

# 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<sup>1</sup>.

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<sup>2</sup>.

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<sup>3</sup>. 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.

---

<sup>1</sup>[https://en.wikipedia.org/wiki/Wave\\_equation](https://en.wikipedia.org/wiki/Wave_equation)

<sup>2</sup><https://www.dosomething.org/us/facts/11-facts-about-earthquakes>

<sup>3</sup><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<sup>4</sup>. P-waves are pressure waves and propagate by alternating compressions and rarefactions<sup>5</sup>. The Secondary waves are known as S-waves, and are shear waves. S-waves are slower than 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  $\lambda$  (Elastic Modulus) and  $\mu$  (Shear Modulus) are known as Lamé parameters, and  $\rho$  is the density of medium. Since we're only focusing on S-waves, we just need to know  $\rho$  and  $\mu$ . After doing some research, we found that silicon density is  $2.329 \text{ g/cm}^3$ <sup>6</sup> and the shear modulus for silicon ranges from 51-80 Gpa<sup>7</sup>. Using  $\rho = 2329 \text{ kg/m}^3$  and  $\mu = 80 \cdot 10^9 \text{ pa}$ , the wave speed was found to be  $5.86 \text{ km/s}$

## 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<sup>8</sup>.

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

---

<sup>4</sup><https://en.wikipedia.org/wiki/P-wave>

<sup>5</sup><https://en.wikipedia.org/wiki/P-wave>

<sup>6</sup><https://en.wikipedia.org/wiki/Silicon>

<sup>7</sup><https://en.wikipedia.org/wiki/Silicon>

<sup>8</sup>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  $\sigma$  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.  $\sigma$  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.

$$\begin{aligned} u(x = 0, t) &= 0 \\ u(x = L, t) &= 0 \end{aligned}$$

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

<sup>9</sup> 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:

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

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  $x_j$ ):

$$\{x_{j-p/2}, \dots, x_j, \dots, x_{j+p/2}\} \quad (2)$$

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

$$\{x_{j-1}, x_j, x_{j+1}\} \quad (3)$$

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

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

To simplify the above into an initial value problem, we represent the spatial variable  $x$  by the  $n+1$  points,  $\{x_1, x_2, \dots, x_{n+1}\}$ , 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,

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

Noticing that

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

---

<sup>9</sup>HW8 and Week 13 Typed Notes

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

Equation (5) can be further simplified into

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

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

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

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

$$\begin{aligned} \ddot{u}_j(t) &= c^2 \left[ u_{j-1} \frac{d^2 L_{j-1}^{(j)}}{dx^2} \Big|_{x=x_j} + u_j \frac{d^2 L_j^{(j)}}{dx^2} \Big|_{x=x_j} + u_{j+1} \frac{d^2 L_{j+1}^{(j)}}{dx^2} \right] + g(x_j, t) \quad (j = 2, \dots, n) \\ \implies \ddot{u}_j(t) &= \frac{c^2}{\Delta x^2} \left[ u_{j-1} - 2u_j + u_{j+1} \right] + g(x_j, t) \quad (j = 2, \dots, n) \end{aligned} \quad (10)$$

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

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

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

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  $\ddot{u} = A_{temp} \cdot u_{temp} + g_{temp}$

$$\begin{bmatrix} \ddot{u}_2 \\ \ddot{u}_3 \\ \vdots \\ \ddot{u}_{n-1} \\ \ddot{u}_n \end{bmatrix} = \frac{c^2}{\Delta x^2} \begin{bmatrix} -2 & 1 & 0 & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -2 \end{bmatrix} \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} + \begin{bmatrix} g(x_2, t) + \frac{c^2}{\Delta x^2} g_a(t) \\ g(x_3, t) \\ \vdots \\ g(x_{n-1}, t) \\ g(x_n, t) + \frac{c^2}{\Delta x^2} g_b(t) \end{bmatrix} \quad (13)$$

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

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

$$\dot{u}_j = v_j \quad (14)$$

$$\dot{v}_j = Au_j + g_j \quad (15)$$

The above can be represented as

$$\begin{bmatrix} \dot{u}_j \\ \dot{v}_j \end{bmatrix} = \begin{bmatrix} 0 & I \\ A_{temp} & 0 \end{bmatrix} \begin{bmatrix} u_j \\ v_j \end{bmatrix} + \begin{bmatrix} 0 \\ g_j \end{bmatrix} \quad (16)$$

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

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

## Trapezoid Method Derivation

<sup>10</sup>

Having an initial value problem as

$$\dot{u} = f(u, t) \quad (17)$$

$$u(t_0) = u_0 \quad (18)$$

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

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

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:

$$f(u(t), t) \approx f(u(t_k), t_k) \left( \frac{t - t_{k+1}}{t_k - t_{k+1}} \right) + f(u(t_{k+1}), t_{k+1}) \left( \frac{t - t_k}{t_{k+1} - t_k} \right) \quad (20)$$

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

$$u(t_{k+1}) - u(t_k) \approx \int_{t_k}^{t_{k+1}} f(u(t_k), t_k) \left( \frac{t - t_{k+1}}{-\Delta t} \right) + f(u(t_{k+1}), t_{k+1}) \left( \frac{t - t_k}{\Delta t} \right) dt \quad (21)$$

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  $\Delta t$ , we can pull these terms out of the integral to get

---

<sup>10</sup>HW5 In Class Solutions

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

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

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

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

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

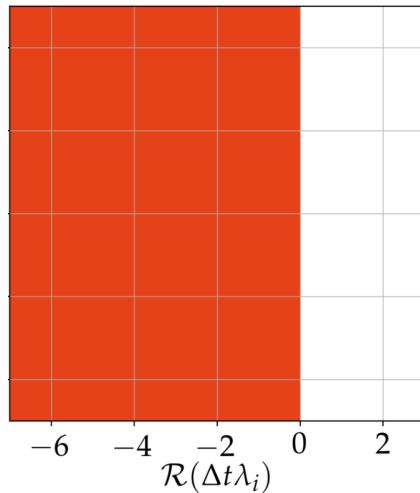


Figure 1: Stability Plot for Trapezoidal Numerical Method



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

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

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

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

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

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

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

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

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

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

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

Using the above values, the following plots were obtained

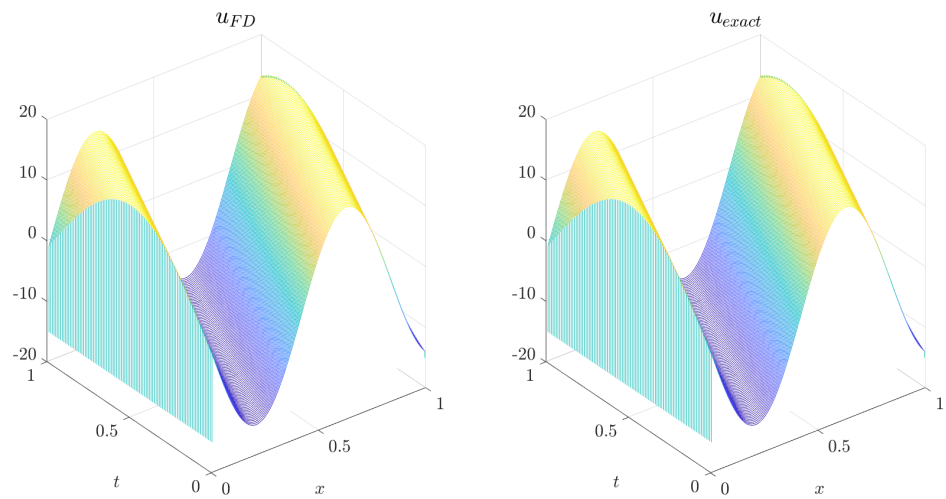


Figure 2: Waterfall Plot for Harmonic Wave

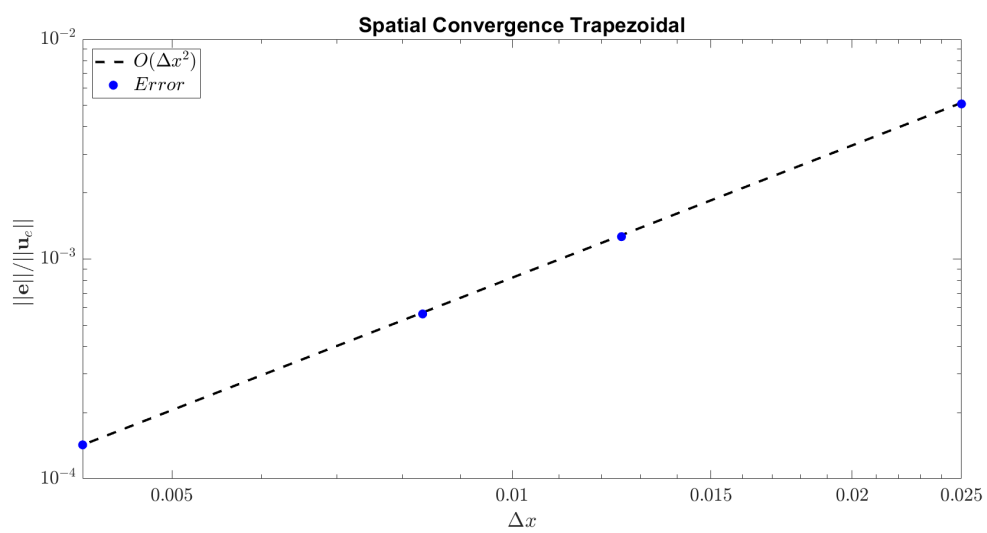


Figure 3: Spatial Convergence for Harmonic Wave

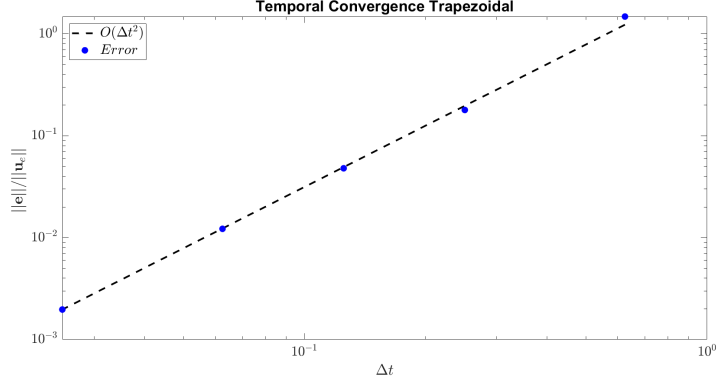


Figure 4: Temporal Convergence for Harmonic Wave

As it can be 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

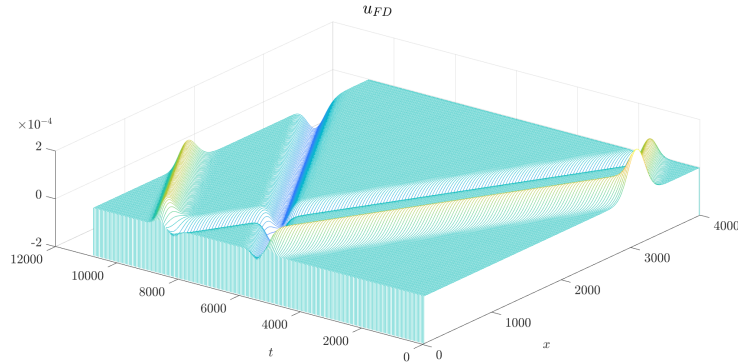
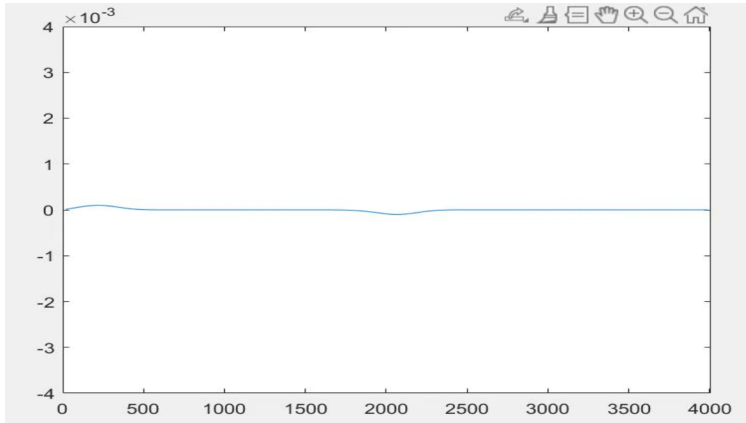


Figure 5: Waterfall Plot for Seismic wave of a Flat Earth

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 Manufactured Solutions, even if we were getting same results as trapezoidal method for waterfall 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
```

```

3 clear all, close all, clc
4
5 % parameters for problem (Distance Reduced by a Factor of 10)
6 a = 0; % Lower Bound x (km)
7 b = 40075/10; % Upper Bound x (km)
8 c = 5.86/10; % Wave Speed (km/s)
9 A = 0.002/10; % Amplitude (km)
10 ln = b-a; % The domain of x (km)
11 T = 60*60*3; % Final time to run to (seconds)
12 dt = 1; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence
13
14
15 % Initial Conditions, Gaussian Function
16 std = 87.5; % Roundness of the waveform
17 u_init = @(x) A*exp(-((x-(b-(9250/10)))/(2*std)).^2); % Initial Displacement
18 v_init = @(x) 0.*x; % Initial Velocity
19
20 %g(x,t) and Boundary Conditions
21 bc_a = @(t) 0; % Dirichlet Boundary Condition at 'a'
22 bc_b = @(t) 0; % Dirichlet Boundary Condition at 'b'
23 fcn = @(x,t) zeros(length(x),1); % Forcing Function
24
25 %# of n points to use
26 nvect = [200];
27
28 tic
29 for j = 1 : length( nvect )
30
31     n = nvect(j);
32
33     %---Build n, xj points, A matrix and g vector
34
35     %Build interp points
36     xj = ( a + (b-a)*(0:n)./n )';
37
38     %grid spacing (uniform)
39     dx = (b-a)/n;
40
41     %Build A matrix
42     %Use truncated version from lecture notes
43     A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
44         1*diag( ones(n-2,1),1) );
45
46     % Matrix of Zeros of size A_temp
47     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); ...
58           zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
59 g = @(t) g1(t) + g2(t);
60
61 %Build RHS for IVP, f(u,t)
62 f = @(u,t) A*u + g(t);
63 %---
64
65 %---Initialize for time stepping
66 uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];
67 tk = 0;
68 tvect = dt : dt : T;
69
70 %# snapshots to save
71 nsmps = 250;
72 ind = max( 1, round(length(tvect)/nsmps) );
73 tsv = tvect( 1 : ind : end );
74
75 u = zeros( 2*n-2, length(tsv));
76
77 cnt = 1;
78 %---
79
80 %---Do time stepping
81 for jj = 1 : length( tvect )
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

```

```

141 figure(3)
142 subplot(4,1,1); hold on;
143 H = area(X(1,:),U(1,:));
144 set(H,'FaceColor',[1 0 0]);
145 xlim([a b]);
146 ylim([-0.04/100 0.04/100]);
147 xlabel('Distance in Kilometers');
148 title('Wave Snapshots at different time intervals');
149
150 subplot(4,1,2);
151 H = area(X(1,:),U(83,:));
152 set(H,'FaceColor',[1 0 0]);
153 xlim([a b]);
154 ylim([-0.04/100 0.04/100]);
155 xlabel('Distance in Kilometers');
156 % title('Wave Snapshots at different intervals');
157
158 subplot(4,1,3);
159 H = area(X(1,:),U(167,:));
160 set(H,'FaceColor',[1 0 0]);
161 xlim([a b]);
162 ylim([-0.04/100 0.04/100]);
163 xlabel('Distance in Kilometers');
164 % title('Wave Snapshots at different intervals');
165
166 subplot(4,1,4);
167 H = area(X(1,:),U(250,:));
168 set(H,'FaceColor',[1 0 0]);
169 xlim([a b]);
170 ylim([-0.04/100 0.04/100]);
171 xlabel('Distance in Kilometers');
172 % title('Wave Snapshots at different intervals');
173
174 hold off
175
176 %% RK4
177
178 clear all; close all; clc
179
180 % parameters for problem (Distance Reduced by a Factor of 10)
181 a = 0; % Lower Bound x (km) (Distance Reduced by a Factor of 10)
182 b = 40075/10; % Upper Bound x (km)
183 c = 5.86/10; % Wave Speed (km/s)
184 A = 0.002/10; % Amplitude (km)
185 ln = b-a; % The domain of x (km)
186 T = 60*60*3; % Final time to run to (seconds)

```



```

187 dt = 1; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence
188
189
190 % Initial Conditions, Gaussian Function
191 std = 87.5; % Roundness of the waveform
192 % u_init = @(x) A*exp(-((x-(a+b)/2)/(std)).^2); % Initial Displacement
193 u_init = @(x) A*exp(-((x-(b-(9250/10)))/(2*std)).^2); % Initial Displacement
194 v_init = @(x) 0.*x; % Initial Velocity
195
196 %g(x,t) and Boundary Conditions
197 bc_a = @(t) 0; % Dirichlet Boundary Condition at 'a'
198 bc_b = @(t) 0; % Dirichlet Boundary Condition at 'b'
199 fcn = @(x,t) zeros(length(x),1); % Forcing Function
200
201 %# of n points to use
202 nvect = [200];
203
204 tic
205 for j = 1 : length( nvect )
206
207     n = nvect(j);
208
209     %---Build n, xj points, A matrix and g vector
210
211     %Build interp points
212     xj = ( a + (b-a)*(0:n)./n )';
213
214     %grid spacing (uniform)
215     dx = (b-a)/n;
216
217     %Build A matrix
218     %Use truncated version from lecture notes
219     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) );
220
221
222     % Matrix of Zeros of size A_temp
223     Zeros = zeros(size(A_temp));
224
225     % The expanded A Matrix for 2nd degree differential equation
226     A = [Zeros eye(size(A_temp)); A_temp Zeros];
227
228     % Identity matrix of size A
229     I = eye(size(A));
230
231     %Build g for this set of xj

```

```

232 g1 = @(t) [zeros(size( xj(2:end-1) )); fcn(xj(2:end-1),t)];
233 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)];
234 g = @(t) g1(t) + g2(t);
235
236 %Build RHS for IVP, f(u,t)
237 f = @(u,t) A*u + g(t);
238 %---
239
240 %---Initialize for time stepping
241 uk = [u_init(xj(2:end-1)); v_init(xj(2:end-1))];
242 tk = 0;
243 tvect = dt : dt : T;
244
245 %# snapshots to save
246 nsmps = 250;
247 ind = max( 1, round(length(tvect)/nsmps) );
248 tsv = tvect( 1 : ind : end );
249
250 u = zeros( n-1, length(tsv));
251
252 cnt = 1;
253 %---
254
255 %---Do time stepping
256 for jj = 1:length(tvect)
257     tkp1 = tk + dt;
258
259     y1 = f(uk, tk);
260     y2 = f(uk + 0.5*dt*y1, tk);
261     y3 = f(uk + 0.5*dt*y2, tk);
262     y4 = f(uk + dt*y3, tk);
263
264     ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);
265
266     uk = ukp1;
267     tk = tkp1;
268
269     %Again, leave this. It sets things up to only save for a relatively
270     %small # of times.
271     if min(abs( tkp1-tsv ) ) < 1e-8
272
273         u(:,cnt) = uk(1:n-1);
274         cnt = cnt + 1;
275
276     end
277

```

```

278     end
279 end
280 toc
281
282 [X,T] = meshgrid( xj(2:end-1), tsv );
283
284 %--Waterfall plot of solns
285 figure(1)
286 waterfall( X,T, u.' ), hold on
287 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
288 title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
289 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
290 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
291 hold off
292
293 % Movie for the wave travel
294 figure(2)
295 for j = 1:nsnps
296     U = u.';
297     plot(X(1,:),U(j,:));
298     xlim([a b]);ylim([-0.04/10 0.04/10]);
299     drawnow
300 end
301
302 % Snapshots of the wave at different times
303 figure(3)
304 subplot(4,1,1); hold on;
305 H = area(X(1,:),U(1,:));
306 set(H,'FaceColor',[1 0 0]);
307 xlim([a b]);
308 ylim([-0.04/100 0.04/100]);
309 xlabel('Distance in Kilometers');
310 title('Wave Snapshots at different time intervals');
311
312 subplot(4,1,2);
313 H = area(X(1,:),U(83,:));
314 set(H,'FaceColor',[1 0 0]);
315 xlim([a b]);
316 ylim([-0.04/100 0.04/100]);
317 xlabel('Distance in Kilometers');
318 % title('Wave Snapshots at different intervals');
319
320 subplot(4,1,3);
321 H = area(X(1,:),U(167,:));
322 set(H,'FaceColor',[1 0 0]);
323 xlim([a b]);
324 ylim([-0.04/100 0.04/100]);

```

```

325 xlabel('Distance in Kilometers');
326 % title('Wave Snapshots at different intervals');
327
328 subplot(4,1,4);
329 H = area(X(1,:),U(250,:));
330 set(H,'FaceColor',[1 0 0]);
331 xlim([a b]);
332 ylim([-0.04/100 0.04/100]);
333 xlabel('Distance in Kilometers');
334 % title('Wave Snapshots at different intervals');

```

## 6.2 Method of Manufactured Solutions Code

```

1 %% Spatial Convergence MMS Trapezoidal
2
3 clear all, close all, clc
4
5 %params for problem
6 a = 0; % Lower Bound x (km)
7 b = 1; % Upper Bound x (km)
8 w = 3.25; % Frequency
9 k = 7.5; % Wave Number
10 c = w/k; % Wave Speed (km/s)
11 Amp = 15; % Amplitude (km)
12 ln = b-a; % The domain of x (km)
13 T = 1; % Final time to run to (seconds)
14 dt = 1e-3; % Make dt small in spatial convergence test so that time error ...
    doesn't pollute convergence
15
16 %Initial Conditions, Gaussian Function
17 uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
18 u_init = @(x) uex(x,0); % Initial Displacement
19 v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity
20
21 %g(x,t) and Boundary Conditions
22 bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
23 bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
24 fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
    Function
25
26 %# of n points to use
27 nvect = [40; 80; 120; 240];
28
29 %initialize error
30 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         + ...
49         1*diag( ones(n-2,1),1) );
50
51     % Matrix of Zeros of size A_temp
52     Zeros = zeros(size(A_temp));
53
54     % The expanded A Matrix for 2nd degree differential equation
55     A = [Zeros eye(size(A_temp)); A_temp Zeros];
56
57     % Identity matrix of size A
58     I = eye(size(A));
59
60     %Build g for this set of xj
61     g1 = @(t) [zeros(size( xj(2:end-1) )); fcn(xj(2:end-1),t)];
62     g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
63         zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
64     g = @(t) g1(t) + g2(t);
65
66     %Build RHS for IVP, f(u,t)
67     f = @(u,t) A*u + g(t);
68     %---
69
70     %---Initialize for time stepping
71     uk = [u_init(xj(2:end-1)); v_init(xj(2:end-1))];
72     tk = 0;
73     tvect = dt : dt : T;
74
75     %# snapshots to save
76     nsnps = 250;
77     ind = max( 1, round(length(tvect)/nsnps) );

```

```

76     tsv = tvect( 1 : ind : end );
77
78     u = zeros( 2*n-2, length(tsv));
79
80     cnt = 1;
81     %---
82
83     %---Do time stepping
84     for jj = 1 : length( tvect )
85
86         tkp1 = tk + dt;
87
88         %Update solution at next time using trap method
89         ukp1 = (((I-dt/2*A)) \ (uk + dt/2*( f(uk,tk) + g(tkp1) )));
90
91         uk = ukp1;
92         tk = tkp1;
93
94         if min(abs( tkp1-tsv ) ) < 1e-8
95
96             u(:,cnt) = uk;
97             cnt = cnt + 1;
98
99         end
100
101     end
102     err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
103 end
104 toc
105
106 [X,T] = meshgrid( xj(2:end-1), tsv );
107 U = u(1:n-1,:).';
108
109 % -Waterfall plot of solns
110
111 figure(1)
112 subplot(1,2,1)
113 waterfall( X,T,U )
114 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
115 title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
116 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
117 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
118
119 subplot(1,2,2)
120 waterfall( X,T, uex(X,T) )
121 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )

```

```

122 title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
123 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
124 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
125 %
126
127 %--Error plots
128 figure(2)
129 c1 = err(end)/(dx^2);
130 loglog( ln./nvect, c1*(ln./nvect).^2, 'k--', 'linewidth', 2 ), hold on
131 title('Spatial Convergence Trapezoidal')
132
133 %plot err
134 loglog( ln./nvect, err , 'b.', 'markersize', 26 )
135 h = legend('$0(\Delta x^2)$', '$Error$');
136 set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
137 %make pretty
138 xlabel( '$\Delta x$', 'interpreter', 'latex', 'fontsize', 16)
139 ylabel( '$||\textbf{e}||/||\textbf{u}_e||$', 'interpreter', 'latex', ...
        'fontsize', 16)
140 set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
141
142
143 %% Temporal Convergence MMS Trapezoidal
144
145 clear all, close all, clc
146
147 %params for problem
148 a = 0; % Lower Bound x (km)
149 b = 1; % Upper Bound x (km)
150 w = 3.25; % Frequency
151 k = 7.5; % Wave Number
152 c = w/k; % Wave Speed (km/s)
153 Amp = 15; % Amplitude (km)
154 ln = b-a; % The domain of x (km)
155 T = 1; % Final time to run to (seconds)
156 dt = 1e-3; % Make dt small in spatial convergence test so that time error ...
        doesn't pollute convergence
157 n = 1000;
158
159 %Initial Conditions, Gaussian Function
160 uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
161 u_init = @(x) uex(x,0); % Initial Displacement
162 v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity
163
164 %g(x,t) and Boundary Conditions
165 bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
166 bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'

```

```

167 fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
    Function
168
169 %# of n points to use
170 dtvect = [2.5; 1; 5e-1; 2.5e-1; 1e-1]*0.25; % From HW8
171
172 %initialize error
173 err = zeros(size(dtvect));
174
175 tic
176 for j = 1:length(dtvect)
177
178     dt = dtvect(j);
179
180     %Build interp points
181     xj = ( a + (b-a)*(0:n)./n )';
182
183     %grid spacing (uniform)
184     dx = (b-a)/n;
185
186     %Build A matrix
187     %Use truncated version from lecture notes
188     A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
        + ...
189         1*diag( ones(n-2,1),1) );
190
191     % Matrix of Zeros of size A_temp
192     Zeros = zeros(size(A_temp));
193
194     % The expanded A Matrix for 2nd degree differential equation
195     A = [Zeros eye(size(A_temp)); A_temp Zeros];
196
197     % Identity matrix of size A
198     I = eye(size(A));
199
200     %Build g for this set of xj
201     g1 = @(t) [zeros(size( xj(2:end-1) ));fcn(xj(2:end-1),t)];
202     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)];
203     g = @(t) g1(t) + g2(t);
204
205     %Build RHS for IVP, f(u,t)
206     f = @(u,t) A*u + g(t);
207     %---
208
209     %---Initialize for time stepping
210     uk = [u_init(xj(2:end-1));v_init(xj(2:end-1))];

```



```

211     tk = 0;
212     tvect = dt : dt : T;
213
214     %# snapshots to save
215     nsnps = 250;
216     ind = max( 1, round(length(tvect)/nsnps) );
217     tsv = tvect( 1 : ind : end );
218
219     u = zeros( 2*n-2, length(tsv));
220
221     cnt = 1;
222     %---
223
224     %---Do time stepping
225     for jj = 1 : length( tvect )
226
227         tkp1 = tk + dt;
228
229         %Update solution at next time using trap method
230         ukp1 = (((I-dt/2*A)) \ (uk + dt/2*( f(uk,tk) + g(tkp1) )));
231
232         uk = ukp1;
233         tk = tkp1;
234
235         if min(abs( tkp1-tsv ) ) < 1e-8
236
237             u(:,cnt) = uk;
238             cnt = cnt + 1;
239
240         end
241
242     end
243     err(jj) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
244 end
245 toc
246
247 %Error plots
248 figure(3)
249 c = err(end)*(1./dt^2);
250 loglog( dtvect, c*(dtvect).^2, 'k--', 'linewidth', 2 ), hold on
251
252 %plot err
253 loglog(dtvect, err , 'b.', 'markersize', 26 )
254 title('Temporal Convergence Trapezoidal')
255
256 %make pretty

```

```

257 h = legend('$0(\Delta t^2)$', '$Error$');
258 set(h, 'Interpreter','latex', 'fontsize', 16, 'Location', 'NorthWest' )
259 xlabel( '$\Delta t$', 'interpreter', 'latex', 'fontsize', 16)
260 ylabel( '$||\textbf{e}||/\textbf{u}_e||$', 'interpreter', 'latex', ...
        'fontsize', 16)
261 set(gca, 'TickLabelInterpreter','latex', 'fontsize', 16 )
262
263 %% Spatial Convergence MMS RK4
264
265 clear all, close all, clc
266
267 %params for problem
268 a = 0; % Lower Bound x (km)
269 b = 1; % Upper Bound x (km)
270 w = 3.25; % Frequency
271 k = 7.5; % Wave Number
272 c = w/k; % Wave Speed (km/s)
273 Amp = 15; % Amplitude (km)
274 ln = b-a; % The domain of x (km)
275 T = 1; % Final time to run to (seconds)
276 dt = 1e-5; % Make dt small in spatial convergence test so that time error ...
        doesn't pollute convergence
277
278 %Initial Conditions, Gaussian Function
279 uex = @(x,t) Amp*sin(w*t - k*x); % Exact Solution
280 u_init = @(x) uex(x,0); % Initial Displacement
281 v_init = @(x) w*Amp*cos(-k*x); % Initial Velocity
282
283 %g(x,t) and Boundary Conditions
284 bc_a = @(t) Amp*sin(w*t - k*a); % Dirichlet Boundary Condition at 'a'
285 bc_b = @(t) Amp*sin(w*t - k*b); % Dirichlet Boundary Condition at 'b'
286 fcn = @(x,t) c^2.*Amp.*k^2.*sin(w*t-k*x) - Amp.*w^2.*sin(w*t-k*x); % Forcing ...
        Function
287
288 %# of n points to use
289 nvect = [40; 80; 120; 240];
290
291 %initialize error
292 err = zeros( size( nvect ) );
293
294 tic
295 for j = 1 : length( nvect )
296
297     n = nvect(j);
298
299     %---Build n, xj points, A matrix and g vector
300

```

```

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         + ...
311         1*diag( ones(n-2,1),1) );
312
313 % Matrix of Zeros of size A_temp
314 Zeros = zeros(size(A_temp));
315
316 % The expanded A Matrix for 2nd degree differential equation
317 A = [Zeros eye(size(A_temp)); A_temp Zeros];
318
319 % Identity matrix of size A
320 I = eye(size(A));
321
322 %Build g for this set of xj
323 g1 = @(t) [zeros(size( xj(2:end-1) )); fcn(xj(2:end-1),t)];
324 g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
325         zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
326 g = @(t) g1(t) + g2(t);
327
328 %Build RHS for IVP, f(u,t)
329 f = @(u,t) A*u + g(t);
330 %---
331
332 %---Initialize for time stepping
333 uk = [u_init(xj(2:end-1)); v_init(xj(2:end-1))];
334 tk = 0;
335 tvect = dt : dt : T;
336
337 %# snapshots to save
338 nsmps = 250;
339 ind = max( 1, round(length(tvect)/nsmps) );
340 tsv = tvect( 1 : ind : end );
341
342 u = zeros( n-1, length(tsv));
343
344 cnt = 1;
345 %---
346 %---Do time stepping

```

```

346     for jj = 1:length(tvect)
347
348         tkp1 = tk + dt;
349
350         y1 = f(uk, tk);
351         y2 = f(uk + 0.5*dt*y1, tk);
352         y3 = f(uk + 0.5*dt*y2, tk);
353         y4 = f(uk + dt*y3, tk);
354
355         ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);
356
357         uk = ukp1;
358         tk = tkp1;
359
360         if min(abs( tkp1-tsv )) < 1e-8
361
362             u(:,cnt) = uk(1:n-1);
363             cnt = cnt + 1;
364         end
365
366     end
367     %---
368     err(j) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
        uex(xj(2:end-1),T) );
369 end
370 toc
371
372 [X,T] = meshgrid( xj(2:end-1), tsv );
373
374 % -Waterfall plot of solns
375
376 figure(1)
377 subplot(1,2,1)
378 waterfall( X,T, u.' )
379 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
380 title('$u_{FD}$', 'fontsize', 20, 'interpreter', 'latex')
381 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
382 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
383
384 subplot(1,2,2)
385 waterfall( X,T, uex(X,T) )
386 set( gca, 'fontsize', 15, 'ticklabelinterpreter', 'latex' )
387 title('$u_{exact}$', 'fontsize', 20, 'interpreter', 'latex')
388 xlabel('$x$', 'fontsize', 15, 'interpreter', 'latex')
389 ylabel('$t$', 'fontsize', 15, 'interpreter', 'latex')
390 %
391

```

```

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('$0(\Delta x^2)$', '$Error$');
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 %% Temporal 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 domain 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

```

```

436 %initialize error
437 err = zeros(size(dtvect));
438
439 tic
440 for j = 1:length(dtvect)
441
442     dt = dtvect(j);
443
444     %Build interp points
445     xj = ( a + (b-a)*(0:n)./n )';
446
447     %grid spacing (uniform)
448     dx = (b-a)/n;
449
450     %Build A matrix
451     %Use truncated version from lecture notes
452     A_temp = (c^2/(dx^2))*(-2*diag( ones(n-1,1),0) + 1*diag( ones(n-2,1),-1) ...
453         + ...
454         1*diag( ones(n-2,1),1) );
455
456     % Matrix of Zeros of size A_temp
457     Zeros = zeros(size(A_temp));
458
459     % The expanded A Matrix for 2nd degree differential equation
460     A = [Zeros eye(size(A_temp)); A_temp Zeros];
461
462     % Identity matrix of size A
463     I = eye(size(A));
464
465     %Build g for this set of xj
466     g1 = @(t) [zeros(size( xj(2:end-1) )); fcn(xj(2:end-1),t)];
467     g2 = @(t) [zeros(size(xj(2:end-1))); c^2/dx^2*bc_a(t); ...
468         zeros(size(xj(3:end-2))); c^2/dx^2*bc_b(t)];
469     g = @(t) g1(t) + g2(t);
470
471     %Build RHS for IVP, f(u,t)
472     f = @(u,t) A*u + g(t);
473     %---
474
475     %---Initialize for time stepping
476     uk = [u_init(xj(2:end-1)); v_init(xj(2:end-1))];
477     tk = 0;
478     tvect = dt : dt : T;
479
480     %# snapshots to save
481     nsnps = 250;
482     ind = max( 1, round(length(tvect)/nsnps) );

```

```

481     tsv = tvect( 1 : ind : end );
482
483     u = zeros( n-1, length(tsv));
484
485     cnt = 1;
486     %---
487
488     %---Do time stepping
489     for jj = 1:length(tvect)
490
491         tkp1 = tk + dt;
492
493         y1 = f(uk, tk);
494         y2 = f(uk + 0.5*dt*y1, tk);
495         y3 = f(uk + 0.5*dt*y2, tk);
496         y4 = f(uk + dt*y3, tk);
497
498         ukp1 = uk + (1/6)*dt*(y1 + 2*y2 + 2*y3 + y4);
499
500         uk = ukp1;
501         tk = tkp1;
502
503         if min(abs( tkp1-tsv )) < 1e-8
504
505             u(:,cnt) = uk(1:n-1);
506             cnt = cnt + 1;
507
508         end
509
510     end
511     %---
512     err(jj) = norm( ukp1(1:n-1) - uex(xj(2:end-1),T) ) / norm( ...
513         uex(xj(2:end-1),T) );
514
515 end
516 toc
517
518 %Error plots
519 figure(3)
520 c = err(end)*(1./dt^4);
521 loglog( dtvect, c*(dtvect).^4, 'k--', 'linewidth', 2 ), hold on
522
523 %plot err
524 loglog(dtvect, err , 'b.', 'markersize', 26 )
525 xlim([1e-2 100])
526 title('Temporal Convergence RK4')
527
528 %make pretty

```

```

527 h = legend('$0(\Delta t^2)$', '$Error$');
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 )

```