Project 1: FDTD Modeling of the 1D Tx-Line Equations

Vicente Garcia

University of Colorado Denver

Dr. Stephen Gedney

ELEC 4333

February 27, 2025

Table of Contents

Abstract

The Finite Difference Time Domain method is a technique widely used in computational electromagnetics that provides second order accurate solutions to Maxwell's equations. This accuracy results useful for real-life applications involving scattering, wave propagation, radar, communication systems, among others. The objective of this project is to develop software that performs an FDTD solution for a finite length one-dimensional transmission line. Cases have been established where different parameters are set such as lossless/ lossy lines, reactive/ resistive loads, and short/open circuit loads. It is expected the software be capable of simulating the transmission line voltages and currents with respect to time and space, based on boundary conditions input by the user. A trapezoidal pulse is defined as Thevenin voltage source. The programming language chosen for software is MATLAB

Keywords: FDTD, Thevenin source, sources and loads, trapezoidal pulse, maxwell equations, MATLAB.

Theory

Electromagnetic waves can be represented with Maxwell's equations. These equations describe the wave behavior in free space or a determined medium. The Finite Difference Time Domain method requires Central Difference approximations, and Taylor' series to achieve high order accuracy solutions. The central difference approximation is based on the subtraction of forward and backward difference operators. At this project, a 1D transmission line will be analyzed during propagation using the FDTD method.

When applying this method to maxwell equations, subscripts such as *i* and *n* are defined for space and time, as seen in (*1*) and (2). *G* and *R* account for the conductive and resistive losses present in the line, while *C* and *L* account for the per unit length capacitance and inductance.

$$\frac{I_{i+\frac{1}{2}}^{n} - I_{i-\frac{1}{2}}^{n}}{\Delta x} \approx -G\frac{V_i^{n+\frac{1}{2}} + V_i^{n-\frac{1}{2}}}{2} - C\frac{V_i^{n+\frac{1}{2}} - V_i^{n-\frac{1}{2}}}{\Delta t} \tag{1}$$

$$\frac{V_{i+1}^{n+\frac{1}{2}} - V_i^{n+\frac{1}{2}}}{\Delta t} \approx -G\frac{R_{i+\frac{1}{2}}^{n+1} + R_{i+\frac{1}{2}}^{n}}{2} - L\frac{I_{i+\frac{1}{2}}^{n+1} - I_{i+\frac{1}{2}}^{n}}{\Delta x} \tag{2}$$

Both equations are consistent if the discrete samples of *V* and *I* are separated in space and time by half cell, $\Delta x/2$, and $\Delta t/2$. It is important to note that, Ampere's Law (1) is centered at point *(i,n)*, and Faraday's Law (2) is centered at point *(i+1/2, n+1/2)*. Linear interpolation is applied at V_i^n and $R_{i+\frac{1}{2}}^{n+\frac{1}{2}}$. The representations of *V* and *I* are staggered in space and time leading to the "leap frogging" technique. Considering $V_i^{n-\frac{1}{2}}$ and $I_{i+\frac{1}{2}}^{n}$ as known values, a recursive update relation can

be derived to predict the in-line values of $V_i^{n+\frac{1}{2}}$ and $I_{i+\frac{1}{2}}^{n+1}$, as seen in (3) and (4). Consequently, the wave equation will advance in time and space. Added to that, the equations must hold the stability criteria. This is true if the space and time steps satisfy the *Courant Stability Limit*, $\frac{c*\Delta t}{\Delta x} \leq CFLN$, where *c* is the wave velocity, $c = \frac{1}{\sqrt{L*C}}$. If the inequality is not true, then the energy of the system will grow unboundedly. The line will have a distance *d* of 0.5m, with space discretization Δx of 0.01m, and a characteristic impedance $Z_0 = \sqrt{\frac{L}{C}}$. The same analysis will be necessary for boundary conditions.

$$V_i^{n+\frac{1}{2}} = \left[\frac{\frac{C}{\Delta t} - \frac{G}{2}}{\frac{C}{\Delta t} + \frac{G}{2}}\right] V_i^{n-\frac{1}{2}} - \left[\frac{1}{\frac{C}{\Delta t} + \frac{G}{2}}\right] \frac{I_{i+\frac{1}{2}}^{n} - I_{i-\frac{1}{2}}^{n}}{\Delta x} \tag{3}$$

$$I_{i+\frac{1}{2}}^{n+1} = \left[\frac{\frac{L}{\Delta t} - \frac{R}{2}}{\frac{L}{\Delta t} + \frac{R}{2}}\right] I_{i+\frac{1}{2}}^{n} - \left[\frac{1}{\frac{L}{\Delta t} + \frac{R}{2}}\right] \frac{I_{i+1}^{n+\frac{1}{2}} - I_{i}^{n+\frac{1}{2}}}{\Delta x} \tag{4}$$

Boundary conditions such as sources and loads must be considered. A Thevenin source with resistance *Rs* is applied at the source node or node 1, with a trapezoidal pulse as input signal. The pulse have an amplitude of 2V, a rise and fall time of 200ns, and a duration time of 500ns. To derive the recursive update relation at the source node, a KCL analysis is performed. The presence of *Rs* slightly modifies the recursive update equation as evidenced in (5). This is useful for determining the initial conditions of source voltages for the wave to travel. When considering lossless transmission lines, the conductance G and resistance R simply go to 0 in the equations.

$$V_1^{n+\frac{1}{2}} = \left[\frac{Rs\left(\frac{C\Delta x}{\Delta t} - \frac{1}{2} - \frac{G\Delta x}{4}\right)}{Rs\left(\frac{C\Delta x}{\Delta t} + \frac{1}{2} + \frac{G\Delta x}{4}\right)}\right] V_1^{n-\frac{1}{2}} + \left[\frac{1}{Rs\left(\frac{C\Delta x}{\Delta t} + \frac{1}{2} + \frac{G\Delta x}{4}\right)}\right] (Vs - Rs * I_1^n) \quad (5)$$

Under the same principle, a recursive update relation can be derived for load voltages if load parameters are known. At load node *N*, different parameters may represent lossless/lossy lines, matched/mismatched loads, and resistive/reactive loads such as series R-L, and parallel R-C. The easiest load configurations for analysis are short circuits and open circuits. That is because for a short circuit, its load voltage is simply 0, meanwhile for an open circuit the load current is 0.

Thus, $V_N^{n+\frac{1}{2}} = 0$ and $I_{N-\frac{1}{2}}^n = 0$.

Since the current load is sampled at node *N-1/2*, it is not possible to know the current at node *N*. This is due to the half-cell discretization of the line. Using image theory, the recursive load voltage update for the open circuit load is derived as indicated in (6). When having resistive loads, the approach is the same independently of mismatches. It can be seen in (7) that conductance losses are accounted for. In this way, the FDTD method will update the voltage and currents over time, based on the source/load conditions. It is important to remark that the recursive update equations are derived using KCL, linear interpolations and central difference approximations in the time and space derivatives.

$$V_N^{n+\frac{1}{2}} = V_N^{n-\frac{1}{2}} + 2\frac{\Delta t}{C * \Delta x} I_{N-\frac{1}{2}}^{n} \tag{6}$$

$$V_N^{n+\frac{1}{2}} = \left[\frac{\frac{C\Delta x}{\Delta t} - \frac{1}{2Rl} - \frac{G\Delta x}{4Rl}}{\frac{C\Delta x}{\Delta t} + \frac{1}{2Rl} + \frac{G\Delta x}{4Rl}}\right] V_N^{n-\frac{1}{2}} + \left[\frac{1}{\frac{C\Delta x}{2\Delta t} + \frac{1}{2Rl} + \frac{G\Delta x}{4Rl}}\right] (Rl * I_{N-1}^{n}) \tag{7}$$

In terms of reactive loads, cases have been explored such as series R-L loads, and parallel R-C loads. A similar approach is done with both load models; however, a series R-L will introduce a matrix representation of the equations. This is done to provide ease at calculations. As evidenced in (*8*), the capacitance *Cl* is introduced at the recursive load voltage update for the parallel R-C load.

$$V_N^{n+\frac{1}{2}} = \left[\frac{\frac{C\Delta x + 2Cl}{2\Delta t} - \frac{1}{2Rl} - \frac{G\Delta x}{4Rl}}{\frac{C\Delta x + 2Cl}{2\Delta t} + \frac{1}{2Rl} + \frac{G\Delta x}{4Rl}}\right] V_N^{n-\frac{1}{2}} + \left[\frac{1}{\frac{C\Delta x + 2Cl}{2\Delta t} + \frac{1}{2Rl} + \frac{G\Delta x}{4Rl}}\right] I_{N-1}^{n} \tag{8}$$

Next, for a series R-L load, the inductor *Ll* needs to be accounted in the recursive update equation. Applying KCL and KVL at the load node and mesh, 2 equations, with 2 unknowns arise. Due to the complexity of the equations, these are represented as matrices. This can be viewed in (9). In reduced notation, these systems can also be expressed as $P * X^{n+\frac{1}{2}} = Q * X^{n-\frac{1}{2}} + Y^{n}$, where P and Q are 2x2 matrixes. Therefore, at each time step, it can be concluded that $X^{n+1/2}$ is the same as (10). Lastly, after careful analysis it is demonstrated that different configurations for loads and sources

will lead to different recursive update equations. As mentioned previously, this is necessary for advancing the line equations across the transmission line with its discrete spaces and times.

$$\begin{bmatrix} \left(\frac{C\Delta x}{2\Delta t}+\frac{G\Delta x}{4}\right) & \frac{1}{2} \\ \frac{1}{2} & -\left(\frac{L}{\Delta t}+\frac{Rl}{2}\right) \end{bmatrix} \begin{pmatrix} V_N^{n+\frac{1}{2}} \\ I_l^{n+\frac{1}{2}} \end{pmatrix}$$

$$= \begin{bmatrix} \left(\frac{C\Delta x}{2\Delta t}+\frac{G\Delta x}{4}\right) & -\frac{1}{2} \\ -\frac{1}{2} & -\left(\frac{L}{\Delta t}-\frac{Rl}{2}\right) \end{bmatrix} \begin{pmatrix} V_N^{n-\frac{1}{2}} \\ I_l^{n+\frac{1}{2}} \end{pmatrix} + \begin{pmatrix} I_{N-1}^{n} \\ 0 \end{pmatrix} \tag{9}$$

$$P^{-1}(Q * X^{n-\frac{1}{2}} + Y^n) = \begin{pmatrix} V_N^{n+\frac{1}{2}} \\ I_l^{n+\frac{1}{2}} \end{pmatrix} \tag{10}$$

Code Description

A top-level code design has been developed to compute the recursive update equations for specific input cases. The code follows the following chart design: input program control, data array initialization, main script, and post process. The main script contains the line, load and source voltage updates. These blocks have been defined as scripts instead of functions due to simplicity.

First, the input program control includes 7 input cases that relate to each case study provided. These are selected by the user under the "*inputcase"* script. As evidenced in Figure 1, a few line parameters are defined independently of a specific case, meaning these stay fixed at all cases. These variables are: *dx, d, ncells, C, L, vp*, and Z_0, where *dx* stands for discretization in space, *d* stands for the line distance, *C* and *L* for the per unit length capacitances and inductances, *vp* for the phase velocity, and Z_0 for the characteristic impedance of the line.

```
%In line script for input variables according to case study chosen
%1    Lossless with matched Rl, Rs
%2   Lossless with open circuit
%0    Lossless with short circuit
%3    Lossless with missmatched Rl and Rs
%4    Lossless with clfn=0.99 and missmatched Rl, Rs
%5    Lossy with missmatched Rl and Rs
%6    Lossless with missmatched Rs, and parallel RC load
%7    Lossless with missmatched Rs, and series RL load

dx=0.01; d=0.5;
ncells=d/dx;
C= 100e-12; L=250e-9;
vp=1/sqrt(L*C);
zo= sqrt(L/C);

% Input Variables
case_study=input("Choose a case number between 0-7 \nCase study= ");
```

Figure 1: Input Case script

Then, the user is prompted to choose a specific case study from 0 to 7. Each case will input variables such as *L, C, R, G* values, as well as simulation time, load type, time snaps, and the

Courant Friedrisch constant. As seen in Figure 2, once the user has specified a case study, the program defines all the input variables pertinent to that specific case study with the use of *if* loops and *elseif* statements. It is valuable to remark that Figure 2 only shows two of the seven cases, for the sake of spacing.

```
if case_study==1

   clfn=1;
   simtime=5e-9;
   dt=clfn*dx/vp;
   nx=ncells+1;
   nt=floor(simtime/dt);
   G=0; R=0;
   Rs=50; Rl=50;
   loadtype=1;
   timesnaps=[0,1.25e-9,2.5e-9,3e-9,5e-9];
   numsnaps=5;

elseif case_study==0

    clfn=1;
    simtime=5e-9;
    C= 100e-12; L=250e-9;
    nx=ncells+1;
    nt=floor(simtime/dt);
    G=0; R=0;
    simtime=5e-9;
    Rs=50; Rl=1e-30;  % short circuit intepreted as very small
    loadtype= 0;
    timesnaps=[2.5e-9, 3e-9, 5e-9];
    numsnaps=3;
```

Figure 2: Input variables for case 1 and 0.

Moving forward with the data array initialization block, a matrix of zeros is defined for current and voltage as seen in Figure 3. Both are 51x1, and 50x1 matrices respectively. These are useful for storing all values of voltage and current along *nx* and *nx-1*. Clearly, the 1st position refers to the source load, while *nx* and *nx-1* refers to the load node for all configurations. Also, a data array *x* is defined for the whole distance of the line using the *linspace* command; that is a sequence that goes from 0 to *d* in 51 step points. Additionally, the same approach is considered for storing voltages at each specific time stamp required. A *ksnap=1* variable is defined to represent the index

of the first snapshot, while *k1* represents the Voltage (V) vs Distance (m) plot. This will update in real time with the voltage values represented on their recursive equations.

```
%Initialization of data arrays
V= zeros(nx,1); I=zeros(nx-1,1);
x=linspace(0,d,nx);

k1=plot(x,V,'Linewidth', 2);
grid on
ylim([-2 3])
ylabel('Voltage (V)')
xlabel('x(m)')
title('Voltage vs Distance')

%Data initialization for snapshots
nsnap = floor(timesnaps/dt)+1;
Vsnap=zeros(nx,numsnaps);
ksnap=1;
```

Figure 3: Initialization of data arrays

```
for n = 1:nt

      vupdate; % update in line voltages
      vsupdate; % update source voltages
      vlupdate; % update load voltages
      iupdate; % update the line currents

       set(k1,'Ydata',V);
       drawnow;
       pause(0.05);

      % output the probe voltages and currents:
      output;

end
```

Figure 4: Main script

Furthermore, the main script runs the core of the whole program. At this section all load, in-line and source updates are computed as noted in Figure 4. A *for n=1:nt* loop is used for defining all the time steps over the simulation. It's important to note that (*3*), (*4*), (*5*), (*7*), (9), and (*10*) need to be slightly modified for readability with MATLAB syntax. Figure 5 shows the array indexing of the voltage update equations as *(2:nx-1).* This means the loop runs from position 2, to *nx-1*. The factor coefficients are defined as *cv2* and *cv1*. Added to that, the same line of action is taken for the in-line current update equations as noted in Figure 6, however, the array index goes from (*1:nx-1).*

```
% In line script for in line voltage updates

%Coefficients represented as cv1 and cv2
cv1 = (C/dt-G/2)/(C/dt+G/2);
cv2 = 1/(C/dt+G/2)/dx;

% Update equation
 V(2:nx-1)= cv1*V(2:nx-1)-cv2*(I(2:nx-1)-I(1:nx-2))
```

Figure 5: In-line voltage update script

```
% In line script for in line current updates

%Coefficients represented as ci1 and ci2
ci1 = (L/dt-0.5*R)/(L/dt+R*0.5);
ci2 = 1.0/(L/dt+0.5*R)/dx;

%Update equation
I(1:nx-1) = ci1*I(1:nx-1)-ci2*(V(2:nx)-V(1:nx-1));
```

Figure 6: In-line current update script

Moreover, at the source node, a trapezoidal pulse is set as Thevenin source. As mentioned previously, its amplitude is of 2V, time rise and fall time of 200ns, and duration time of 500ns. Using *if* loops with *elseif* statements, the trapezoidal pulse specifications are established as viewed in Figure 7. Namely, the source voltage equations are updated using *cvs1* and *cvs2* as factor coefficients. This can be noticed in Figure 8.

```
Vs=0;
% Source time
t=(n-0.5)*dt;

if t <= 0
        Vs = 0;
    elseif t< trise
        Vs = Vamp*t/trise;
    elseif t< trise+tduration
        Vs = Vamp;
elseif t< trise+tfall+tduration
        Vs = Vamp*(trise+tfall+tduration-t)/tfall
    else
    Vs= 0;

end
```

Figure 7: Source voltage update script

```
%Vs coefficients
cvs2 = (1.0/(C*dx*0.5/dt+0.5/Rs+0.25*G*dx));
cvs1 = (C*dx*0.5/dt-0.5/Rs-0.25*G*dx)*cvs2;
cvs2= cvs2/Rs;

%Vs update
V(1)=cvs1*V(1)+cvs2*(Vs-Rs*I(1));
```

Figure 8: Source voltage update script ctd

Likewise, the voltage equations need to be updated at the load node. This is done respectively for each load type and case study. When defining variables for the input cases, a different load type variable is defined for resistive loads, open/short circuit loads, and reactive load combinations such as parallel R-C, and series R-L. Factor coefficients are defined as *cvl1*, and *cvl2*. Using *if* loops and *elseif* statements the recursive equations for voltage are updated as noted in Figure 9. It is valuable to remark that a load type of -1 represents an open circuit, 0 represents a short circuit, 1 a pure resistive load, 2 a parallel R-C load, and 3 a series R-L load.

```
if loadtype==-1
    cvl1= (C/dt-G/2)/(C/dt+G/2);
    cvl2= 1/(C/dt+G/2)/dx;
    V(nx)= cvl1*V(nx)+cvl2*I(nx-1);

elseif loadtype==0
    V(nx)=0;

elseif loadtype==1
    cvl1=(C*0.5*dx/dt-0.5/Rl-0.25*G*dx);
    cvl2=1/(C*0.5*dx/dt+0.5/Rl+0.25*G);
    V(nx)= cvl2*(cvl1*V(nx)+I(nx-1));

elseif loadtype==2
    cvl1= (C*0.5*dx/dt+Cl/dt-0.5/Rl);
    cvl2= 1/(C*0.5*dx/dt+Cl/dt+0.5/Rl);
    V(nx)= cvl2*(cvl1*V(nx)+I(nx-1));

elseif loadtype==3

    P = [(C*0.5*dx/dt), (0.5); (0.5), (-(Ll/dt+Rl*0.5))];
    Q = [(C*0.5*dx/dt),(-0.5); (-0.5),(-(Ll/dt-Rl*0.5))];
    pinv = inv(P);

    X= [V(nx);IL];
    Y =[I(nx-1);0];
    X=pinv*(Q*X+Y);
    V(nx)=X(1);
    IL=X(2);

end
```

Figure 9: Load voltage update script

```
% Output script for line voltage at discrete times


    if(ksnap <= numsnaps && n == nsnap(ksnap))

      Vsnap(:,ksnap) = V;

       ksnap = ksnap+1;  % update to next snap shot index

    end
```

Figure 10: Output script validation for voltages at time stamps

Lastly, as part of the post processing block, the update equations are computed for all steps in the line. This data is updated in graph *k1*. As demonstrated with the animated plot, all voltage values are updated with each iteration inside the loop. Additionally, to update the voltage values

for all time stamps, an *if* loop is defined where the *ksnap* index will grab the voltage value representing that index position; this is seen in Figure 10. On the other hand, all *Vsnap* values are stored and plotted once the simulation time is done. This can be evidenced in Figure 11.

```
% plot the voltage snapshots:
for ksnap = 1:numsnaps

    if(ksnap == 1)

        figure();
   end

    % plot the graph of this time stamp in the figure
   plot(x,Vsnap(:,ksnap),'LineWidth',1);
   grid on
   title('Voltage vs Distance')

    % This stores the time of the snap shot by converting the number to
    % string, and concatinating with the unts of nanoseconds

    legendtext{ksnap} = strcat(num2str(timesnaps(ksnap)*1e9),'ns');

    hold on;

end

%Label the x and y axes.
xlabel('x (m)');
ylabel('Voltage (V)');

% Set the limits of the x bounds and y-bounds
axis([0 d -2.0 2.0]);

% This creates a legend using the cell array of strings
legend(legendtext,'Location','best');

% set the font size, so the plot is readable in a report
set(gca,'FontSize',10)
```

Figure 11: Code for voltage update values with time stamps

Case Studies Validation

As mentioned in *code description,* 7 different case studies have been assigned for analysis. Each of these cases have different parameters that will influence the wave behavior. Case 1 is a uniform lossless transmission line with matched load *Rl*, that is $Z_0 = Rs = Rl = 50\Omega$. Its capacitance is of 100pF/m, while its inductance is 250nH/m. Considering a lossless line, the conductance is 0 S/m. The magic time step provided uses *CFLN* of 1, assuming an ideal stable case. The simulation time is 5ns, and the time stamps are at 0, 1.25ns, 2.5ns, 3ns, and 5ns. As evidenced in Figure 12, the pulse travels along the line without attenuation or reflection. That is expected since loads and sources are matched. The load type variable for this case equals to 1, meaning a resistive load configuration.

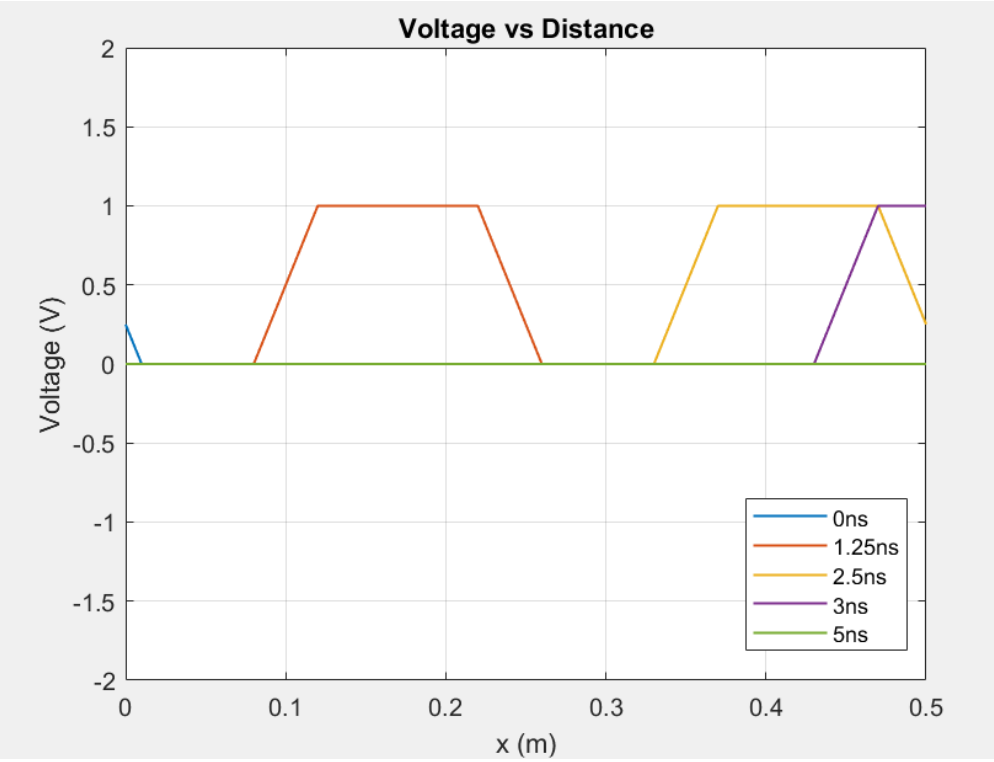

Figure 12: Case study 1 results

Case study 2 repeats case study 1 but with short and open circuit loads instead. As mentioned previously, shorts and open circuits loads are the easiest configuration for analysis. At this case, the time stamps are at 2.5ns, 3ns, and 5ns. An open circuit load is represented by case 2 while a short circuit load is represented by case 0. Figure 13 demonstrates how full reflection of the wave occurs. That is, because at the open circuit, no current will flow through the node, hence total reflection. Also, the load type variable for an open circuit is -1. On the contrary, in case 0, current doesn't encounter any resistance, hence, no voltage presents at the load. The load type variable for this case is 0. Also, a negative reflection is present due to mismatches with the load node as seen in Figure 14.

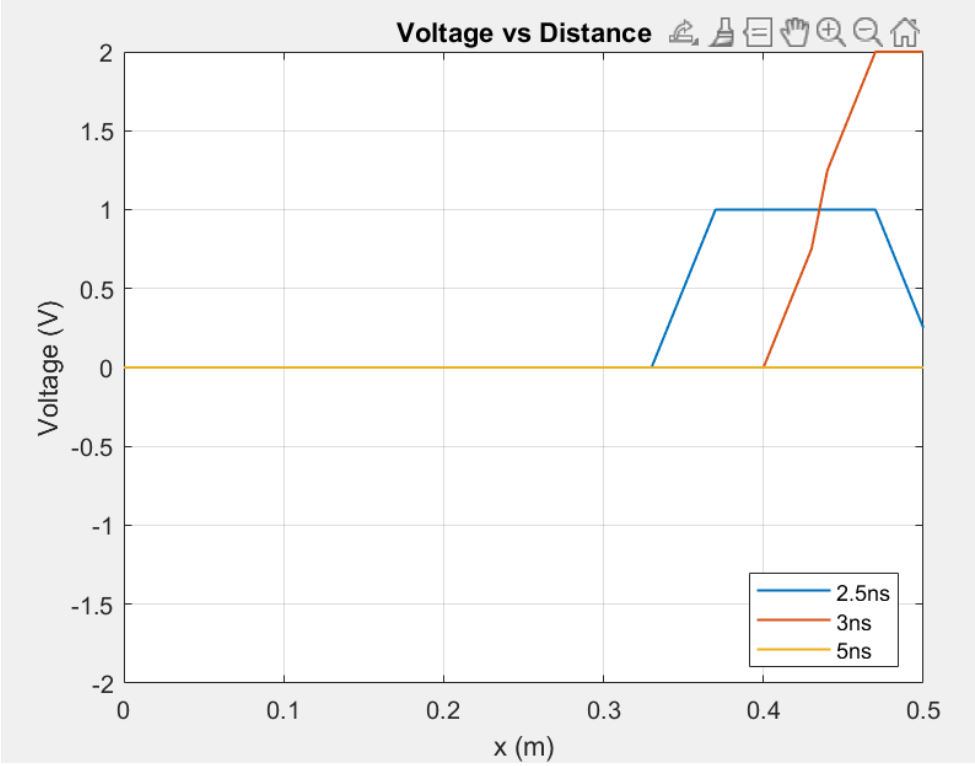

Figure 13: Case study 2 results

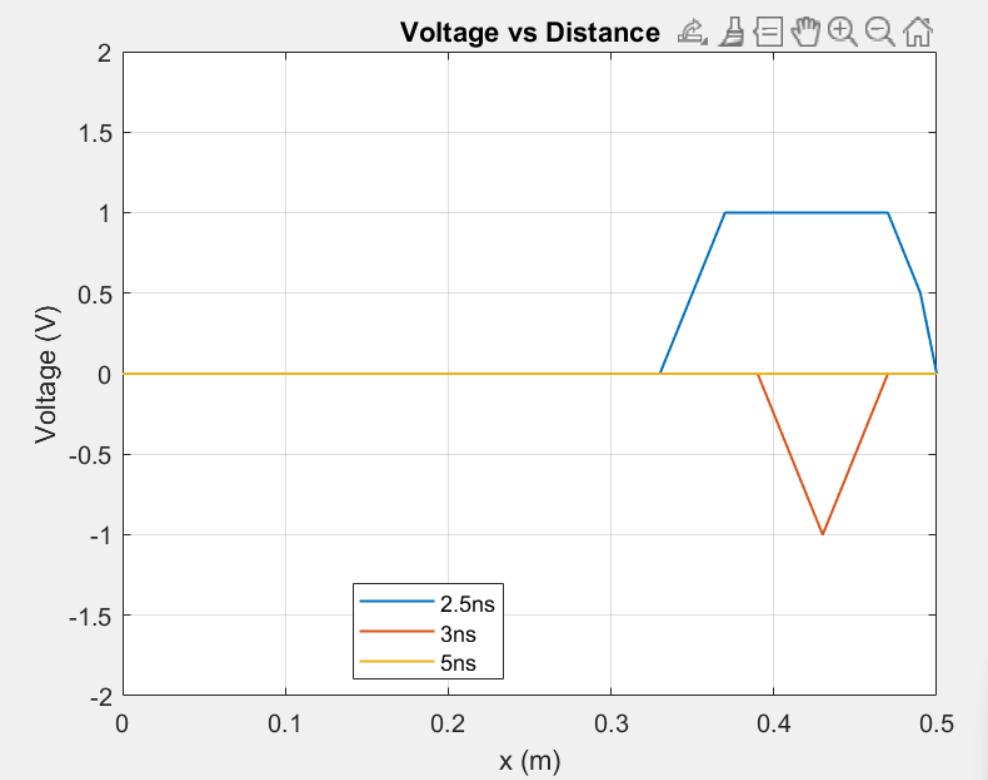

Figure 14: Case study 0 results

Case study 3 repeats case study 1 but uses resistive loads. Mismatches are present from the source and load since *Rs=25Ω*, and *Rl=150Ω*. The simulation time increases to 10ns. The load voltages are evaluated at the following time stamps: 2.5ns, 3ns, 5ns, 7.5ns, and 10ns. The load type variable for this case equals to 1, meaning a resistive load configuration. It is evidenced in Figure 15 that the pulse bounces off the source and loads continuously until losing all its power. Reflection occurs at both nodes, until the pulse is fully dissipated. Expected results are met.

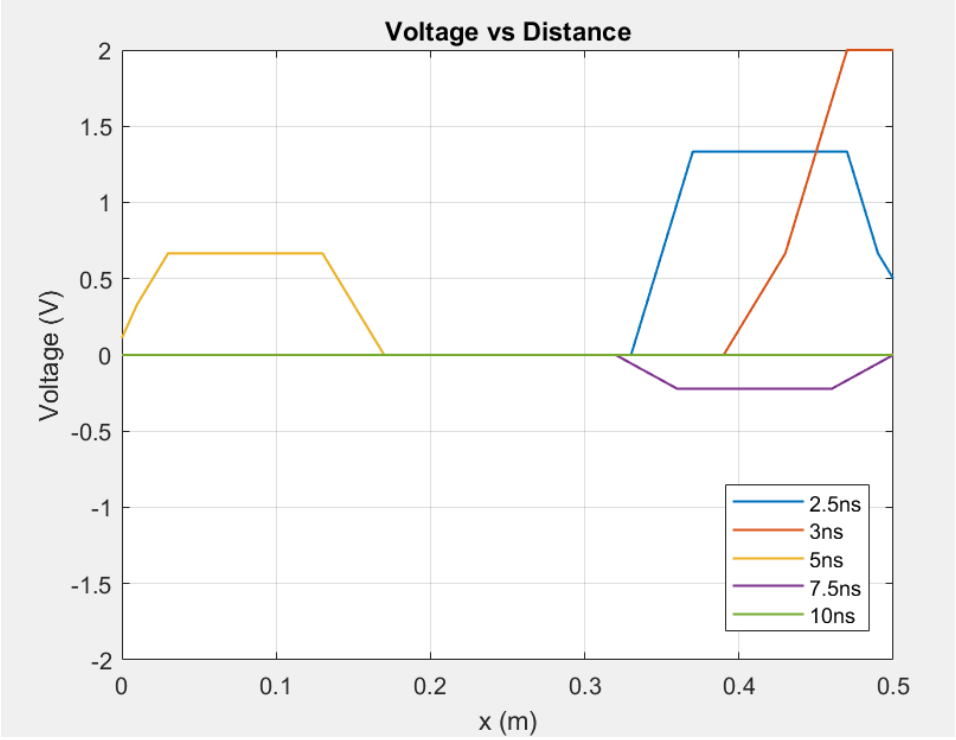

Figure 15: Case study 3 results

Case study 4 repeats case study 3 but includes an effective CLFN constant of 0.99. That is 99% of the magic time step. The load type variable for this case equals to 1, meaning a resistive load configuration. It can be noticed in Figure 16 that some ringing is present across most frequencies in the pulse. That is because by modifying the CFLN number, the discretization of space and time get affected as well. Consequently, error will be introduced, leading to "ladder" effect in the curves. Besides that, the wave bounces off the source and the load continuously until it loses all its power. The wave behaves as expected.

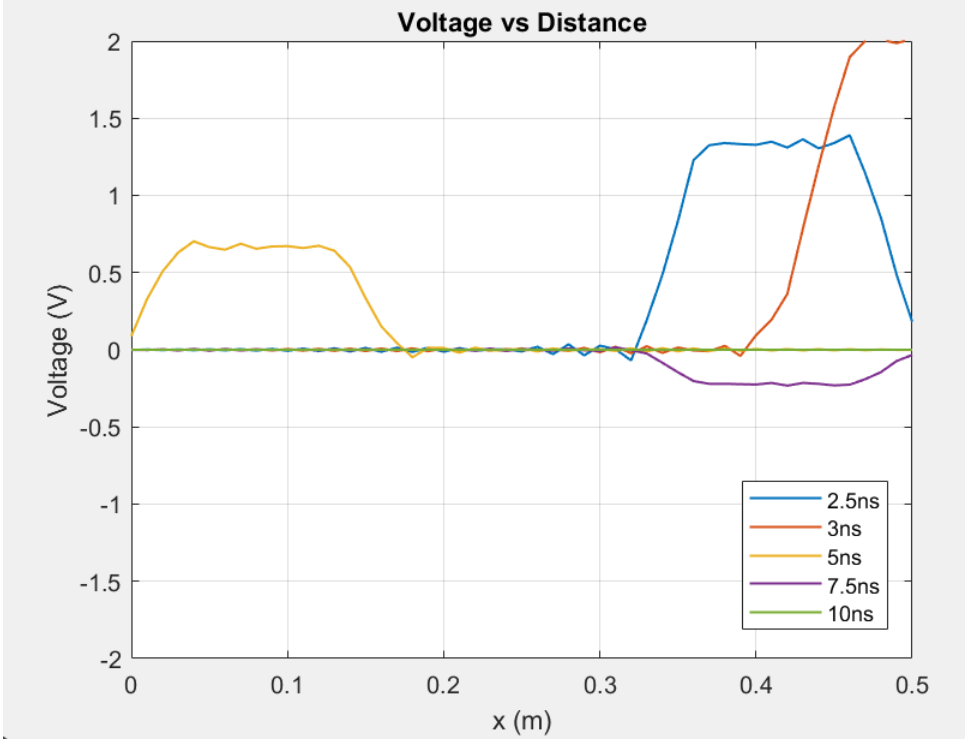

Figure 16: Case study 4 results.

Case study 5 repeats case study 3 but includes losses in the line. The per unit length conductances, and resistances, have values of 25mS/m and 50mΩ/s respectively. The load type variable for this case equals to 1, meaning a resistive load configuration. Losses in the line dissipate the pulse continuously while bouncing on and off with the source and the load. In this case, some power is lost during reflection and propagation because of the line parameters. Figure 17 demonstrates results for case study 5.

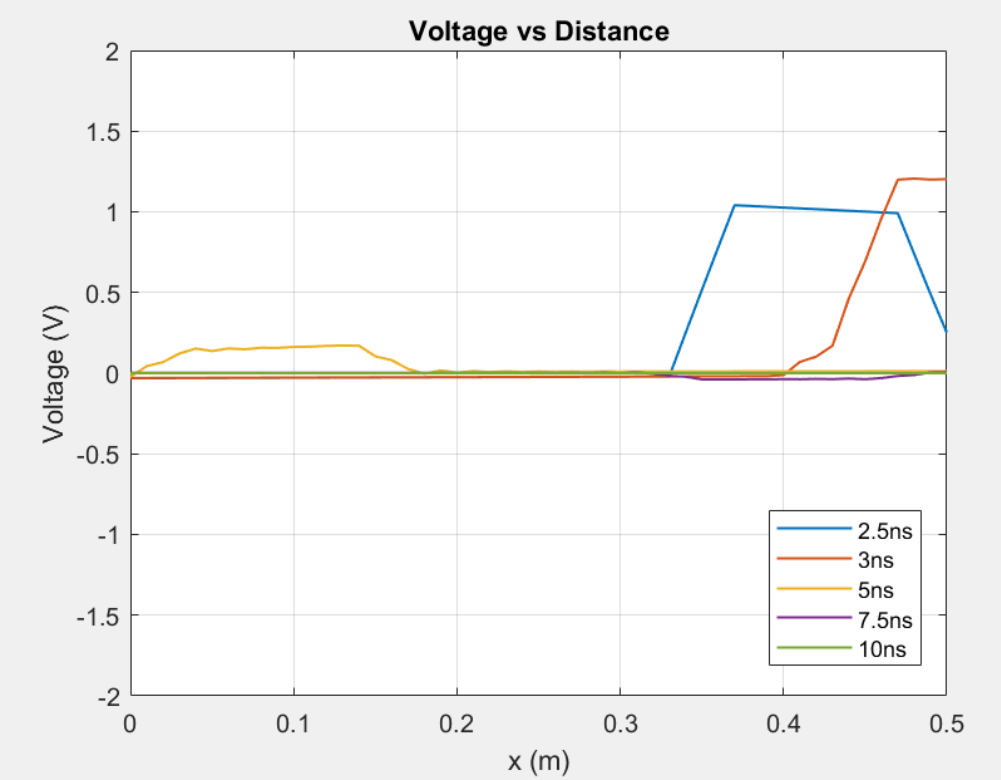

Figure 17: Case study 5 results

Case study 6 repeats case study 3 but uses a parallel R-C load configuration. At this case the load type variable is 2. A load capacitor is introduced of 5pF, while the load resistance increases to 150Ω. Figure 18 demonstrates clear evidence of reflections and distortion caused by the load. The R-C load causes initial reflection and then dissipates as the pulse travels. Due to the behavior of the capacitor it tends to act as a short circuit at very high frequencies, but after a while its voltage recovers. This causes a negative reflection as seen in Figure 18. It can be compared that as the capacitor charges, its impedance increases. This is because the capacitor is a frequency dependent element.

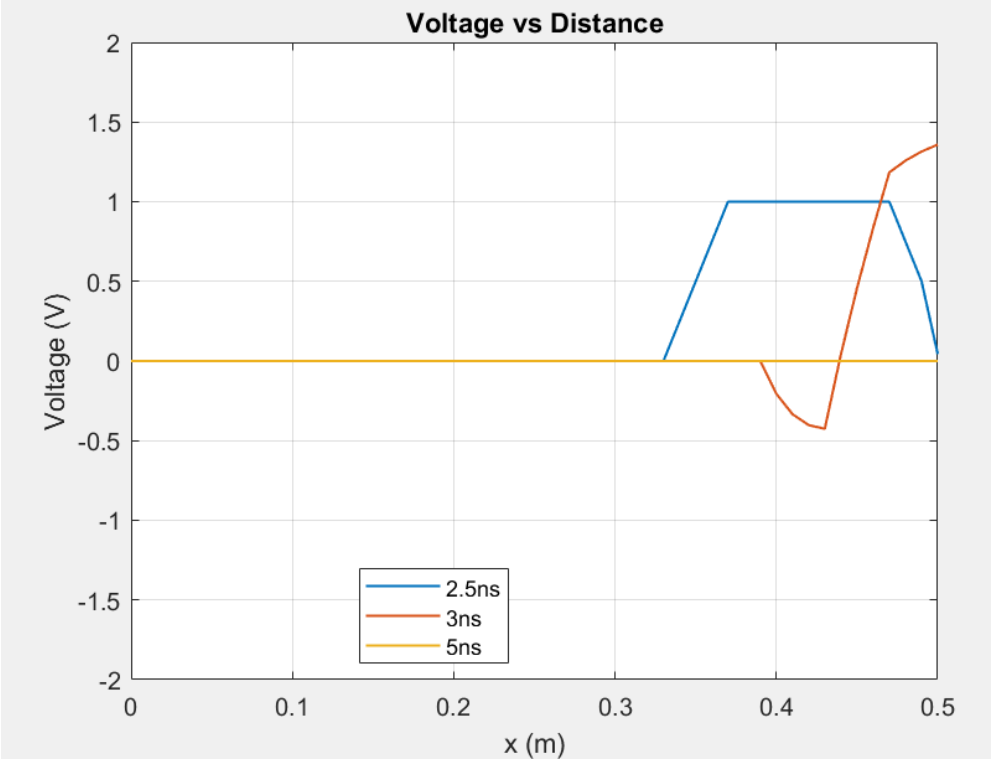

Figure 18: Case study 6 results

Lastly, case study 7 repeats case study 3, but uses a series R-L load configuration. The load type variable for this case is 3. A load inductor is introduced of 10nH, while the load resistance decreases to 10Ω. Figure 19 demonstrates clear evidence of reflections and distortion caused by the load. The R-L load causes initial reflection and then dissipates as the pulse travels. Due to the behavior of the inductor, it acts with higher impedances at high frequencies, but after a while its voltage settles down. This causes a positive reflection as seen in Figure 19. Quite the opposite as case study 6. All simulations have met expectations, resulting in successes in the experiment.

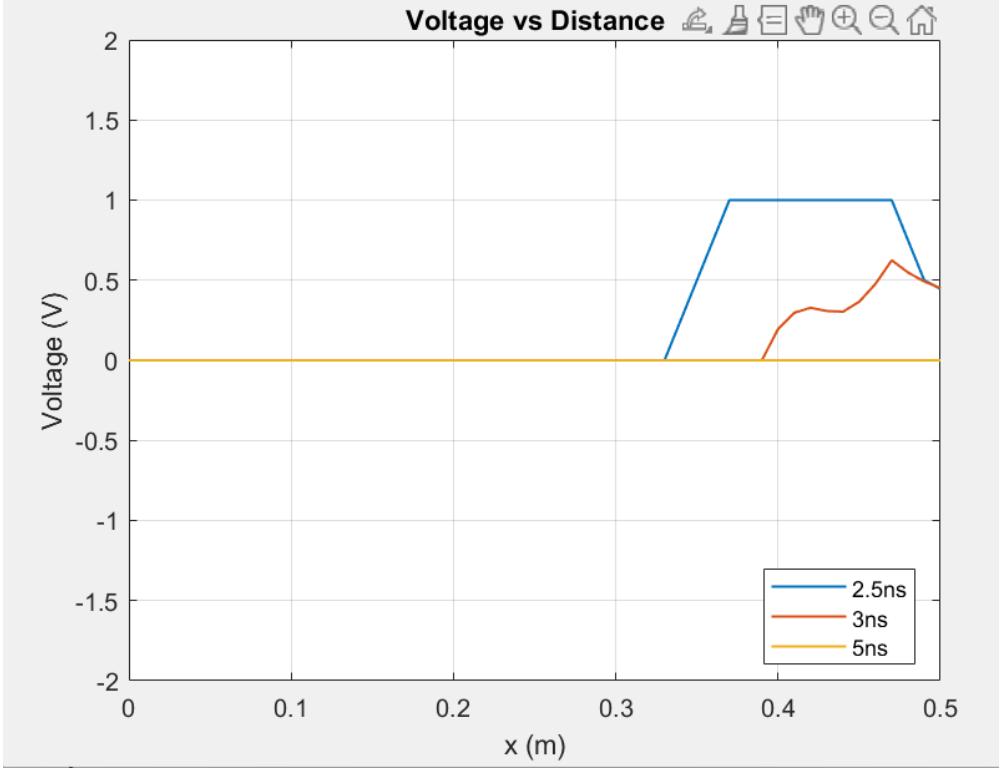

Figure 19: Case study 7 results

Summary

In conclusion, it can be said that all cases have met the expected results. The FDTD method has been successfully applied for the propagation of a trapezoidal pulse along a 1D transmission line terminated with different loads configurations. As a student with little exposure to programming in MATLAB, it's been a real struggle to understand the syntax and commands for the program. However, it's also been a rewarding experience. I feel more confident in terms of programing and understanding real life scenarios with the use of MATLAB. At all cases, it can be demonstrated how reflections, charges, discharges, and losses exist during propagation. This has been compared and analyzed concluding that the wave behavior will be influenced by the load configurations as well as the magic time step.

References

SciPy Community. (n.d.). scipy.integrate.solve_ivp. *SciPy v1.10.0 Manual*. Retrieved 2/27/2025 from https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html

MathWorks. (n.d.). *MATLAB documentation (Version R2024a)*. Retrieved 2/25/2025, from https://www.mathworks.com/help

MATLAB Crash Course for Beginners (2022, November 30). *Fundaments of MATLAB*. YouTube. Retrieved 2/22/2025 from https://www.youtube.com/watch?v=7f50sQYjNRA

Gedney, S.D. (2010). *Introduction to the Finite-Difference Time-Domain (FDTD) Method for Electromagnetics.* Springer Nature Switzerland AG. 10.1007/978-3-031-01712-4