AMME2000 - Assignment 2, 2024

Riley Smith, SID: 530618904

May 2024

1 Wave Equation

1.1 Initial Conditions and Boundary Conditions

Below are the initial conditions of the problem stated mathematically. Equation 1 is the initial displacement condition and Equation 3 is the initial velocity condition.

$$u(x, 0) = 0 \tag{1}$$

$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0.05 \sin\left(\frac{3\pi x}{L}\right) + 0.02 \sin\left(\frac{8\pi x}{L}\right) \tag{2}$$

The boundary conditions of the problem can also be identified as the tensioned cable is fixed at both ends.

$$u(0, t) = u(L, t) = 0 \tag{3}$$

1.2 Applying the Initial Displacement Condition

The general solution to the wave equation can be seen in Equation 4. It is only valid if the boundary conditions are $u(0, t) = u(L, t) = 0$. This applies to this problem, as can be seen in Equation 3.

$$u(x, t) = \sum_{n=0}^{\infty} \Big[A_n \cos(\lambda_n \cdot t) + B_n \sin(\lambda_n \cdot t)\Big] \sin\left(\frac{n\pi x}{L}\right) \tag{4}$$

Where,

$$\lambda_n = \frac{n\pi c}{L} \qquad c = \sqrt{\frac{T}{\rho}} \tag{5}$$

We can then apply the initial Displacement condition (Equation 1) to reduce the general solution.

$$u(x, 0) = \sum_{n=0}^{\infty} \Big[A_n \cos(\lambda_n \cdot 0) + B_n \sin(\lambda_n \cdot 0)\Big] \sin\left(\frac{n\pi x}{L}\right) = 0 \tag{6}$$

This then simplifies to:

$$u(x, 0) = \sum_{n=0}^{\infty} A_n \sin\left(\frac{n\pi x}{L}\right) = 0 \tag{7}$$

This equation must equal zero for all x-values. Because $\sin(\frac{n\pi x}{L})$ will only equal zero when $n = 0$. Due to this, all A_n values must be equal to zero. This is because the sine functions for different n are unique and will not cancel each other out. Thus, the only way to satisfy Equation 7 is:

$$A_n = 0 \tag{8}$$

We can then apply this to the general solution in Equation 4.

$$u(x, t) = \sum_{n=1}^{\infty} = B_n \sin(\lambda_n \cdot t) \sin\left(\frac{n\pi x}{L}\right) \tag{9}$$

The lower limit of the summation has changed from $n = 0$ to $n = 1$. This is because if $n = 0$, $sin\Big(\frac{n\pi x}{L}\Big) = 0$, meaning the equation will be equal to 0, having no effect on the value of the sum.

1.3 B_n Coefficients

Using the initial velocity condition (Equation 2), the B_n coefficients can be calculated. Firstly, we will differentiate Equation 9, which is:

$$\frac{\partial u}{\partial t} = \sum_{n=1}^{\infty} B_n \lambda_n \sin\left(\frac{n\pi x}{L}\right)\cos(\lambda_n t) \tag{10}$$

Then, solving this for $t = 0$, like the initial condition, gives:

$$\frac{\partial u}{\partial t}\bigg|_{t=0} = \sum_{n=1}^{\infty} B_n \lambda_n \sin\left(\frac{n\pi x}{L}\right) \tag{11}$$

We can then set the amplitude 'section' of the equation to equal that of the initial velocity conditions for their respective n.

$$B_3\frac{3\pi c}{L} = 0.05 \qquad\qquad B_8\frac{8\pi c}{L} = 0.02 \tag{12}$$

We can then solve for the B_n coefficients,

$$B_3 = \frac{0.05L}{3\pi c} \qquad\qquad B_8 = \frac{0.02L}{8\pi c} \tag{13}$$

As there are no other modes in the initial velocity equation, we can assume that for all other modes $B_n = 0$.

$$B_3 = \frac{0.05L}{3\pi c} \qquad\qquad B_8 = \frac{0.02L}{8\pi c} \qquad\qquad \text{otherwise } B_n = 0 \tag{14}$$

We can now substitute in these values in order to write the full analytical solution to the problem.

$$u(x,t) = \frac{0.05L}{3\pi c} sin\Big(\frac{3\pi c}{L}t\Big) sin\Big(\frac{3\pi x}{L}\Big) + \frac{0.02L}{8\pi c} sin\Big(\frac{8\pi c}{L}t\Big) sin\Big(\frac{8\pi x}{L}\Big) \tag{15}$$

1.4 Analytical Solution MATLAB Implementation

From the analytical solution found in Equation 15, we can implement it into MATLAB to model it.

As seen in Figure 1, the plot shows the displacement of the cable at different times. Initially, the displacement is zero everywhere, this reflects the initial condition of the problem. As time increases, the cable presents standing wave patterns due to the initial velocity condition.

1.5 'Starting Problem'

The starting problem arises in the CTCS (Central Time Central Space) numerical scheme, which calculates displacement at a future timestep in terms of the current and previous timesteps. While this approach generally works well, it becomes problematic at $t = 0$s because the scheme attempts to reference a displacement at $t = -1$s, which is not defined.

$$u_i^{n+1} = 2u_i^n - \mathbf{u_i^{n-1}} + \alpha^2(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \tag{16}$$

In order to resolve this we can estimate initial velocity, which is known in this problem, using a central difference. This produces a u_i^{n-1} which we can rearrange for. Assuming $u_t(x,0) = g(x)$

$$u_{t,i}^n \approx \frac{u_i^1 + u_i^{-1}}{2\Delta t} = g \tag{17}$$

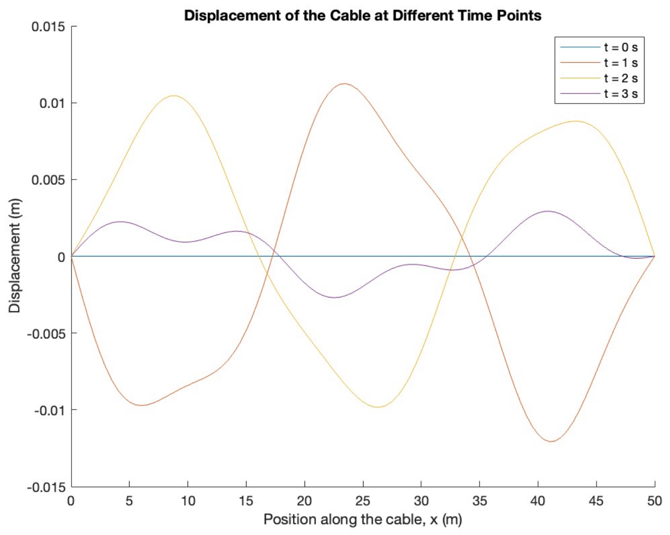

Figure 1: Plot of the Analytical Solution at different time points

$$u_i^{-1} = u_i^1 - 2\Delta t g \tag{18}$$

From here we can substitute this into Equation 16 in order to solve for the first time step in the CTCS scheme.

$$u_i^1 = 2u_i^0 - (u_i^1 - 2\Delta t g) + \alpha^2(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0) \tag{19}$$

$$2u_i^1 = 2u_i^0 + 2\Delta t g + \alpha^2(u_{i+1}^0 - 2u_i^0 + u_{i-1}^0) \tag{20}$$

The lines of code used to implement this can be found below:

```
% Initial velocity profile
v = 0.05 * sin(3 * pi * x / L) + 0.02 * sin(8 * pi * x / L);

% Solution for the first timestep using the initial velocity
for i = 2:nx-1
    u(2, i) = u(1, i) + dt * v(i) + 0.5 * sigma * (u(1, i+1) - 2 * u
        (1, i) + u(1, i-1));
end
```

1.6 CTCS Scheme MATLAB Implementation

As seen in Figure 2, the numerical and analytical solutions are overlaid at the same times in order to determine the accuracy of the numerical solution. The numerical solution is very similar to the analytical, only deviating slightly. The biggest difference can be seen when $t = 3$s around $x = 9$m, albeit a very small difference. Overall, the numerical solution is very similar compared to the analytical solution.

To calculate the maximum time step for stability, the equation $dt = \frac{dx}{c}$. This is found through a von Neumann stability analysis. This step can be found in my code which generated the plot in the appendix.

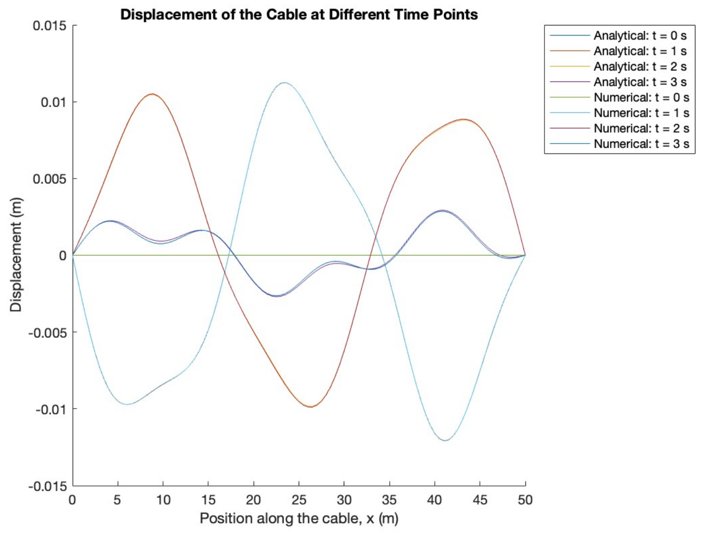

Figure 2: Plot of the Analytical and Numerical Solution overlaid

1.7 Maximum Displacement

Figure 3 shows the change in displacement over the first 10 seconds at different points along the x-axis. The positions $x = 15$m and $x = 20$m show more pronounced oscillations compared to $x = 45$m, reflecting the constructive and destructive interference of the wave components at these points. The maximum displacement over the first 100s was found to be 0.0133m at x = 41m when t = 86.2s. The code which found this can be found in the Appendix.

1.8 Forcing Effect Implementation

The forcing term discretised can be seen below:

$$F_i^n = 0.14\cos(2\pi \cdot f \cdot dt \cdot n)\sin(\frac{2\pi \cdot f \cdot dx \cdot i}{c} \tag{21}$$

After implementing the forcing term into the numerical solution, there are some apparent changes to the behaviour of the wire over time. As time increases, the forcing term begins to dominate the equation, displacement increases, as well as a node is formed around x = 25m. If the tightrope walker was at x = 30m, in order to stay safe, they should make their way to x = 25m where displacement is the least and it is the most stable. The displacement first reaches 0.25m at t = 111.22s, this was found using the code found in the Appendix.

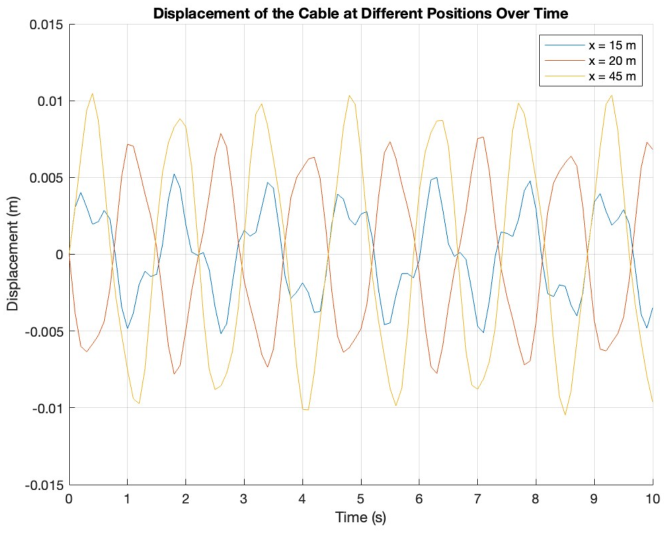

Figure 3: Plot of the displacement at x = 15, 20, 45m over the first 10s

2 Laplace Equation

2.1 Pressure Boundary Conditions

The pressure boundary conditions of the problem can be stated mathematically as below,

$$P(x, L_y) = \begin{cases} \frac{2P_1 x}{L_x} & 0 \leq x \leq \frac{L_x}{2} \\ -\frac{2P_1 x}{L_x} + 2P_1 & \frac{L_x}{2} \leq x \leq L_x \end{cases} \tag{22}$$

$$P(x,0) = P(0,y) = P(L_x, y) = 0 \tag{23}$$

2.2 Separation of Variables

We can then use seperation of variables on the general Laplace Equation in order to solve it analytically. The general equation is as shown below.

$$\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} \qquad P_{xx} + P_{yy} = 0 \tag{24}$$

We can assume,

$$P(x,y) = F(x) \cdot G(y) \tag{25}$$

Thus,

$$\frac{G}{F_{xx}} + \frac{F}{G_{yy}} = 0 \tag{26}$$

$$\frac{F_{xx}}{F} = -\frac{G_{yy}}{G} = -p^2 \tag{27}$$

This then forms two ODE's which can be seen below:

$$F_{xx} + p^2 F = 0 \tag{28}$$

$$G_{yy} - p^2 G = 0 \tag{29}$$

These ODE's can now be identified, then be solved to find the steady-state pressure distribuition of the problem.

2.2.1 x ODE

The x ODE is given in the form of a second-order linear homogeneous differential equation.

$$F_{xx} + p^2 F = 0 \tag{30}$$

The general solution to this differential equation is:

$$F(x) = a\cos(px) + b\sin(px) \tag{31}$$

We can apply the boundary conditions of the problem to solve for $F(x)$.

$$F(0) = a(1) + b(0) = 0, \quad a = 0 \tag{32}$$

$$F(L_x) = b\sin(pL_x) = 0 \tag{33}$$

b cannot be equal to 0 as that would mean $F(x) = 0$, thus the $\sin(pL_x)$ term must be equal to 0.

$$b \neq 0, \qquad pL_x = n\pi \tag{34}$$

For this to be true $pL_x = n\pi$ where $n = 1, 2, 3, ...$ as $\sin(n\pi)$ is always equal to 0. Rearranging for p gives:

$$p = \frac{n\pi}{L_x} \tag{35}$$

Substituting this into F(x) gives the x ODE where b_n will be found later.

$$F_n(x) = b_n \sin(px) \tag{36}$$

2.2.2 y ODE

The y ODE is given in the form of a second-order linear differential equation

$$G_{yy} - p^2 G = 0 \tag{37}$$

The general solution to this differential equation is:

$$G_n(y) = A_n e^{py} + B_n e^{-py} \tag{38}$$

We can apply a boundary condition of the problem to solve for $G(y)$.

$$G_n(0) = A_n(1) + B_n(1) = 0 \tag{39}$$

$$A_n = -B_n \tag{40}$$

Substituting this into $G(y)$ gives:

$$G_n(y) = A_n(e^{py} - e^{-py}) \implies 2A_n \sinh(py) \tag{41}$$

2.2.3 Complete Steady-State Pressure Solution

After $G_n(y)$ and $F_n(x_)$ have been solved, we can substitute in these values into the equation, $P(x,y)F(x) \cdot G(y)$.

$$P(x,y) = A_n^* \sinh(py)\sin(px) \tag{42}$$

$$P(x,y) = A_n^* \sinh(\frac{n\pi y}{L_x})\sin(\frac{n\pi x}{L_x}) \tag{43}$$

Where A_n^* is the combination of the coefficients present in $F_n(x)$ and $G_n(y)$.

2.3 Complete Solution for $P(x,y)$

The value of A_n^* can be found in the tabulated Fourier coefficients in the datasheet. A_n^* can be seen below where the boundary condition is a triangular function.

$$A_n^* = \frac{8P_1 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \tag{44}$$

Thus the complete solution can be written as:

$$P(x,y) = \sum_{n=1}^{\infty} \frac{8P_1 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \sinh(\frac{n\pi y}{L_x})\sin(\frac{n\pi x}{L_x}) \tag{45}$$

2.4 Adaptation of $P(x,y)$ for New Scenario

In order to model the cracker being held from two sides, i.e pressure on two sides of the cracker. We can use the existing solution for $P(x,y)$ as found in Section 2.3. To do this we must transform the system to a (x,q) coordinate system, where $q = y - L_y$. This means when $y = 0$, $q = -L_y$ and when $y = L_y$, $q = 0$. This transforms the following equations:

$$P(x,q) = \sum_{n=1}^{\infty} \frac{8P_2 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi \cdot -L_y}{L_x})} \sinh(\frac{n\pi q}{L_x})\sin(\frac{n\pi x}{L_x}) \tag{46}$$

This can then be transformed back to a (x,y) coordinate system.

$$P(x,y) = \sum_{n=1}^{\infty} -\frac{8P_2 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \sinh(\frac{n\pi(y-L_y)}{L_x})\sin(\frac{n\pi x}{L_x}) \tag{47}$$

This is then added to the initial $P(x,y)$ for the first problem in order to model the cracker being held from two sides.

$$P(x,y) = \sum_{n=1}^{\infty} \frac{8P_1 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \sinh(\frac{n\pi y}{L_x})\sin(\frac{n\pi x}{L_x}) - \frac{8P_2 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \sinh(\frac{n\pi(y-L_y)}{L_x})\sin(\frac{n\pi x}{L_x}) \tag{48}$$

This can be simplified into:

$$P(x,y) = \sum_{n=1}^{\infty} \frac{8P_1 \sin(\frac{n\pi}{2})}{n^2\pi^2 \sinh(\frac{n\pi L_y}{L_x})} \Big[P_1 \sin(\frac{n\pi y}{L_x} - P_2 \sin(\frac{n\pi(y-L_y)}{L_x}\Big] \sin(\frac{n\pi x}{L_x}) \tag{49}$$

2.5 Analytical Solution MATLAB Implementation

In Figure 4, the steady-state pressure distribution is shown across the shape. The distribution is consistent with the boundary conditions, with the pressure peaking at $x = \frac{L_x}{2}$ and then decreasing along the axis in both directions. This indicates that the pressure is highest at the midpoint of the x-axis and diminishes symmetrically towards the boundaries. The figure also shows that the pressure is applied from both sides, which aligns with the given problem setup. The symmetrical pattern observed in the pressure distribution confirms the expected behaviour under the specified boundary conditions, illustrating a balanced and steady state within the shape.

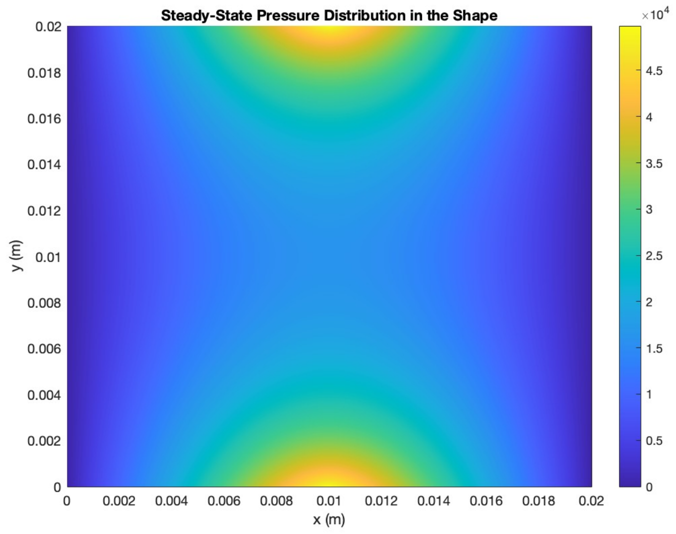

Figure 4: Plot of the Steady-state Pressure Distribution Analytical Solution

2.6 Gauss-Jacobi Numerical Scheme MATLAB Implementation

The Gauss-Jacobi Numerical scheme can be stated mathematically as seen below:

$$P_{i,j} = \frac{1}{4}(P_{i+1,j} + P_{i-1,j} + P_{i,j+1} + P_{i,j-1}) \tag{50}$$

The initial guess for the pressure distribution, P, is set to zero for all interior points, while the boundary conditions are applied to the edges. The boundary conditions are applied using the following lines of MATLAB code.

```
% L_x/2
peakIndex = round(nx / 2);
for i = 1:nx
    if i <= peakIndex
        P(1, i) = P2 * (i-1) / (peakIndex-1);
        P(end, i) = P1 * (i-1) / (peakIndex-1);
    else
        P(1, i) = P2 * (nx-i) / (nx-peakIndex);
        P(end, i) = P1 * (nx-i) / (nx-peakIndex);
    end
end
```

The convergence of the Gauss-Jacobi iterative scheme is tested by evaluating the maximum absolute difference between the pressure values of consecutive iterations. The scheme is considered to have converged when this difference falls below a specified tolerance. This is checked throughout the code shown by the MATLAB code below.

```
    err = 1;
    tol = 1e-10; % Tolerance for convergence
    while err > tol
        %Numercal Scheme Here
        err = max(max(abs(P - P_old)));
    end
```

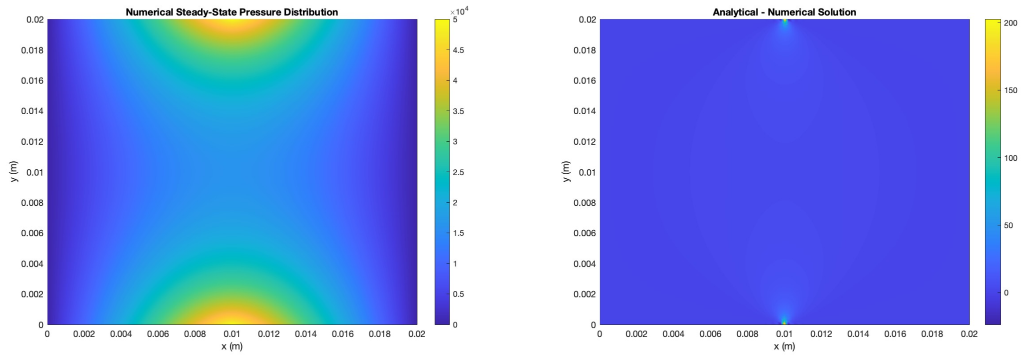

Figure 5: Plot of the Numerical Solution (Left) and the I_{diff} Plot $[I_{diff} = I_{Analytical} - I_{Numerical}]$ (Right)

2.7 Maximum L_y Value

The maximum L_y value that solves the inequality $I_x P_{yaverage} \leq 9.25 \cdot 10^{-4}$, where $I_x = \frac{L_x(L_y)^3}{12}$, can be found using the MATLAB code found in the Appendix. The maximum L_y value, to the nearest 1/2mm, is 31.5mm or 0.0315m. As seen by the output of the MATLAB code:

$$\textbf{Maximum Ly such that the shape does not break: 0.0315 m} \tag{51}$$

3 Fourier Integral

3.1 Boundary Conditions

The boundary conditions for this problem can be stated mathematically as shown below:

$$Q(-\infty,t) = Q(+\infty,t) = 0 \tag{52}$$

An analytical solution cannot be obtained using Fourier Series for this problem as it has an infinite domain. Fourier Series are used for problems which have a finite domain and defined boundary conditions over this domain. This problem has neither of these conditions, thus Fourier Series cannot be used for to solve this problem.

3.2 Simplification of Fourier Integral

The Fourier Integral of the heat equation is given by:

$$Q(x,t) = \frac{1}{\pi}\int_0^\infty \Big[\int_{-\infty}^\infty f(v)\cos(px-pv)e^{-Dp^2t}\,dv\Big]\,dp \tag{53}$$

This equation can then be simplified using this integral,

$$\int_0^\infty e^{-s^2}\cos(2bs)\,ds = \frac{\sqrt{\pi}}{2}e^{-b^2} \tag{54}$$

To do this, we must first change the order of integration.

$$Q(x,t) = \frac{1}{\pi}\int_{-\infty}^\infty f(v)\Big[\int_0^\infty \cos(px-pv)e^{-Dp^2t}\,dp\Big]\,dv \tag{55}$$

We are only solving for the integral inside of the square brackets. From here, we need to transform the integral into that of the form of Equation 54 Let,

$$s^2 = Dp^2t \qquad s = p\sqrt{Dt} \qquad p = \frac{s}{\sqrt{Dt}} \tag{56}$$

$$px - pv = 2bs \qquad b = \frac{p(x-v)}{2s} \qquad b = \frac{x-v}{2\sqrt{Dt}} \tag{57}$$

From here, we can solve for dp in terms of ds.

$$\frac{dp}{ds} = \frac{1}{\sqrt{Dt}} \qquad dp = \frac{ds}{\sqrt{Dt}} \tag{58}$$

We can now substitute these terms we just solved for into Equation 55

$$\int_0^\infty \cos(px-pv)e^{-Dp^2t}\,dp \;\Rightarrow\; \int_0^\infty \cos(2bs)e^{-s^2}\,\frac{ds}{\sqrt{Dt}} \tag{59}$$

$$\frac{1}{\sqrt{Dt}}\int_0^\infty \cos(2bs)e^{-s^2}\,ds \tag{60}$$

We can now compute the integral given the solution in Equation 54

$$\frac{1}{\sqrt{Dt}}\Big[\frac{\sqrt{\pi}}{2}e^{-b^2}\Big] \;\Rightarrow\; \frac{\sqrt{\pi}}{2\sqrt{Dt}}e^{-\frac{(x-v)^2}{4Dt}} \tag{61}$$

Substituting this into Equation 55 gives:

$$Q(x,t) = \frac{1}{2\sqrt{D\pi t}}\int_{-\infty}^{\infty} f(v)e^{-\frac{(x-v)^2}{4Dt}}\ dv \tag{62}$$

3.3 $Q(x,t)$ as a Sum of Error Functions

To obtain $Q(x,t)$ as a sum of error functions we must solve the remaining integral in Equation 62 However, we will first apply the initial condition, $f(v)$.

$$Q(x,0) = f(v) = \begin{cases} Q_0 & \text{if } -a < x < a \\ 0 & \text{otherwise} \end{cases} \tag{63}$$

Applying this to Equation 62 gives

$$Q(x,t) = \frac{Q_0}{2\sqrt{D\pi t}}\int_{-a}^{a} e^{-\frac{(x-v)^2}{4Dt}} \tag{64}$$

To solve the integral in Equation 64 we must use the integral below:

$$\int_0^x e^{-z^2} dz = \frac{\sqrt{\pi}}{2}\text{erf}(x) \tag{65}$$

To do this we must substitute terms in Equation 64 for terms in Equation 65

$$z^2 = \frac{(x-v)^2}{4Dt} \qquad\qquad z = \pm\frac{(x-v)}{2\sqrt{Dt}} \tag{66}$$

We will take the negative root of z.

$$\frac{dz}{dv} = \frac{1}{2\sqrt{Dt}} \qquad\qquad dv = 2\sqrt{Dt}\ dz \tag{67}$$

We then change the limits of the integral

$$z(a) = \frac{(a-x)}{2\sqrt{Dt}} \qquad\qquad z(-a) = -\frac{(x+a)}{2\sqrt{Dt}} \tag{68}$$

Substituting these terms into Equation 64 gives:

$$Q(x,t) = \frac{Q_0}{\sqrt{\pi}}\int_{-\frac{(x+a)}{2\sqrt{Dt}}}^{\frac{(a-x)}{2\sqrt{Dt}}} e^{-z^2}\ dz \tag{69}$$

This can then be rewritten in the form of

$$Q(x,t) = \frac{Q_0}{\sqrt{\pi}}\Big[\int_{-\frac{(x+a)}{2\sqrt{Dt}}}^{0} e^{-z^2}\ dz + \int_0^{\frac{(a-x)}{2\sqrt{Dt}}} e^{-z^2}\ dz\Big] \tag{70}$$

$$Q(x,t) = \frac{Q_0}{\sqrt{\pi}}\Big[-\int_0^{-\frac{(x+a)}{2\sqrt{Dt}}} e^{-z^2}\ dz + \int_0^{\frac{(a-x)}{2\sqrt{Dt}}} e^{-z^2}\ dz\Big] \tag{71}$$

The equation is now in the form of the integral in Equation 65 it can then be solved for,

$$Q(x,t) = \frac{Q_0}{2}\Big[-\operatorname{erf}\Big(-\frac{(a+x)}{2\sqrt{Dt}}\Big) + \operatorname{erf}\Big(\frac{a-x}{2\sqrt{Dt}}\Big)\Big] \tag{72}$$

As the error function is an odd function, i.e $-\operatorname{erf}(-x) = \operatorname{erf}(x)$. Equation 72 can be rewritten as

$$Q(x,t) = \frac{Q_0}{2}\Big[\operatorname{erf}\Big(\frac{a+x}{2\sqrt{Dt}}\Big) + \operatorname{erf}\Big(\frac{a-x}{2\sqrt{Dt}}\Big)\Big] \tag{73}$$

3.4 Changes in Maximum Charge over Time

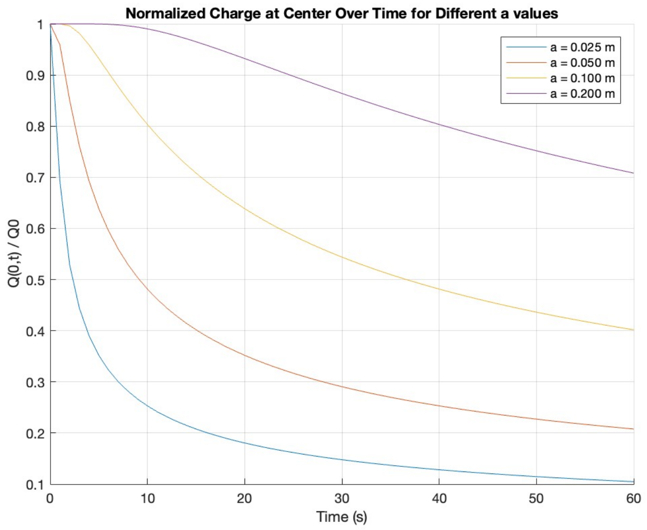

Figure 6: Plot of the Maximum Charge value changing with time at different a values

As seen in Figure 6, the maximum charge decreases over time as it diffuses across the wire. As the value of a increases, the rate at which the charge diffuses is more gradual, this is because there is a greater area of initial charge across the wire. For smaller values of a, the charge diffuses more quickly, leading to a steeper decline in the normalized charge over time.

A MATLAB Code

A.1 Wave Equation

A.1.1 Analytical Solution MATLAB Implementation

```
function A = Question1_4(time_points, L, T, rho, x)
    c = sqrt(T / rho); % Calculate the wave speed
    A = 0;
    % Given initial velocity coefficients
    B3 = (0.05 * L) / (3 * pi * c);
    B8 = (0.02 * L) / (8 * pi * c);

    % Calculate and plot displacement for each time point
    for t = time_points
        % Displacement calculation using vectorized form
        A = B3 * sin(3 * pi * c * t/L)*sin(3 * pi * x/L)+...
            B8 * sin(8 * pi * c * t/L)*sin(8 * pi * x/L);
    end
end

% Given parameters
L = 50; % Length of the cable in meters
T = 5e3; % Tension in Newtons
rho = 9.86; % Linear density in kg/m
time_points=[0,1,2,3];

figure;
hold on;

for time = time_points
    x = linspace(0, L, nx);
    A = Question1_4(time, L, T, rho, x);
    plot(x, A, 'DisplayName', sprintf('t = %d s', time));
end

% Customize plot
xlabel('Position along the cable, x (m)');
ylabel('Displacement (m)');
title('Displacement of the Cable at Different Time Points');
legend('show');
hold off;
```

A.1.2 CTCS Scheme MATLAB Implementation

```
function [x, t, u] = WaveNumericalSolution(tf, nx, L, T, rho)
```

```
% Input :
%    tf - end time for the solution [s]
%    nx - number of points in the spatial domain
%    L - Length of the cable [m]
%    T - Tension [N]
%    rho - Linear density [kg/m]
% Output :
%    x - x-domain points [m]
%    t - time domain points [s]
%    u - matrix of displacement values, row n is solution at timestep n [m
   ]

% Wave speed [m/s]
c = sqrt(T / rho);

% Generate domain
dx = L / (nx - 1);          % x-domain spacing [m]
x = linspace(0, L, nx);     % x-domain [m]
dt = dx / c;             % Maximum stable timestep [s]
nt = ceil(tf / dt + 1); % Number of timesteps
t = 0:dt:tf;                % Time domain [s]

% Coefficient in CTCS scheme
sigma = (dt * c / dx)^2;

% Initialize solution array
u = zeros(nt, nx);

% Initial velocity profile
v = 0.05 * sin(3 * pi * x / L) + 0.02 * sin(8 * pi * x / L);

% Solution for the first timestep using the initial velocity
for i = 2:nx-1
    u(2, i) = u(1, i) + dt * v(i) + 0.5 * sigma * (u(1, i+1) - 2 * u(1, i)
        + u(1, i-1));
end
u(2, 1) = 0; u(2, nx) = 0; % Boundary conditions

% Solution for the remaining timesteps
for n = 2:nt-1
    for i = 2:nx-1
        u(n+1, i) = 2 * u(n, i) - u(n-1, i) + sigma * (u(n, i+1) - 2 * u(n
           , i) + u(n, i-1));
    end
    u(n+1, 1) = 0; u(n+1, nx) = 0; % Boundary conditions
end
```

```
return


% Given parameters
L = 50; % Length of the cable in meters
T = 5e3; % Tension in Newtons
rho = 9.86; % Linear density in kg/m
time_points = [0, 1, 2, 3]; % Time points to plot
nx = 101; % Number of spatial points
tf = max(time_points); % End time for the solution

% Call the WaveNumericalSolution function
[x, t, u] = WaveNumericalSolution(tf, nx, L, T, rho);

% Plotting
figure;
hold on;

for time = time_points
    x = linspace(0, L, nx);
    A = WaveAnalyticalSolution(time, L, T, rho, x);
    plot(x, A, 'DisplayName', sprintf('Analytical: t = %d s', time));
end

% Plot numerical solutions for different time points
for k = 1:length(time_points)
    time_index = round(time_points(k) / (tf / (length(t)-1))) + 1;
    plot(x, u(time_index, :), 'DisplayName', sprintf('Numerical: t = %d s'
        , time_points(k)));
end

% Customize plot
legend('Location','bestoutside')
xlabel('Position along the cable, x (m)');
ylabel('Displacement (m)');
title('Displacement of the Cable at Different Time Points');
legend('show');
hold off;
```

A.1.3 Maximum Displacement

```
% Given parameters
L = 50; % Length of the cable in meters
T = 5e3; % Tension in Newtons
rho = 9.86; % Linear density in kg/m
```

```
time_points = 0:0.1:10; % Time points from 0 to 10 seconds in increments
    of 0.1 second
positions = [15, 20, 45]; % Positions along the cable to plot

% Calculate displacement using the analytical solution
displacements = zeros(length(time_points), length(positions));
x = positions;

for i = 1:length(positions)
    displacements(:, i) = WaveAnalyticalSolution(time_points, L, T, rho,
        positions(i));
end

% Plotting
figure;
hold on;

% Plot displacement at specified positions as a function of time
for i = 1:length(positions)
    plot(time_points, displacements(:, i), 'DisplayName', sprintf('x = %d
        m', positions(i)));
end

% Customize plot
xlabel('Time (s)');
ylabel('Displacement (m)');
title('Displacement of the Cable at Different Positions Over Time');
legend('show');
grid on;
hold off;



% Given parameters
L = 50; % Length of the cable in meters
T = 5e3; % Tension in Newtons
rho = 9.86; % Linear density in kg/m
time_points = 0:0.1:100; % Time points from 0 to 100 seconds in increments
     of 0.1 second
nx = 101; % Number of spatial points
positions = linspace(0, L, nx); % Positions along the cable

% Calculate displacement using the analytical solution
displacements = zeros(length(time_points), length(positions));

for i = 1:length(positions)
```

```
    displacements(:, i) = WaveAnalyticalSolution(time_points, L, T, rho,
        positions(i));
end

% Find maximum displacement over the entire time period at any position
[max_displacement, max_index] = max(displacements(:));
[max_time_index, max_pos_index] = ind2sub(size(displacements), max_index);

max_time = time_points(max_time_index);
max_position = positions(max_pos_index);

% Display the maximum displacement and its position and time
fprintf('Maximum displacement is %.4f m at position x = %.2f m and time t
    = %.2f s\n', max_displacement, max_position, max_time);
```

A.1.4 Forcing Effect Implementation

```
    function [x, t, u] = WaveNumericalSolution(tf, nx, L, T, rho)
% WaveNumericalSolution - Numerical solution to the wave equation
% with a forcing term using the CTCS scheme
% Input :
%    tf - end time for the solution [s]
%    nx - number of points in the spatial domain
%    L - Length of the cable [m]
%    T - Tension [N]
%    rho - Linear density [kg/m]
% Output :
%    x - x-domain points [m]
%    t - time domain points [s]
%    u - matrix of displacement values, row n is solution at timestep n [m
    ]

% Wave speed [m/s]
c = sqrt(T / rho);

% Generate domain
dx = L / (nx - 1);          % x-domain spacing [m]
x = linspace(0, L, nx);    % x-domain [m]
dt = dx / c;                % Timestep [s]
nt = ceil(tf / dt + 1);    % Number of timesteps
t = 0:dt:tf;                % Time domain [s]

% Coefficient in CTCS scheme
sigma = (dt * c / dx)^2;

% Initialize solution array
```

```
u = zeros(nt, nx);

% Initial velocity profile
v = 0.05 * sin(3 * pi * x / L) + 0.02 * sin(8 * pi * x / L);

% Solution for the first timestep using the initial velocity
for i = 2:nx-1
    F = 0.14 * cos(2 * pi * 540 * dt * 0) * sin(2 * pi * dx * (i) / L);
    u(2, i) = u(1, i) + dt * v(i) + 0.5 * sigma * (u(1, i+1) - 2 * u(1, i)
        + u(1, i-1)) + (dt^2 / rho) * F;
end
u(2, 1) = 0; u(2, nx) = 0; % Boundary conditions

% Solution for the remaining timesteps
for n = 2:nt-1
    for i = 2:nx-1
        F = 0.14 * cos(2 * pi * 540 * dt * n) * sin(2 * pi * dx * (i) / L)
            ;
        u(n+1, i) = 2 * u(n, i) - u(n-1, i) + sigma * (u(n, i+1) - 2 * u(n
            , i) + u(n, i-1)) + (dt^2 / rho) * F;
    end
    u(n+1, 1) = 0; u(n+1, nx) = 0; % Boundary conditions
end

threshold = 0.25;
reached_threshold = false;
for n = 1:nt
    if any(abs(u(n, :)) >= threshold)
        fprintf('Displacement first reaches %.2f m at t = %.2f s\n',
           threshold, t(n));
        reached_threshold = true;
        break;
    end
end

return
```

A.2 Laplace Equation

A.2.1 Analytical Solution MATLAB Implementation

```
    function P = Question2_5
% Given parameters
Lx = 20e-3;
Ly = 20e-3;
P1 = 5e4;
```

```
P2 = 5e4;
nx = 101;
ny = 101;
N = 100;


% Generate grid
x = linspace(0, Lx, nx);
y = linspace(0, Ly, ny);
[X, Y] = meshgrid(x, y);

% Compute P(x, y)
P = zeros(ny, nx);
for i = 1:nx
    for j = 1:ny
        xi = X(j,i);
        yj = Y(j,i);
        % Sum Fourier terms
        for n = 1:N
            An = (8 * sin(n*pi/2)) / (n^2 * pi^2 * sinh(n*pi*Ly/Lx));
            term1 =  An * P1 * sinh((n * pi * yj) / Lx);
            term2 = -An * P2 * sinh((n * pi * (yj-Ly)) / Lx);
            P(j, i) = P(j, i) + (term1 + term2) * sin(n * pi * xi / Lx);
        end

    end
end
% Plot

figure;
pcolor(X, Y, P);
shading interp;
colorbar;
xlabel('x (m)');
ylabel('y (m)');
title('Steady-State Pressure Distribution in the Shape');
```

A.2.2 Gauss-Jacobi Numerical Solution

```
    % Combined Script to Call Question2_5 and Question2_6 and Plot Side by
        Side

% Given parameters
Lx = 20e-3;
Ly = 20e-3;
P1 = 5e4;
```

```matlab
P2 = 5e4;
nx = 101;
ny = 101;
N = 100;

% Call functions
P_2_5 = Question2_5(Lx, Ly, nx, ny, N, P1, P2);
P_2_6 = Question2_6(Lx, Ly, nx, ny, N, P1, P2);

% Generate grid
x = linspace(0, Lx, nx);
y = linspace(0, Ly, ny);
[X, Y] = meshgrid(x, y);

% Create subplots
figure;

% Subplot for Numerical Solution
subplot(1, 2, 1);
pcolor(X, Y, P_2_6);
shading interp;
colorbar;
xlabel('x (m)');
ylabel('y (m)');
title('Numerical Steady-State Pressure Distribution');

% Subplot for I diff
subplot(1, 2, 2);
pcolor(X, Y, P_2_6-P_2_5);
shading interp;
colorbar;
xlabel('x (m)');
ylabel('y (m)');
title('Analytical - Numerical Solution');

function P = Question2_6(Lx, Ly, nx, ny, N, P1, P2)
    P = zeros(ny, nx); % Initial guess
    % Set the x-axis grid
    x = linspace(0, Lx, nx);

    % Boundary conditions
    peakIndex = round(nx / 2);
    for i = 1:nx
        if i <= peakIndex
            P(1, i) = P2 * (i-1) / (peakIndex-1);
            P(end, i) = P1 * (i-1) / (peakIndex-1);
        else
            P(1, i) = P2 * (nx-i) / (nx-peakIndex);
            P(end, i) = P1 * (nx-i) / (nx-peakIndex);
        end
    end

    err = 1;a
    tol = 1e-10; % Tolerance for convergence
    while err > tol
        P_old = P; % Store the previous values
        for j = 2:ny-1
            for i = 2:nx-1
                P(j, i) = 0.25 * (P_old(j+1, i) + P_old(j-1, i) + P_old(j,
                    i+1) + P_old(j, i-1));
            end
        end
        err = max(max(abs(P - P_old)));
    end
end

function P = Question2_5(Lx, Ly, nx, ny, N, P1, P2)
    % Generate grid
    x = linspace(0, Lx, nx);
    y = linspace(0, Ly, ny);
    [X, Y] = meshgrid(x, y);

    % Compute P(x, y)
    P = zeros(ny, nx);
    for i = 1:nx
        for j = 1:ny
            xi = X(j,i);
            yj = Y(j,i);
            % Sum Fourier terms
            for n = 1:N
                An = (8 * sin(n*pi/2)) / (n^2 * pi^2 * sinh(n*pi*Ly/Lx));
                term1 =  An * P1 * sinh((n * pi * yj) / Lx);
                term2 = -An * P2 * sinh((n * pi * (yj-Ly)) / Lx);
                P(j, i) = P(j, i) + (term1 + term2) * sin(n * pi * xi / Lx
                    );
            end
        end
    end
end
```

A.2.3 Max L_y Value

```
function Ly_max = Question2_7(Lx)
% Constants
nx = 101; % Number of points in x
ny = 101; % Number of points in y
N = 100; % Number of Fourier terms
Pressure = 5e4; % Boundary condition pressure at y=Ly
threshold = 9.25e-4;
% Start Ly from a small value, e.g., the width of the shape
Ly = Lx;
dLy = 0.0005; % Increment for Ly (To the nearest 1/2 mm)

while true
    % Generate grid
    x = linspace(0, Lx, nx);
    y = linspace(0, Ly, ny);
    [X, Y] = meshgrid(x, y);

    % Compute P(x, y)
    P = zeros(ny, nx);
    for i = 1:nx
        for j = 1:ny
            xi = X(j,i);
            yj = Y(j,i);
            for n = 1:N
                An = (8 * sin(n*pi/2)) / (n^2 * pi^2 * sinh(n*pi*Ly/Lx
                    ));
                term1 =  An * Pressure * sinh((n * pi * yj) / Lx);
                term2 = -An * Pressure * sinh((n * pi * (yj-Ly)) / Lx)
                    ;
                P(j, i) = P(j, i) + (term1 + term2) * sin(n * pi * xi
                    / Lx);
            end
        end
    end

    % Calculate average pressure at central y-axis
    mid_x_index = round(nx/2);
    P_y_avg = mean(P(:, mid_x_index));

    % Check the inequality
    Ix = (Lx * Ly^3) / 12;
    if Ix * P_y_avg > threshold
        break;
    end

    % Increment Ly
```

```
        Ly = Ly + dLy;
    end

    % Last valid Ly
    Ly_max = Ly - dLy;
    fprintf('Maximum Ly such that the shape does not break: %.4f m\n',
        Ly_max);
end

Question2_7(0.02)
```

A.3 Fourier Integral

A.3.1 Changes in Maximum Charge over Time

```
% Temporal (t) parameters
max_time = 60;
nt = 61;
time = linspace(0, max_time, nt);

% Initialization
Q0 = 1;  % Normalization factor for charge
a_values = [0.025, 0.05, 0.1, 0.2];  % Different 'a' scenarios

% Prepare figure for plotting
figure;
hold on;
grid on;
legendEntries = strings(1, length(a_values));

% Computing and plotting Q(0,t) for each 'a'
for i = 1:length(a_values)
    a = a_values(i);
    Q_t = zeros(1, nt);  % Initialize Q_t array
    for n = 1:nt
        Q_t(n) = 0.5 * (erf((a + 0) / (2 * sqrt(3e-4 * time(n)))) + erf((a
             - 0) / (2 * sqrt(D * time(n)))));
    end
    plot(time, Q_t / Q0, 'DisplayName', sprintf('a = %.3f m', a));
    legendEntries(i) = sprintf('a = %.3f m', a);
end

% Customize the plot
xlabel('Time (s)');
ylabel('Q(0,t) / Q0');
title('Normalized Charge at Center Over Time for Different a values');
```

```
legend(legendEntries, 'Location', 'best');
hold off;
```