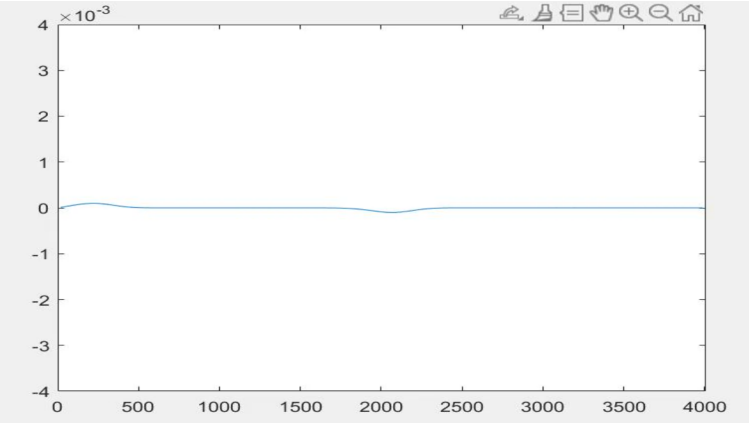


Figure 5: Waterfall Plot for Seismic wave of a Flat Earth

As it can be seen in the plot, as the wave propagates through time, it crosses the domain until it hits the boundary, which we had assumed to be the edge of the Flat Earth. Since we had theorized that the wave would just continue propagating on the lower surface of the Flat Earth, when the wave hits the boundary, it gets reflected and continues on its path in the reverse direction, due to our implementation of the 0 Dirichlet boundary conditions, which provide perfect reflection. Because we did not include dampening of the wave for simplicity, the amplitude of the wave remains constant, apart from the initial condition which has twice the amplitude due to superposition.

Further, the plots of the motion of wave were also animated and converted into a video file. The provided below is the Youtube video, and should be able play on Adobe Acrobat Reader. If not, the video can be found at link: https://youtu.be/eUqaZxLd6WA



As it can be in the video, the numerical method worked perfectly in modelling the seismic wave equation, which can be seen in the waterfall plot provided in Figure 5 as well.

5 Conclusions

To conclude, we were able to implement both the Finite Difference Method and Trapezoidal Time stepping correctly, as can be seen by the plots and video. Even though we got a good simulation, it can always be improved upon. For example, we can use a better time stepping method, such as RK4 Method, which known to be a much more accurate method and widely used. We had tried implementing the RK4 method, and while we were able to get the correct spatial convergence, we were not able to get the correct temporal convergence for Method of Manuactured Solutions, even if we were getting same results as trapezoidal method for watrerfall plots and wave video, and hence it was decided to leave this method out of the report (Though the code is still included for RK4 in Appendix).

Another way that this could be improved is by adding a dampening effect to simulate the actual propagation of a seismic wave, as any wave will lose energy over time, but for simplicity, was left out of our model.

6 Appendix

6.1 Main Code for Flat Earth

```
1 %% Trapezoidal
2
```

```
clear all, close all, clc

% parameters for problem (Distance Reduced by a Factor of 10)
a = 0; % Lower Bound x (km)
b = 40075/10; % Upper Bound x (km)
c = 5.86/10; % Wave Speed (km/s)
A = 0.002/10; % Amplitude (km)
ln = b-a; % The domian of x (km)
T = 60*60*3; % Final time to run to (seconds)
dt = 1; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence


% Initial Conditions, Gaussian Function
std = 87.5; % Roundness of the waveform
u_init = @(x) A*exp(-((x-(b-(9250/10)))/(2*std)).^2); % Initial Displacement
v_init = @(x) 0.*x; % Initial Velocity

%g(x,t) and Boundary Conditions
bc_a = @(t) 0; % Dirichlet Boundary Condition at 'a'
bc_b = @(t) 0; % Dirichlet Boundary Condition at 'b'
fcn = @(x,t) zeros(length(x),1); % Forcing Function

%# of n points to use
nvect = [200];

tic
for j = 1 : length( nvect )

    n = nvect(j);

    %---Build n, xj points, A matrix and g vector

    %Build interp points
    xj = ( a + (b-a)*(0:n)./n )';

    %grid spacing (uniform)
    dx = (b-a)/n;

    %Build A matrix
    %Use truncated version from lecture notes
    A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
         1*diag( ones(n-2,1),1) );

    % Matrix of Zeros of size A_temp
    Zeros = zeros(size(A_temp));
```

```
48
49      % The expanded A Matrix for 2nd degree differential equation
50      A = [Zeros eye(size(A_temp)); A_temp Zeros];
51
52      % Identity matrix of size A
53      I = eye(size(A));
54
55      %Build g for this set of xj
56      g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
57      g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
            zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
58      g = @(t) g1(t) + g2(t);
59
60      %Build RHS for IVP, f(u,t)
61      f = @(u,t) A*u + g(t);
62      %---
63
64      %---Initialize for time stepping
65      uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
66      tk = 0;
67      tvect = dt : dt : T;
68
69      %# snapshots to save
70      nsnps = 250;
71      ind = max( 1, round(length(tvect)/nsnps) );
72      tsv = tvect( 1 : ind :  end );
73
74      u = zeros( 2*n-2, length(tsv));
75
76      cnt = 1;
77      %---
78
79      %---Do time stepping
80      for jj = 1 : length( tvect )
81
82          tkp1 = tk + dt;
83
84          %Update solution at next time using trap method
85          ukp1 = (((I-dt/2*A)) \ (uk + dt/2*( f(uk,tk)  + g(tkp1) )));
86
87          uk = ukp1;
88          tk = tkp1;
89
90          if min(abs( tkp1-tsv ) ) < 1e-8
91
92              u(:,cnt) = uk;
93              cnt = cnt + 1;
```

```
94
95          end
96
97      end
98
99  end
100 toc
101
102 [X,T] = meshgrid( xj(2:end-1), tsv );
103 U = u(1:n-1,:).';
104
105 %--Waterfall plot of solns
106 figure(1)
107 waterfall( X,T,U ), hold on
108 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
109 title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
110 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
111 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
112 hold off
113
114 % Movie for the wave travel
115 figure(2)
116 for j = 1:nsnps
117     plot(X(1,:),U(j,:));
118     xlim([a b]);ylim([-0.04/10 0.04/10]);
119     F(j) = getframe(gcf) ;
120     drawnow
121 end
122
123 % save video
124 writerObj = VideoWriter('SeismicWave.avi');
125 % set the seconds per image
126 writerObj.FrameRate = 10;
127 % open the video writer
128 open(writerObj);
129
130 % write the frames to the video
131 for j = 1:length(F)
132     % convert the image to a frame
133     frame = F(j) ;
134     writeVideo(writerObj, frame);
135 end
136
137 % close the writer object
138 close(writerObj);
139
140 % Snapshots of the wave at different times
```

```
figure(3)
subplot(4,1,1); hold on;
H = area(X(1,:),U(1,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
title('Wave Snapshots at different time intervals');

subplot(4,1,2);
H = area(X(1,:),U(83,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');

subplot(4,1,3);
H = area(X(1,:),U(167,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');

subplot(4,1,4);
H = area(X(1,:),U(250,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');

hold off

%% RK4

clear all; close all; clc

% parameters for problem (Distance Reduced by a Factor of 10)
a = 0; % Lower Bound x (km) (Distance Reduced by a Factor of 10)
b = 40075/10; % Upper Bound x (km)
c = 5.86/10; % Wave Speed (km/s)
A = 0.002/10; % Amplitude (km)
ln = b-a; % The domian of x (km)
T = 60*60*3; % Final time to run to (seconds)
```

```
dt = 1; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence


% Initial Conditions, Gaussian Function
std = 87.5; % Roundness of the waveform
% u_init = @(x) A*exp(-((x-(a+b)/2)/(std)).^2); % Initial Displacement
u_init = @(x) A*exp(-((x-(b-(9250/10)))/(2*std)).^2); % Initial Displacement
v_init = @(x) 0.*x; % Initial Velocity

%g(x,t) and Boundary Conditions
bc_a = @(t) 0; % Dirichlet Boundary Condition at 'a'
bc_b = @(t) 0; % Dirichlet Boundary Condition at 'b'
fcn = @(x,t) zeros(length(x),1); % Forcing Function

%# of n points to use
nvect = [200];

tic
for j = 1 : length( nvect )

    n = nvect(j);

    %---Build n, xj points, A matrix and g vector

    %Build interp points
    xj = ( a + (b-a)*(0:n)./n )';

    %grid spacing (uniform)
    dx = (b-a)/n;

    %Build A matrix
    %Use truncated version from lecture notes
    A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
        1*diag( ones(n-2,1),1) );

    % Matrix of Zeros of size A_temp
    Zeros = zeros(size(A_temp));

    % The expanded A Matrix for 2nd degree differential equation
    A = [Zeros eye(size(A_temp)); A_temp Zeros];

    % Identity matrix of size A
    I = eye(size(A));

    %Build g for this set of xj
```

```
    g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
    g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
        zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
    g = @(t) g1(t) + g2(t);

    %Build RHS for IVP, f(u,t)
    f = @(u,t) A*u + g(t);
    %---

    %---Initialize for time stepping
    uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
    tk = 0;
    tvect = dt : dt : T;

    %# snapshots to save
    nsnps = 250;
    ind = max( 1, round(length(tvect)/nsnps) );
    tsv = tvect( 1 : ind :  end );

    u = zeros( n-1, length(tsv));

    cnt = 1;
    %---

    %---Do time stepping
    for jj = 1:length(tvect)
        tkp1 = tk + dt;

        y1 = f(uk, tk);
        y2 = f(uk + 0.5*dt*y1, tk);
        y3 = f(uk + 0.5*dt*y2, tk);
        y4 = f(uk + dt*y3, tk);

        ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);

        uk = ukp1;
        tk = tkp1;

        %Again, leave this. It sets things up to only save for a relatively
        %small # of times.
        if min(abs( tkp1-tsv ) ) < 1e-8

            u(:,cnt) = uk(1:n-1);
            cnt = cnt + 1;

        end

```

```
    end
end
toc

[X,T] = meshgrid( xj(2:end-1), tsv );

%--Waterfall plot of solns
figure(1)
waterfall( X,T, u.' ), hold on
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
hold off

% Movie for the wave travel
figure(2)
for j = 1:nsnps
    U = u.';
    plot(X(1,:),U(j,:));
    xlim([a b]);ylim([-0.04/10 0.04/10]);
    drawnow
end

% Snapshots of the wave at different times
figure(3)
subplot(4,1,1); hold on;
H = area(X(1,:),U(1,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
title('Wave Snapshots at different time intervals');

subplot(4,1,2);
H = area(X(1,:),U(83,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');

subplot(4,1,3);
H = area(X(1,:),U(167,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
```

```
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');

subplot(4,1,4);
H = area(X(1,:),U(250,:));
set(H,'FaceColor',[1 0 0]);
xlim([a b]);
ylim([-0.04/100 0.04/100]);
xlabel('Distance in Kilometers');
% title('Wave Snapshots at different intervals');
```

6.2 Method of Manufactured Solutions Code

```
%% Spatial Convergence MMS Trapezoidal

clear all, close all, clc

%params for problem
a = 0; % Lower Bound x (km)
b = 1; % Upper Bound x (km)
w = 3.25; % Frequency
k = 7.5; % Wave Number
c = w/k; % Wave Speed (km/s)
Amp = 15; % Amplitude (km)
ln = b-a; % The domian of x (km)
T = 1; % Final time to run to (seconds)
dt = 1e-3; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence

%Initial Conditions, Gaussian Function
uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
u_init = @(x) uex(x,0); % Initial Displacement
v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity

%g(x,t) and Boundary Conditions
bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
    Function

%# of n points to use
nvect = [40; 80; 120; 240];

%initialize error
err = zeros( size( nvect ) );
```

```
31
32 tic
33 for j = 1 : length( nvect )
34
35     n = nvect(j);
36
37     %---Build n, xj points, A matrix and g vector
38
39     %Build interp points
40     xj = ( a + (b-a)*(0:n)./n )';
41
42     %grid spacing (uniform)
43     dx = (b-a)/n;
44
45     %Build A matrix
46     %Use truncated version from lecture notes
47     A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
           + ...
48          1*diag( ones(n-2,1),1) );
49
50     % Matrix of Zeros of size A_temp
51     Zeros = zeros(size(A_temp));
52
53     % The expanded A Matrix for 2nd degree differential equation
54     A = [Zeros eye(size(A_temp)); A_temp Zeros];
55
56     % Identity matrix of size A
57     I = eye(size(A));
58
59     %Build g for this set of xj
60     g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
61     g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
           zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
62     g = @(t) g1(t) + g2(t);
63
64     %Build RHS for IVP, f(u,t)
65     f = @(u,t) A*u + g(t);
66     %---
67
68     %---Initialize for time stepping
69     uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
70     tk = 0;
71     tvect = dt : dt : T;
72
73     %# snapshots to save
74     nsnps = 250;
75     ind = max( 1, round(length(tvect)/nsnps) );
```

```
    tsv = tvect( 1 : ind :  end );

    u = zeros( 2*n-2, length(tsv));

    cnt = 1;
    %---

    %---Do time stepping
    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        %Update solution at next time using trap method
        ukp1 = (((I-dt/2*A)) \ (uk + dt/2*( f(uk,tk)  + g(tkp1) )));

        uk = ukp1;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8

            u(:,cnt) = uk;
            cnt = cnt + 1;

        end

    end
    err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
end
toc

[X,T] = meshgrid( xj(2:end-1), tsv );
U = u(1:n-1,:).';

% -Waterfall plot of solns

figure(1)
subplot(1,2,1)
waterfall( X,T,U )
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')

subplot(1,2,2)
waterfall( X,T, uex(X,T) )
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
```

```
title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
%

%--Error plots
figure(2)
c1 = err(end)/(dx^2);
loglog( ln./nvect, c1*(ln./nvect).^2, 'k--', 'linewidth', 2 ), hold on
title('Spatial Convergence Trapezoidal')

%plot err
loglog( ln./nvect, err , 'b.', 'markersize', 26 )
h = legend('$O(\Delta x^2)$', '$Error$');
set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
%make pretty
xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', ...
    'fontsize', 16)
set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )


%% Temporial Convergence MMS Trapezoidal

clear all, close all, clc

%params for problem
a = 0; % Lower Bound x (km)
b = 1; % Upper Bound x (km)
w = 3.25; % Frequency
k = 7.5; % Wave Number
c = w/k; % Wave Speed (km/s)
Amp = 15; % Amplitude (km)
ln = b-a; % The domian of x (km)
T = 1; % Final time to run to (seconds)
dt = 1e-3; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence
n = 1000;

%Initial Conditions, Gaussian Function
uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
u_init = @(x) uex(x,0); % Initial Displacement
v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity

%g(x,t) and Boundary Conditions
bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
```

```
fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
    Function

%# of n points to use
dtvect = [2.5; 1; 5e-1; 2.5e-1; 1e-1]*0.25; % From HW8

%initialize error
err = zeros(size(dtvect));

tic
for j = 1:length(dtvect)

    dt = dtvect(j);

    %Build interp points
    xj = ( a + (b-a)*(0:n)./n )';

    %grid spacing (uniform)
    dx = (b-a)/n;

    %Build A matrix
    %Use truncated version from lecture notes
    A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
        1*diag( ones(n-2,1),1) );

    % Matrix of Zeros of size A_temp
    Zeros = zeros(size(A_temp));

    % The expanded A Matrix for 2nd degree differential equation
    A = [Zeros eye(size(A_temp)); A_temp Zeros];

    % Identity matrix of size A
    I = eye(size(A));

    %Build g for this set of xj
    g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
    g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
        zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
    g = @(t) g1(t) + g2(t);

    %Build RHS for IVP, f(u,t)
    f = @(u,t) A*u + g(t);
    %---

    %---Initialize for time stepping
    uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
```

```
    tk = 0;
    tvect = dt : dt : T;

    %# snapshots to save
    nsnps = 250;
    ind = max( 1, round(length(tvect)/nsnps) );
    tsv = tvect( 1 : ind :  end );

    u = zeros( 2*n-2, length(tsv));

    cnt = 1;
    %---

    %---Do time stepping
    for jj = 1 : length( tvect )

        tkp1 = tk + dt;

        %Update solution at next time using trap method
        ukp1 = (((I-dt/2*A)) \ (uk + dt/2*( f(uk,tk)  + g(tkp1) )));

        uk = ukp1;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8

            u(:,cnt) = uk;
            cnt = cnt + 1;

        end

    end
    err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
end
toc

%Error plots
figure(3)
c = err(end)*(1./dt^2);
loglog( dtvect, c*(dtvect).^2, 'k--', 'linewidth', 2 ), hold on

%plot err
loglog(dtvect, err , 'b.', 'markersize', 26 )
title('Temporal Convergence Trapezoidal')

%make pretty
```

```
h = legend('$O(\Delta t^2)$', '$Error$');
set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', ...
    'fontsize', 16)
set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )

%% Spatial Convergence MMS RK4

clear all, close all, clc

%params for problem
a = 0; % Lower Bound x (km)
b = 1; % Upper Bound x (km)
w = 3.25; % Frequency
k = 7.5; % Wave Number
c = w/k; % Wave Speed (km/s)
Amp = 15; % Amplitude (km)
ln = b-a; % The domian of x (km)
T = 1; % Final time to run to (seconds)
dt = 1e-5; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence

%Initial Conditions, Gaussian Function
uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
u_init = @(x) uex(x,0); % Initial Displacement
v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity

%g(x,t) and Boundary Conditions
bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
    Function

%# of n points to use
nvect = [40; 80; 120; 240];

%initialize error
err = zeros( size( nvect ) );

tic
for j = 1 : length( nvect )

    n = nvect(j);

    %---Build n, xj points, A matrix and g vector

```

```
301     %Build interp points
302     xj = ( a + (b-a)*(0:n)./n )';
303
304     %grid spacing (uniform)
305     dx = (b-a)/n;
306
307     %Build A matrix
308     %Use truncated version from lecture notes
309     A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
            + ...
310          1*diag( ones(n-2,1),1) );
311
312     % Matrix of Zeros of size A_temp
313     Zeros = zeros(size(A_temp));
314
315     % The expanded A Matrix for 2nd degree differential equation
316     A = [Zeros eye(size(A_temp)); A_temp Zeros];
317
318     % Identity matrix of size A
319     I = eye(size(A));
320
321     %Build g for this set of xj
322     g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
323     g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
            zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
324     g = @(t) g1(t) + g2(t);
325
326     %Build RHS for IVP, f(u,t)
327     f = @(u,t) A*u + g(t);
328     %---
329
330     %---Initialize for time stepping
331     uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
332     tk = 0;
333     tvect = dt : dt : T;
334
335     %# snapshots to save
336     nsnps = 250;
337     ind = max( 1, round(length(tvect)/nsnps) );
338     tsv = tvect( 1 : ind :  end );
339
340     u = zeros( n-1, length(tsv));
341
342     cnt = 1;
343     %---
344
345     %---Do time stepping
```

```
    for jj = 1:length(tvect)

        tkp1 = tk + dt;

        y1 = f(uk, tk);
        y2 = f(uk + 0.5*dt*y1, tk);
        y3 = f(uk + 0.5*dt*y2, tk);
        y4 = f(uk + dt*y3, tk);

        ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);

        uk = ukp1;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8

            u(:,cnt) = uk(1:n-1);
            cnt = cnt + 1;
        end

    end
    %---
    err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
end
toc

[X,T] = meshgrid( xj(2:end-1), tsv );

% -Waterfall plot of solns

figure(1)
subplot(1,2,1)
waterfall( X,T, u.' )
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')

subplot(1,2,2)
waterfall( X,T, uex(X,T) )
set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
%
```

```
392 %--Error plots
393 figure(2)
394 c1 = err(end)/(dx^2);
395 loglog( ln./nvect, c1*(ln./nvect).^2, 'k--', 'linewidth', 2 ), hold on
396 title('Spatial Convergence RK4')
397
398 %plot err
399 loglog( ln./nvect, err , 'b.', 'markersize', 26 )
400 h = legend('$O(\Delta x^2)$', '$Error$');
401 set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
402 %make pretty
403 xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
404 ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', ...
        'fontsize', 16)
405 set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
406
407 %% Temporial Convergence MMS RK4
408
409 clear all, close all, clc
410
411 %params for problem
412 a = 0; % Lower Bound x (km)
413 b = 1; % Upper Bound x (km)
414 w = 3.25; % Frequency
415 k = 7.5; % Wave Number
416 c = w/k; % Wave Speed (km/s)
417 Amp = 15; % Amplitude (km)
418 ln = b-a; % The domian of x (km)
419 T = 1; % Final time to run to (seconds)
420 dt = 1e-3; % Make dt small in spatial convergence test so that time error ...
        doesn't pollute convergence
421 n = 5000;
422
423 %Initial Conditions, Gaussian Function
424 uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
425 u_init = @(x) uex(x,0); % Initial Displacement
426 v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity
427
428 %g(x,t) and Boundary Conditions
429 bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
430 bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
431 fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
        Function
432
433 %# of n points to use
434 dtvect = [2.5; 1; 5e-1; 2.5e-1; 1e-1]*0.25; % From HW8
435
```

```
%initialize error
err = zeros(size(dtvect));

tic
for j = 1:length(dtvect)

    dt = dtvect(j);

    %Build interp points
    xj = ( a + (b-a)*(0:n)./n )';

    %grid spacing (uniform)
    dx = (b-a)/n;

    %Build A matrix
    %Use truncated version from lecture notes
    A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
        1*diag( ones(n-2,1),1) );

    % Matrix of Zeros of size A_temp
    Zeros = zeros(size(A_temp));

    % The expanded A Matrix for 2nd degree differential equation
    A = [Zeros eye(size(A_temp)); A_temp Zeros];

    % Identity matrix of size A
    I = eye(size(A));

    %Build g for this set of xj
    g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
    g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
        zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
    g = @(t) g1(t) + g2(t);

    %Build RHS for IVP, f(u,t)
    f = @(u,t) A*u + g(t);
    %---

    %---Initialize for time stepping
    uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
    tk = 0;
    tvect = dt : dt : T;

    %# snapshots to save
    nsnps = 250;
    ind = max( 1, round(length(tvect)/nsnps) );
```

```
    tsv = tvect( 1 : ind :  end );

    u = zeros( n-1, length(tsv));

    cnt = 1;
    %---

    %---Do time stepping
    for jj = 1:length(tvect)

        tkp1 = tk + dt;

        y1 = f(uk, tk);
        y2 = f(uk + 0.5*dt*y1, tk);
        y3 = f(uk + 0.5*dt*y2, tk);
        y4 = f(uk + dt*y3, tk);

        ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);

        uk = ukp1;
        tk = tkp1;

        if min(abs( tkp1-tsv ) ) < 1e-8

            u(:,cnt) = uk(1:n-1);
            cnt = cnt + 1;

        end

    end
    %---
    err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
end
toc

%Error plots
figure(3)
c = err(end)*(1./dt^4);
loglog( dtvect, c*(dtvect).^4, 'k--', 'linewidth', 2 ), hold on

%plot err
loglog(dtvect, err , 'b.', 'markersize', 26 )
xlim([1e-2 100])
title('Temporal Convergence RK4')

%make pretty
```

```
527 h = legend('$O(\Delta t^2)$', '$Error$');
528 set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
529 xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
530 ylabel( '$||\textbf{e}||/||\textbf{u}_e||$ ', 'interpreter', 'latex', ...
        'fontsize', 16)
531 set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
```