Waves and Transmission Lines

Project

Noam Navon - 322937384
Harel Shpunt - 207073132
Elhanan Kadosh - 316481134
Ron Hadad - 318553971
Niv Amgar - 208498956

Dr. Gregory Samelsohn
10 May 2024

Table of Contents

1. Background

In this project, we focused on wave propagation through a pseudo-infinite transmission line using the Finite Difference Time Domain method, described in detail in the supplemented file. Here, we clarify several of the methods we used during the project.

The term “infinite” transmission line is generally inaccurate when considering a simulation program since infinite data points are impractical to store and compute. Hence, to achieve our goal, we can make windows of transmission lines cascaded and remember two at each time; the output of the first is the input of the second, and since they are matched in impedance, there will be no reflections, so all the data is encompassed in a pair of windows. Incrementing coordinates to each new window will result in the illusion of an infinite transmission line. Two pairs can be used to handle both positive and negative waves.

A more concise solution is to shorten or isolate the ends of a transmission line, hence making total energy reflection at the edges. This approach is less accurate as an infinite transmission line descriptor but is easier to implement. Also, all the data points can be remembered and computed at once. Shorted contacts are identified by zero voltage, and isolated contacts have zero current between them. The reflection coefficient can be calculated as described in Equation 1.1

$$(1.1)\quad \Gamma_L = \frac{R_L - Z_0}{R_L + Z_0}$$

When shorted, the load impedance is zero, so the reflection coefficient is negative 1 for arbitrary characteristic impedance. This implies a full reflection of the voltage with an inverted sign.

Working on the project, we preferred specifying the capacitance and inductance of each lumped element on the transmission line instead of phase velocity and characteristic impedance.

$$(1.2)\quad v = \frac{1}{\sqrt{LC}} \qquad (1.3)\quad Z_0 = \sqrt{\frac{L}{C}}$$

Consider interest in the propagation values at integer lengths measured in meters so the space vector is fixed. Varying inductance and capacitance along the transmission line

will vary the phase velocity accordingly; we will obtain varying time differences for a fixed relation described in Equation 1.4.

$$(1.4) \quad \Delta t = \frac{\Delta z}{v}$$

When the wave goes through changes of the line in these spatial qualities, the velocity changes, and the time interval might be too big for the transition point, resulting in divergence of the voltage. For instance, if the time it takes for the wave to get from one side of the transmission line to the other is chosen as the differential time, the simulation will not calculate each step in the transmission line, so any obtained result will be nonsense. This is an exaggeration, but it exemplifies the importance of choosing relatively small time intervals for calculation.

A too-small interval will give good results but require many calculations for practical distance, while a too-big time interval is problematic, as discussed. The solution in Equation 1.5 is known as the Courant-Friedrichs-Lewy condition.

$$(1.5) \quad \Delta t \leq \frac{\Delta z}{v_{\max}}$$

For nonuniform increments of spatial coordinates, we obtain nonuniform time increments based on the location of interest. This is problematic to show on a graph as a snapshot in time, so we changed the expression further in Equation 1.6 to make the time difference constant.

$$(1.6) \quad \Delta t = \frac{\Delta z_{\min}}{v_{\max}}$$

When using the Leapfrog integration of the FDTD technique, we see half-integer increments in the staggered grid for both time and space values (see Figure 1.1). We need to emphasize that since MATLAB indexing is integer-valued, we calculate both as if they are measured in integer time and space steps, then treat the actual values in the plot.

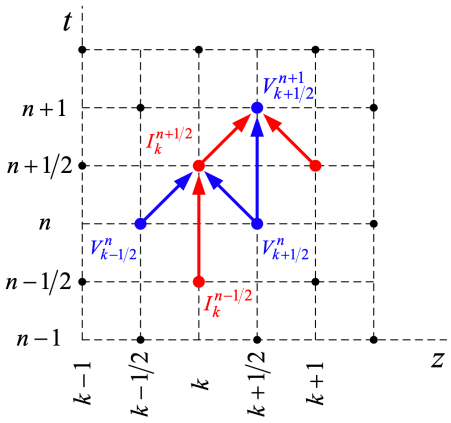

Figure 1

The half step must be addressed. For instance, the transmission line of the national company for producing and delivering electricity transmits electric waves at the speed of light and the frequency of fifty Hertz. Under these conditions, a lumped system is about 300 kilometers long, so half the spatial step is 150 kilometers.

This introduction is fundamental for the proper coding and simulation. From here, we proceed to explain our code step by step.

Note: Numbering of codes, figures, and equations, corresponds to chapter's number.

2. Transmission Line Class

A physical object such as TL is reasonable to simulate as an object; this way, we can specify physical attributes related to the line and implement methods that make it a functional object. The code is shown below:

```
classdef TransmissionLine
    properties
        Coordinate (1,:) double
        Inductance (1,:) double
        Capacitance (1,:) double
        Impedance (1,:) double
        Admitance (1,:) double
        PhaseVelocity (1,:) double
    end

    methods
        function obj = TransmissionLine(space,L,C)
            obj.Coordinate = space;
            obj.Inductance = L;
            obj.Capacitance = C;
            obj.Impedance = sqrt((obj.Inductance)./(obj.Capacitance));
            obj.Admitance = 1./obj.Impedance;
            obj.PhaseVelocity = 1./sqrt((obj.Inductance).*(obj.Capacitance));
        end

        function [time, resultV, resultI] = CalcWavesInfTL(obj, V, I, T)
            dz = [obj.Coordinate(1) diff(obj.Coordinate)];
            dt = min(dz)/max(obj.PhaseVelocity);
            time = dt*(0:T-1);

            resultV = [V; zeros(T-1, length(V))];
            resultI = [I; zeros(T-1, length(I))];

            for t=2:T
                I = I+obj.Admitance.*(((obj.PhaseVelocity).*dt)./dz).*([0 V(1:end-1)] - V);
                V = V+obj.Impedance.*(((obj.PhaseVelocity).*dt)./dz).*(I - [I(2:end) 0]);
                V(1)=0;
                V(end)=0;

                resultV(t,:) = V;
                resultI(t,:) = I;
            end
        end
    end
end
```

Code 2

All the physical properties can vary along the line corresponding to the measurement coordinate. The constructor method requires a space vector, inductance, and capacitance and realizes all other properties.

An important consideration—for an arbitrary distribution along the line, we must examine the spectral content. Based on the highest frequency (in constraints of importance), differential spatial steps for lumped elements are chosen. However, for our code's simplicity, we manually chose spatial increments.

The second method takes initial voltage, current distributions, and the number of time steps to compute. It returns a corresponding time vector and matrices for voltage and current where rows are the time of occurrence and columns are the place of occurrence. So, we can treat each row as a snapshot of the distribution and each column as the transient measured in a specific coordinate on the line.

The calculation is straightforward. Space increments are defined as changes in coordinates, and time increments are determined as explained in Equation 1.6. Time vector construction is trivial, and then a loop starts to compute the result matrices, iterating for each timing.

The core of the project is these pair of equations resulting from FDTD:

$$(2.1)\quad I_k^{n+1/2} = I_k^{n-1/2} + Y_k \frac{v_k \Delta t}{\Delta z}(V_{k-1/2}^n - V_{k+1/2}^n)$$

$$(2.2)\quad V_{k+1/2}^{n+1} = V_{k+1/2}^n + Z_k \frac{v_k \Delta t}{\Delta z}(I_k^{n+1/2} - I_{k+1}^{n+1/2})$$

For the equations to make sense in integer-based indexing, we made a slight variation that we fixed later. We added a half to the current time index and reduced a half from the voltage space index; thus, the modified equations are:

$$(2.3)\quad I_k^{n+1} = I_k^n + Y_k \frac{v_k \Delta t}{\Delta z}(V_{k-1}^n - V_k^n)$$

$$(2.4)\quad V_k^{n+1} = V_k^n + Z_k \frac{v_k \Delta t}{\Delta z}(I_k^{n+1} - I_{k+1}^{n+1})$$

Now, the code is a precise implementation of these equations. Incrementation in time is made as an assignment of a new value to the vector, and spatial coordinate variations are made by shifting the vector and padding with zero at the end. Notice how the second equation uses the time-incremented current, so the execution order can’t be exchanged. After executing these equations, the voltage edges are assigned with zeros as with shorted contacts, the vectors are saved to the corresponding row in the result matrices, and the loop continues until the required amount of time steps is achieved.

3. Animate Distribution Function

The result matrices of voltage and current are plotted row by row. If a real-time video is required, where the time of occurrence corresponds to the time of the video, a video with specified FPS can be written using the VideoWriter class. Our function takes the time interval that the TL generated, space interval, result matrices, and the variable ‘skips’ that determines how many frames are skipped between each plot. It helps reduce computations in case we want a faster simulation at the expense of a less smooth video.

```
function AnimateDist(time, space ,resultV, resultI, skips)

    for t=1:length(time)
        if mod(t, skips+1) == 0
            subplot(2,1,1)
            plot(space ,resultI(t,:),Color="r")
            txt = {'Time:' time(t)-time(2)/2};
            text('Units', 'normalized', 'Position', [0.01, 0.98], ...
                 'String', txt, 'VerticalAlignment', 'top', ...
                 'HorizontalAlignment', 'left');
            ylim([-1.5 1.5])
            ylabel("I[A]")
            xlim([0 space(end)])
            xlabel("z(m)")
            drawnow;

            subplot(2,1,2)
            plot(space+space(1)/2 ,resultV(t,:),Color="b")
            txt = {'Time:' time(t)};
            text('Units', 'normalized', 'Position', [0.01, 0.98], ...
                 'String', txt, 'VerticalAlignment', 'top', ...
                 'HorizontalAlignment', 'left');
            ylim([-1.5 1.5])
            ylabel("V[V]")
            xlim([0 space(end)])
            xlabel("z(m)")
            drawnow;
        end
    end
end
```

Code 3

For each time, we now plot the distribution along the line. Notice how we subtract half the time difference from the current time and add half the space difference to the voltage plot. This compensation is applied for the matrices that were generated for the modded Equations 2.3 and 2.4, so now we have correct results as described in Equations 2.1/2.2

The limits of y-coordinates are just for clarity and are specified for our distributions.

4. Plot Transient Function

This function takes the same variables as the distribution animation function, but the last variable is the location of the measurement probe. This variable is a vector of integers bounded by the length of the space vector, specifying the coordinates where we want to measure the result transient.

```
function PlotTransient(time, space ,resultV, resultI, probeLoc)

    plotAmount = length(probeLoc);

    for plotNum = 1:plotAmount
        subplot(2,plotAmount,plotNum)
        plot(time-time(2)/2 ,resultI(:,probeLoc(plotNum)),Color="r")
        txt = {'Space:' space(probeLoc(plotNum))};
        text('Units', 'normalized', 'Position', [0.01, 0.98], ...
             'String', txt, 'VerticalAlignment', 'top', ...
             'HorizontalAlignment', 'left');
        ylim([-0.5 0.5])
        ylabel("I[A]")
        xlim([0 time(end)])
        xlabel("t(s)")
        drawnow;

        subplot(2,plotAmount,plotAmount + plotNum)
        plot(time ,resultV(:,probeLoc(plotNum)),Color="b")
        txt = {'Space:' space(probeLoc(plotNum)) + space(1)/2};
        text('Units', 'normalized', 'Position', [0.01, 0.98], ...
             'String', txt, 'VerticalAlignment', 'top', ...
             'HorizontalAlignment', 'left');
        ylim([-0.5 0.5])
        ylabel("V[V]")
        xlim([0 time(end)])
        xlabel("t(s)")
        drawnow;
    end
end
```

Code 4

Once again, there is the fix of adding and subtracting half steps of space and time.

5. First Task

With the class and function prepared, we can now get to the code of the tasks:

```
function QuestA
    L = ones(1,1000);
    C = ones(1,1000);
    space = 1:1000;
    T = 1000;

    TL = TransmissionLine(space,L,C)

    s = 20;
    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = exp(-(((1:1000)-500).^2)./(2*s^2));
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    AnimateDist(time, space, resultV, resultI, 5)

    pause(1)

    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = -exp(-(((1:1000)-501).^2)./(2*s^2));
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    AnimateDist(time, space, resultV, resultI, 5)

    pause(1)

    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = zeros(1,1000);
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    AnimateDist(time, space, resultV, resultI, 5)

    pause(1)

    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = 0.5*exp(-(((1:1000)-500).^2)./(2*s^2));
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    AnimateDist(time, space, resultV, resultI, 0)
end
```

Code 5

The initial conditions were given as positive and negative waves, which we converted to voltage and current by simple calculations. Notice that in the second initial distribution, the current bell center is around 501 rather than 500. This is because the current is calculated before the voltage, and the voltage has an offset of half a spatial step.

Figure 5.1 captures the results of a rightward wave. Notice how the wave is reflected at the end, with the voltage sign inverted.

The time in seconds when the screenshot is taken is noted at the corner of the graph.

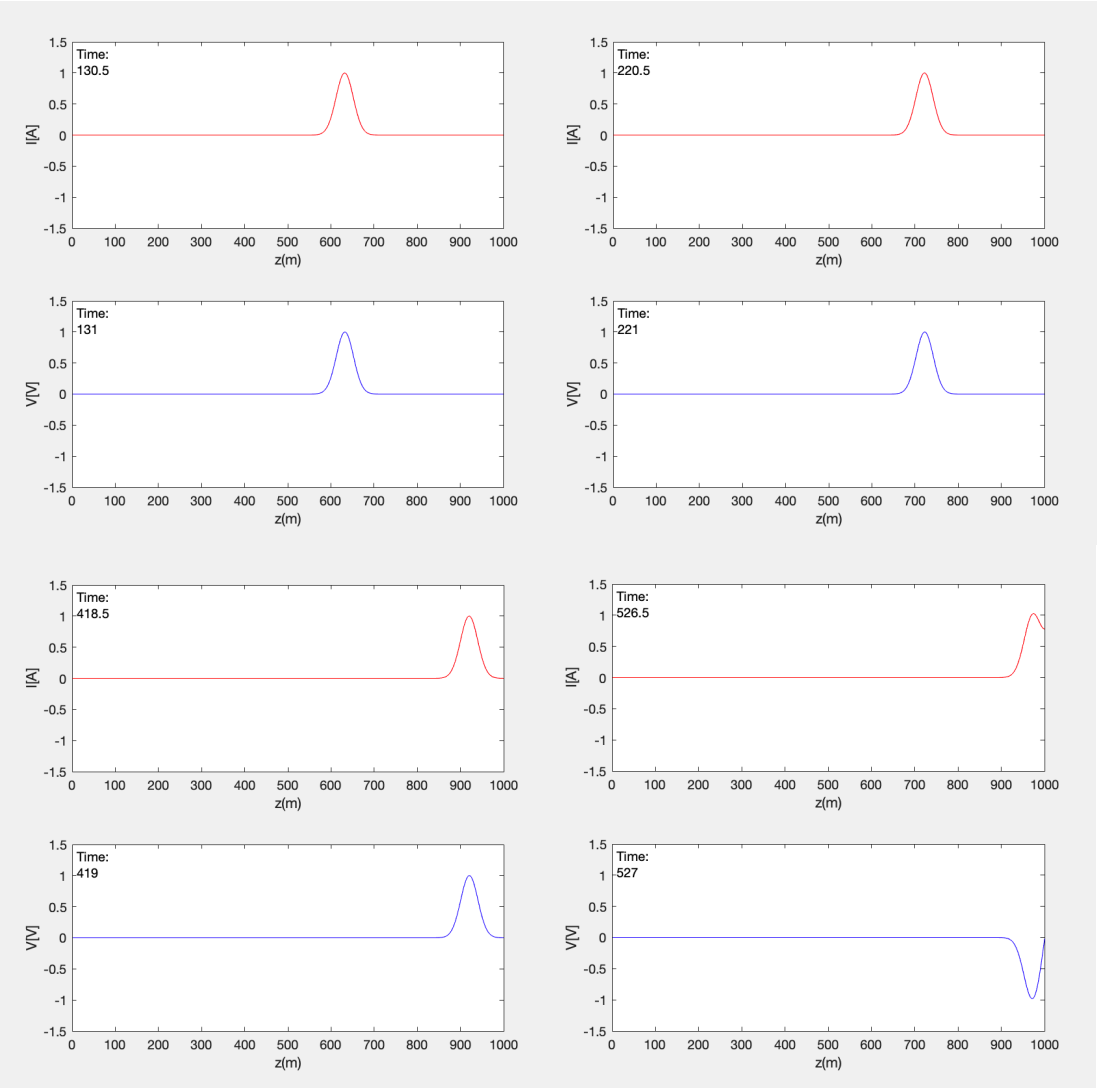

Figure 5.1

Figure 5.2 shows a leftward wave traveling in the TL. Once again, the voltage wave is inverted at the end of the line. The current has the same polarity as the rightward voltage wave and inverted polarity compared to the leftward voltage wave. The characteristic impedance also scales it.

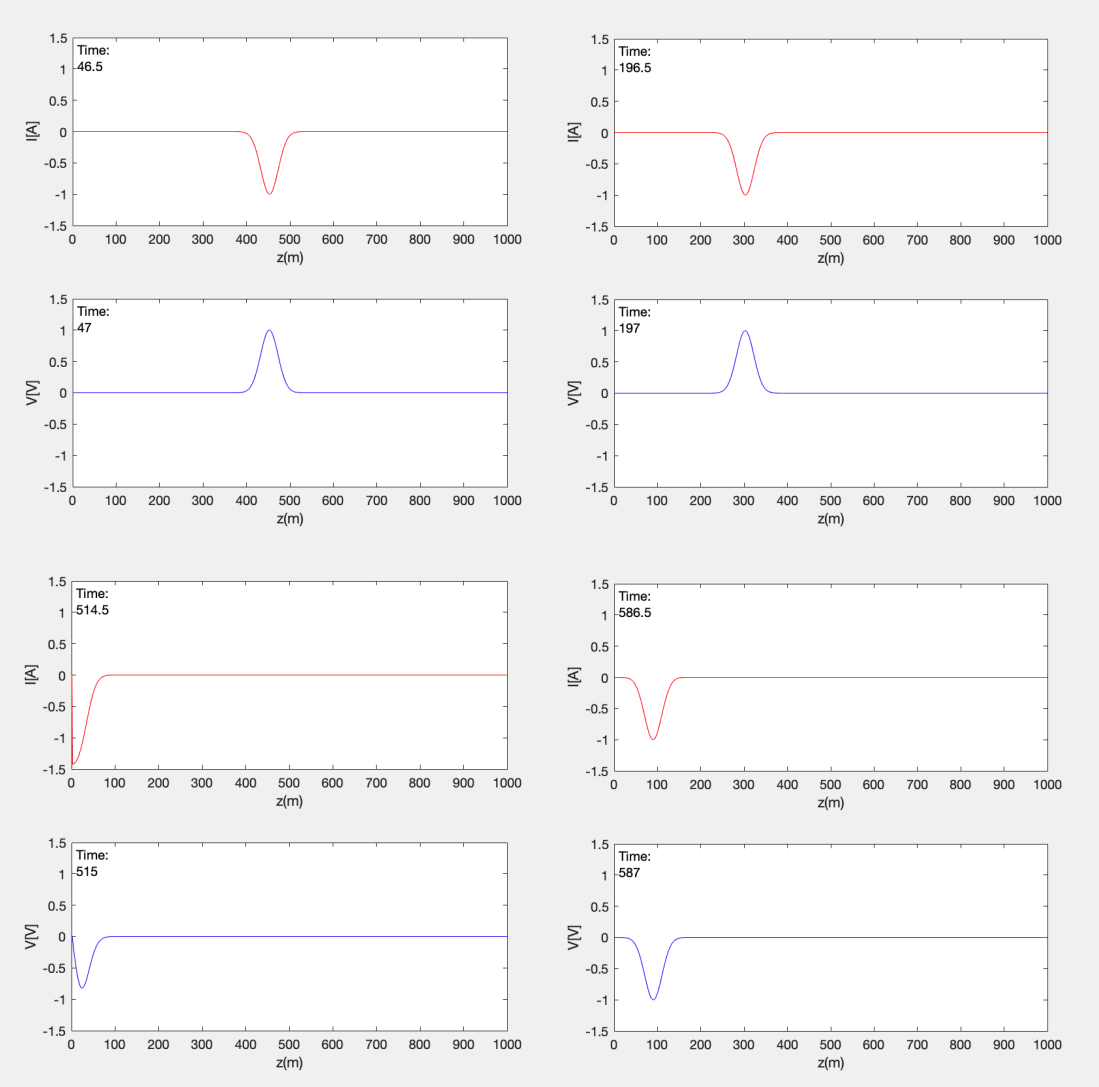

Figure 5.2

Figure 5.3 shows leftward and rightward waves with equal amplitudes splitting from the center. Notice that the current initial condition happens at a negative time step since it precedes the voltage initial condition, which is calibrated to time zero.

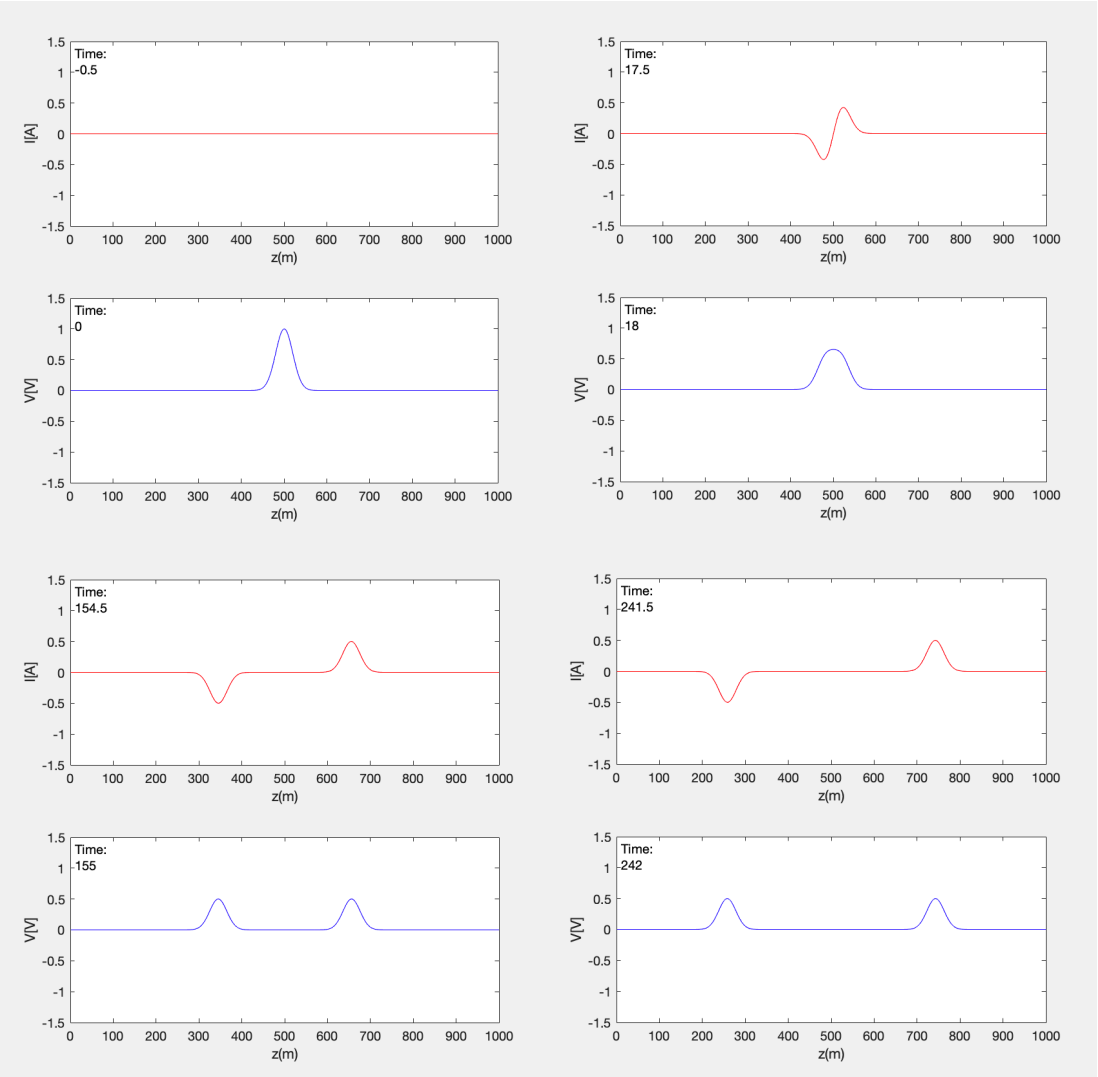

Figure 5.3

Finally, Figure 5.4 shows 25% amplitude leftward and 75% amplitude rightward waves. The graphs show Initial conditions, then after splitting, at the ends, and finally as they get reflected.

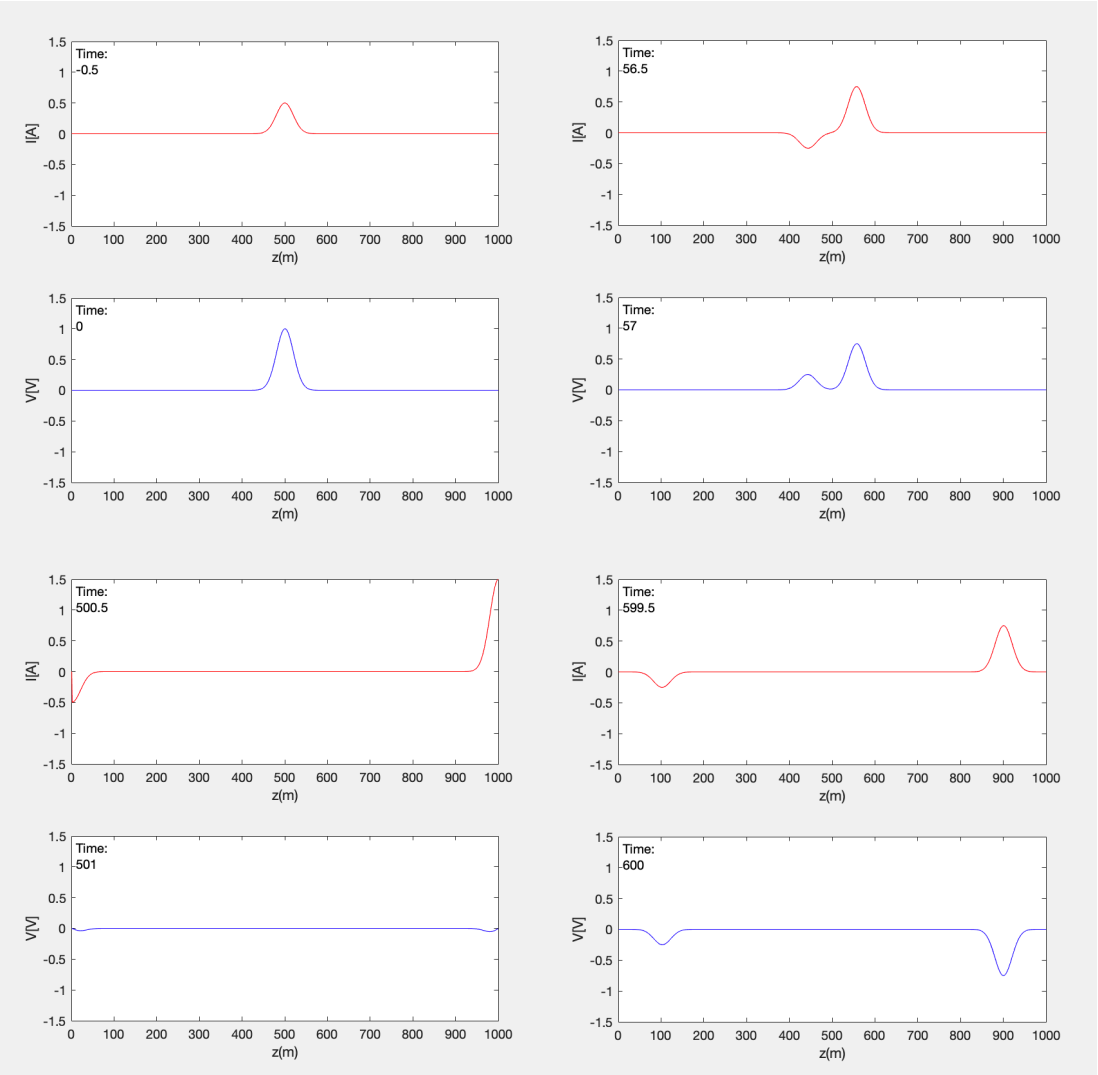

Figure 5.4

6. Second Task

The code is similar to the previous task code, but this time, we used the transient plot function at coordinates 250, 500, and 750 meters.

```
function QuestB
    L = ones(1,1000);
    C = ones(1,1000);
    space = 1:1000;
    T = 1000;

    TL = TransmissionLine(space,L,C)

    s = 20;
    V = 10*exp(-(((1:1000)-500).^2)./(2*s^2));
    V = [diff(V) 0];
    I = V;
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    PlotTransient(time, space, resultV, resultI, [250 500 750])

    pause(1)

    V = 10*exp(-(((1:1000)-500).^2)./(2*s^2));
    V = [diff(V) 0];
    I = -10*exp(-(((1:1000)-501).^2)./(2*s^2));
    I = [diff(I) 0];
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    PlotTransient(time, space, resultV, resultI, [250 500 750])

   pause(1)

    V = 10*exp(-(((1:1000)-500).^2)./(2*s^2));
    V = [diff(V) 0];
    I = zeros(1,1000);
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    PlotTransient(time, space, resultV, resultI, [250 500 750])

    pause(1)

    V = 10*exp(-(((1:1000)-500).^2)./(2*s^2));
    V = [diff(V) 0];
    I = 5*exp(-(((1:1000)-500).^2)./(2*s^2));
    I = [diff(I) 0];
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    PlotTransient(time, space, resultV, resultI, [250 500 750])
end
```

Code 6

For a rightward wave in the shape of the derivative of the Gaussian bell, we obtain Figure 6.1. Notice how, in coordinate ~250, there is nothing all along the simulation. This is because the wave goes from 500 meters, and the simulation needs more time to reach the 250-meter coordinate after reflection. In the simulation length we fixed, the wave returns precisely to where it was launched from. This is evident from the 500-meter graph, where we see wavelets at the start and the end of the simulation, from when the wave is launched to when it returns. The last pair of graphs is yet another example.

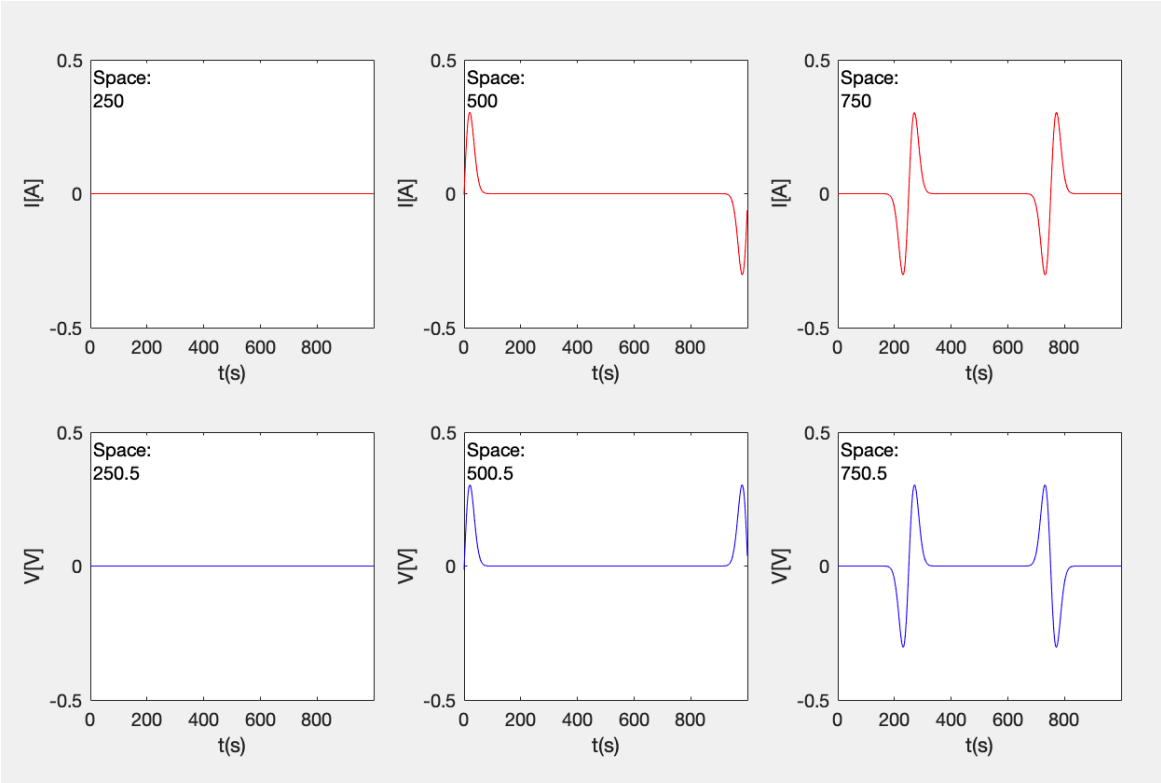

Figure 6.1

For the leftward wave, we obtain Figure 6.2. Now, the silence is at the right.

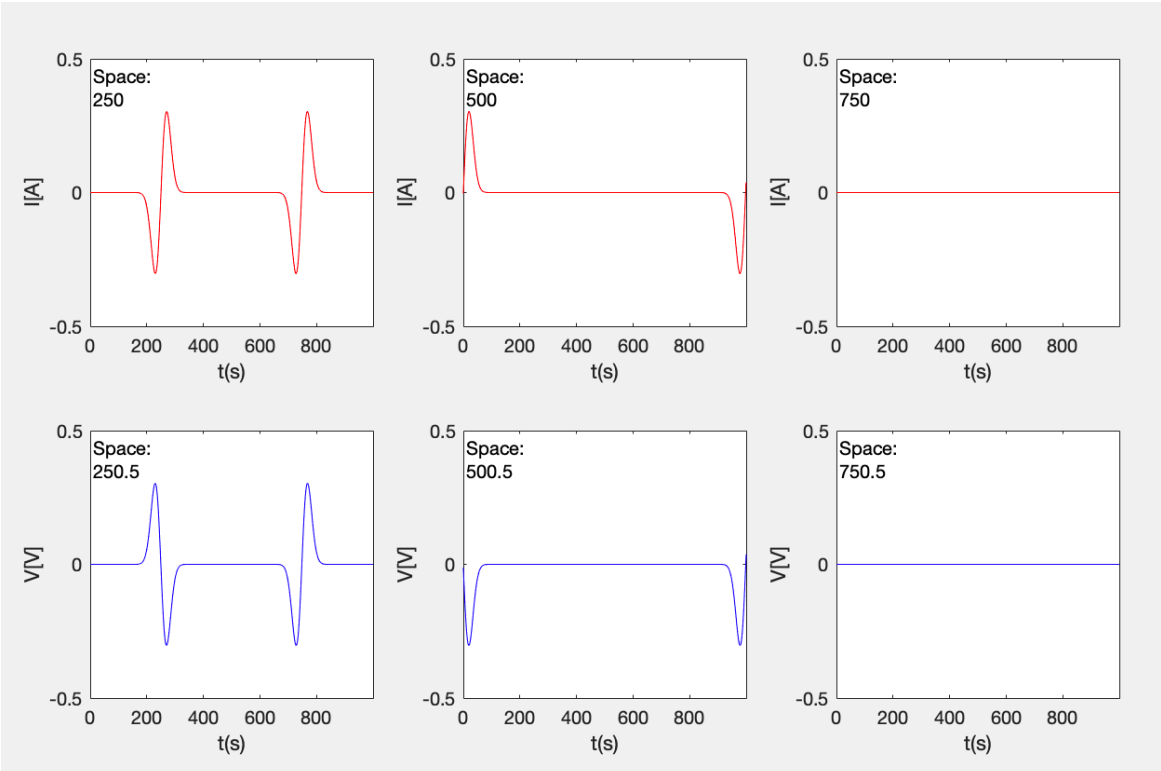

Figure 6.2

Figure 6.3 shows the results for equal left and right voltage waves shaped as a derivative of the Gaussian bell. The left and right are now similar.

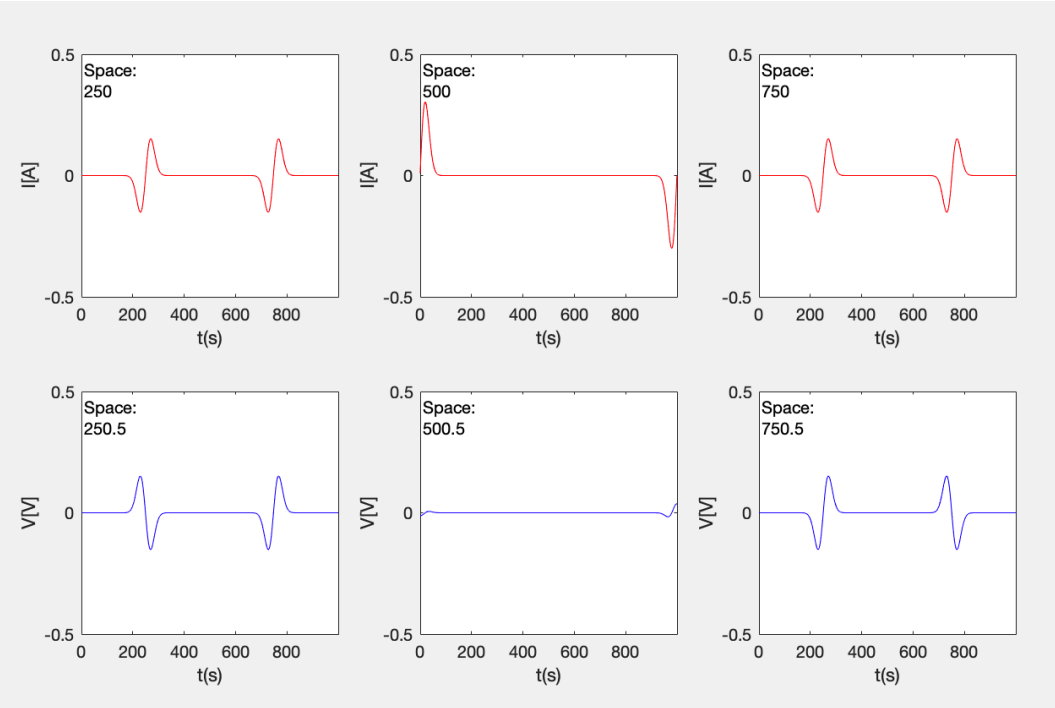

Figure 6.3

Figure 6.4 shows the results for 25% amplitude left and 75% amplitude right voltage waves. The right side transient is now stronger, as expected.

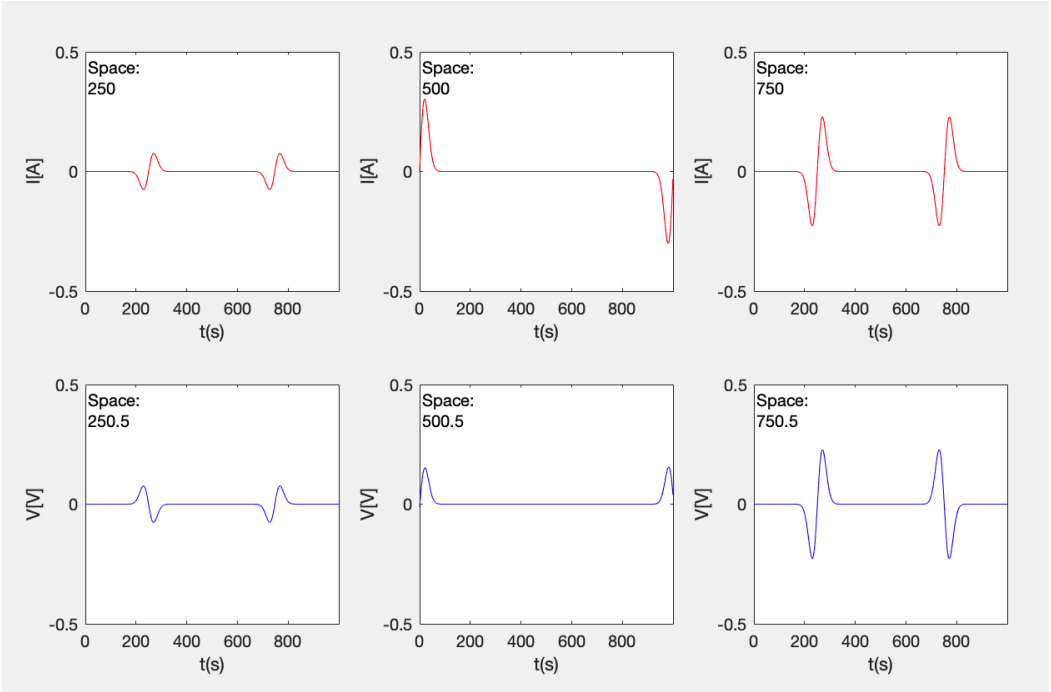

Figure 6.4

7. Third Task

When the Courant-Friedrichs-Lewy condition is satisfied (see Equation 1.5), the time difference is small enough, and the simulation is fine. This can be implemented simply by reducing the time difference a little where it is defined (the CalcWavesInfTL method). Choosing a too-small interval will result in a slow, computationally heavy simulation.

Code 7.1 : `dt = min(dz)/max(TL.PhaseVelocity)-0.1;`

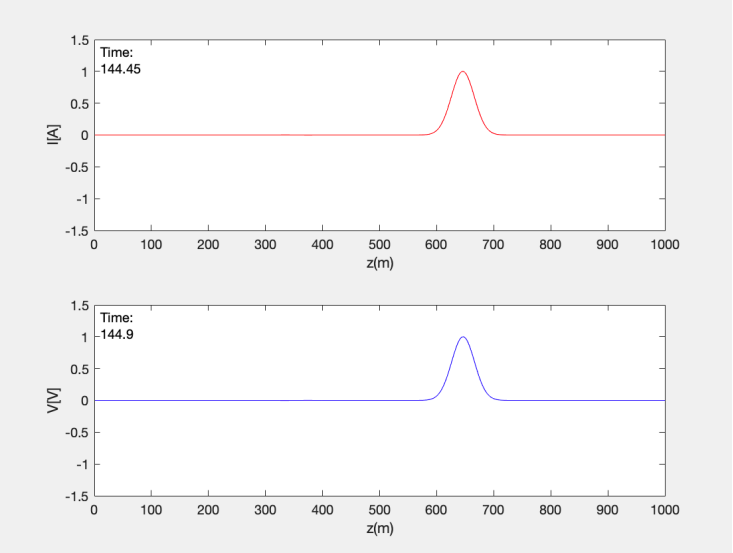

Figure 7.1

When the condition is not satisfied, the simulation won't work as expected since we don't examine the phenomena in reasonable time intervals. Figure 7.2 shows how the result escalates. This code modification shows the effect for homogeneous line.

Code 7.2 : `dt = min(dz)/max(TL.PhaseVelocity)+0.1;`

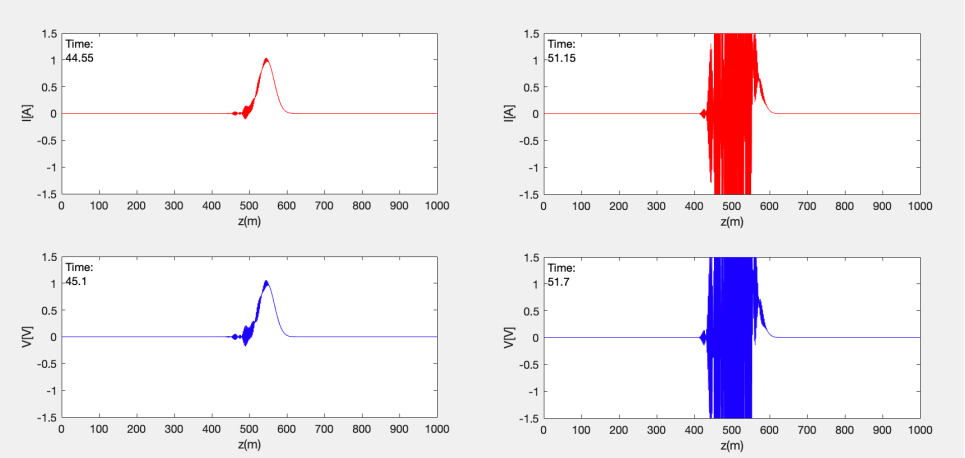

Figure 7.2

For more experimentation, see the QuestC.m file, should that be required.

8. Fourth Task

As the waves reach the end of the line, they are reflected back, where the voltage sign is inverted because we simulated a shorted-end TL. Figure 8.1 is the same example as the first task, the third part. We focused more on screenshots where the waves are close to the ends of the line.

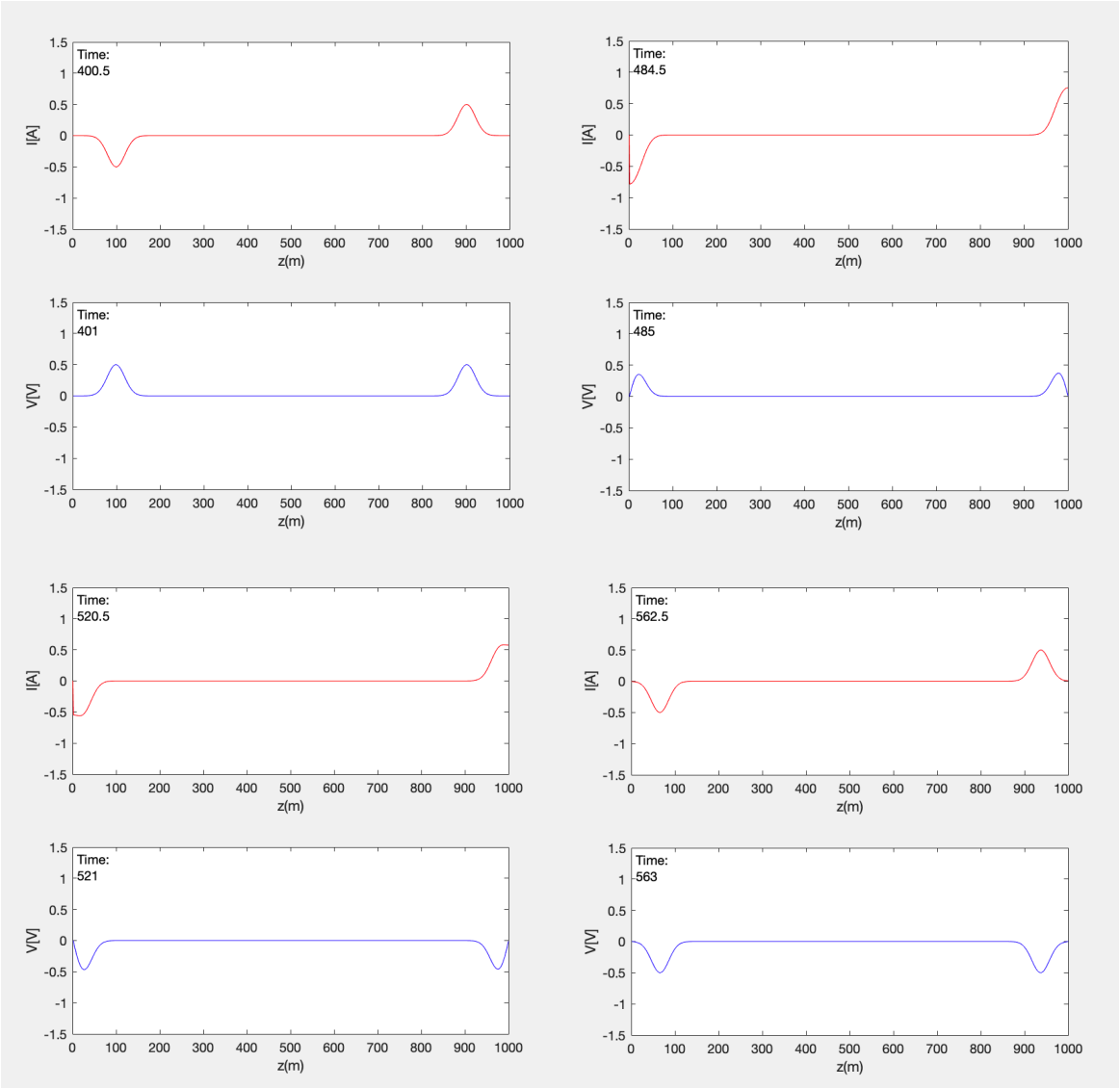

Figure 8.1

9. Fifth Task

The ground is set for cascading different transmission lines; it is easily done by varying the capacitance and inductance:

```
function QuestE
    clear all
    clc

    L = [ones(1,750) 4*ones(1,250)];
    C = ones(1,1000);
    space = 1:1000;
    T = 1000;

    TL = TransmissionLine(space,L,C)

    s = 20;
    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = V;
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);
    AnimateDist(time, space, resultV, resultI, 5)
end
```

Code 9.1

At first, it seems like the wave ran into an obstacle at 750 meters. Then, we gradually start to see the division between reflected and transmitted waves, see Figure 9.1.

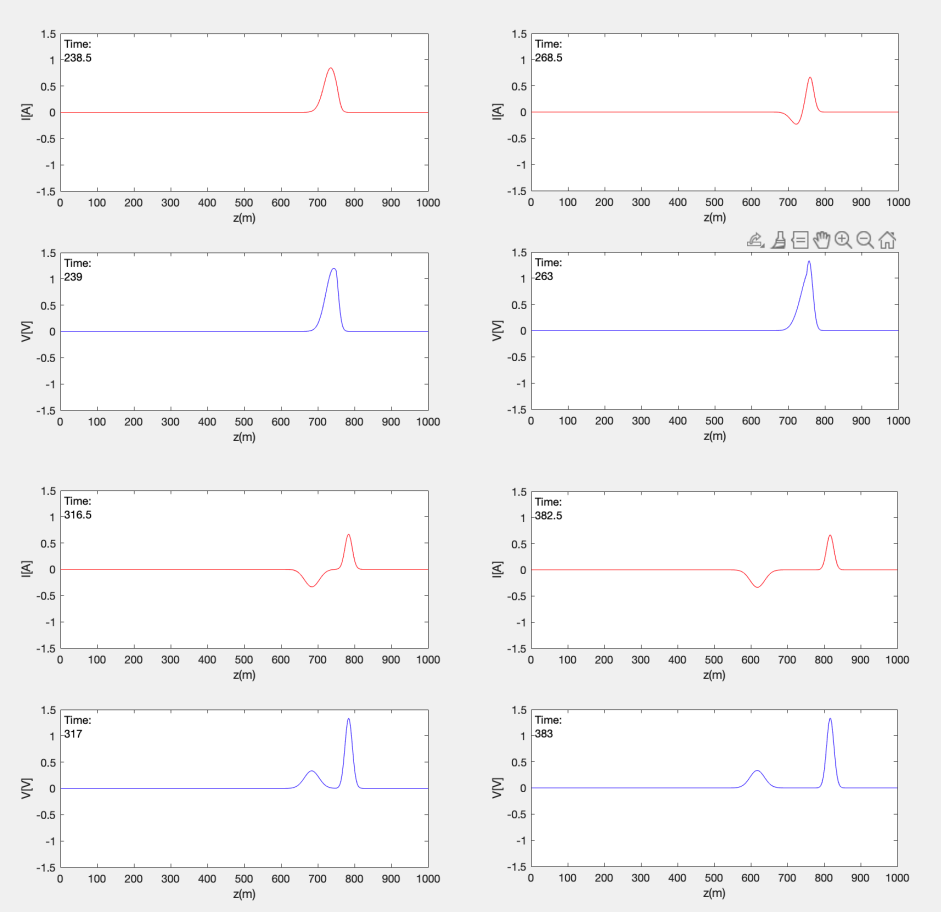

Figure 9.1

The strength of our approach is that we can easily cascade any number of transmission lines and simulate defective transmission lines by adding randomness to the capacitance and inductance. For example, in Code 9.2, we are cascading five lines with different inductance per lumped chain, and each line has a different length.

```
L = [pi*ones(1,100) ones(1,300) 2*ones(1,200) 4*ones(1,150) 3*ones(1,250)];}
C = ones(1,1000) + 0.1*rand(1,1000);
```

Code 9.2

Figure 9.2 shows the result in one screenshot. Since the impedances are similar, the main wave is noticeable at the right, but the reflections are still prominent. The overall simulation is much more vibrant when different left and right waves move through each other.

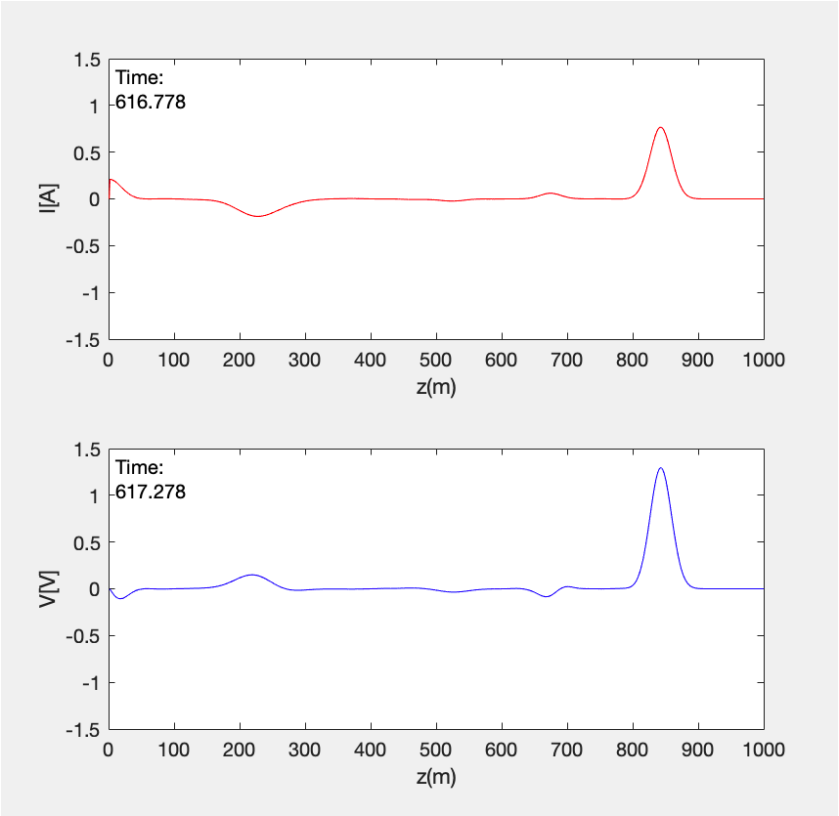

Figure 9.2

10. Bonus

Since we've come this far, it would be a shame not to use our matrices for more advanced visuals. Here is a code for drawing voltage and current as a function of both space and time. Notice that the half-step correction is absent here, so the results should be a little bit shifted to compensate.

```
function Bonus
    clear all
    clc

    L = [3*ones(1,250) 10*ones(1,250) ones(1,250) 6*ones(1,250)];
    C = ones(1,1000);
    space = (1:1000);
    T = 1000;

    TL = TransmissionLine(space,L,C)

    s = 20;
    V = exp(-(((1:1000)-500).^2)./(2*s^2));
    I = V;
    [time, resultV, resultI] = TL.CalcWavesInfTL(V, I, T);


    [Time, Space] = meshgrid(time, space);
    subplot(1,2,1)
    surf(Space, Time, resultV', EdgeColor="none");
    xlabel('Space (m)');
    ylabel('Time (s)');
    zlabel('Voltage (V)');
    title('Voltage Distribution on a Transmission Line');
    colorbar

    subplot(1,2,2)
    surf(Space, Time, resultI', EdgeColor="none");
    xlabel('Space (m)');
    ylabel('Time (s)');
    zlabel('Current (A)');
    title('Current Distribution on a Transmission Line');
    colorbar
end
```

Code 10

The impression these Figure 10.1 make can not be argued. At one glance, we get insights about the propagation of waves through space and time. Through rotation and zooming in, it is possible to acquire better measures in a location of interest.

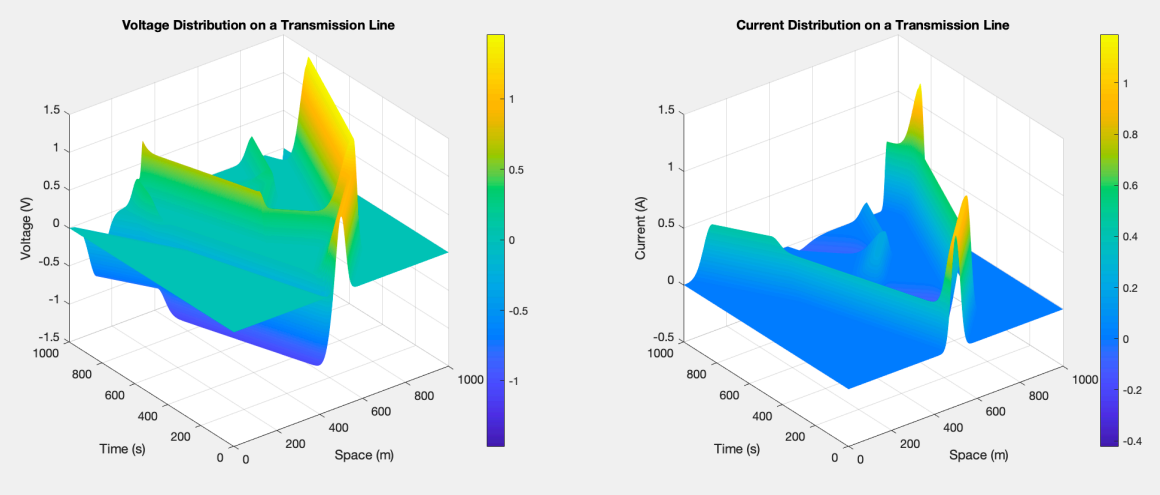

Figure 10.1

By rotating the graph to an angle where the axis of the value is parallel to the angle of sight, we obtain Figure 10.2. This is a waveform diagram for any arbitrary wave shape! (Echoes diagram.) Here, where the wave is a simple bell, the diagram is simple. For complex data transmission plus noise, we should expect something much messier.

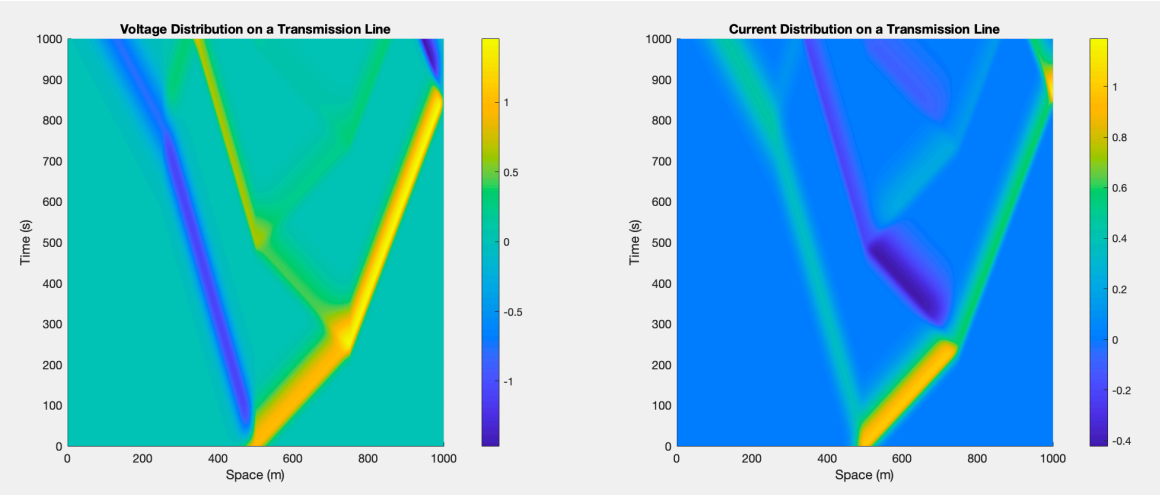

Figure 10.2

All the parameters can be varied in the Bonus.m file.