Wave Equation

PHYS 481 -- Assignment 5

Patrick Harrison 30023631 & Craig Michie 30001523

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
```

Introduction

A one dimensional wave on a string with length L say on the x-axis has a displacement $u(x,t)$ on the y-axis. u is given in the PDE

$$\frac{\partial^2 u}{\partial t^2} = c^2 \frac{\partial^2 u}{\partial x^2}$$

On the string example, $x \in (0, L)$ and $t \in (0, T]$. The shape of the string at time $t = 0$ is given by $\frac{\partial}{\partial t} u(x,0) = 0$. As this is a second order PDE we need to specify two inital conditions as

$$u(0,t) = 0 \qquad (1)$$
$$u(L,t) = 0 \qquad (2)$$

From this we know that the string is fixed at a displacment of zero at both ends. To solve the PDE of the string, We need to break the domain into discret parts in x and t. We can visualize a point x_i at time t_i as a two dimensional grid. This mesh must have spacing Δx and Δt.

Thus for a certain position in the mesh, the differential equation is

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

We can now impliment an approximation to the derivatives by uing finite differences.

$$\frac{\partial^2}{\partial t^2} u(x_i, t_n) \approx \frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2}$$

Implimenting this into the wave equation.

$$\frac{u_i^{n+1} - 2u_i^n + u_i^{n-1}}{\Delta t^2} = c^2 \frac{u_{i+1}^n - 2u_i^n + u_{i-1}^n}{\Delta x^2}$$

The Courant number denoted $C := c\frac{\Delta t}{\Delta x}$ relates the physical parameter c with the numerical parameters Δt and Δx. We can use this to rewrite the discrete wave equation to be:

$$u_i^{n+1} = -u_i^{n-1} + 2u_i^n + C^2(u_{i+1}^n - 2u_i^n + u_{i-1}^n) \qquad (*)$$

Explain that for a function $A\sin(kx)\cos(\omega t)$ the wave length in space is $\lambda = \frac{2\pi}{k}$ and the period in time is $P = \frac{2\pi}{\omega}$. Use these expressions to find the wave length in space and period in time of u_e above.

A function $\sin(kx)\cos(\omega t)$ has a wave number k where we can write k as:

$$k = \frac{2\pi}{\lambda} \qquad \text{Period} = \frac{\lambda}{2\pi}$$

The function $u_e(x,t) = A\sin\left(\frac{\pi}{L}mx\right)\cos\left(\frac{\pi}{L}mct\right)$ so we see

$$\begin{aligned} \frac{\pi}{L}m &= k & \frac{\pi}{L}mc &= \omega \\ \frac{\pi}{L}m &= \frac{2\pi}{\lambda} & \frac{\pi}{L}mc &= \frac{2\pi}{P} \\ \frac{2L}{m} &= \lambda & \frac{2L}{mc} &= P \end{aligned}$$

Impliment code to numerically and analytically solve the wave.

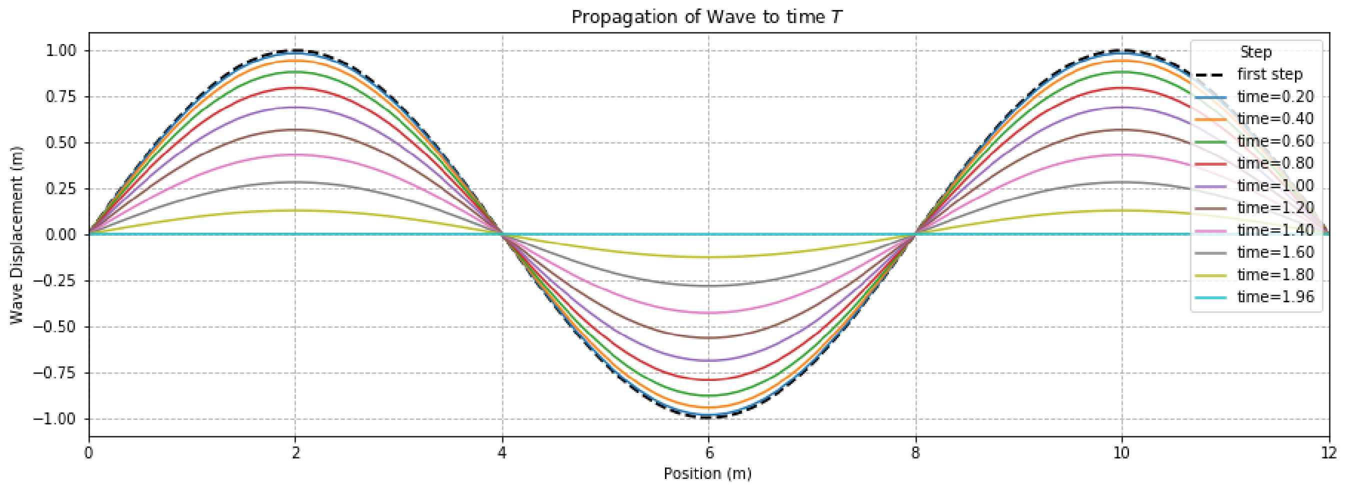
```
In [2]: class Wave:
            '''Waves like the queen of England'''
            def __init__(self, L, T, A, m, c):
                ''' function to set up physical and numerical parameters to solve the
                    problem.
                Parameters:
                    L: domain in x              (float or int)
                    T: time domain              (float or int)
                    A: amplitude of the wave    (float or int)
                    m: number of nodes          (int)
                    c: speed of wave            (float or int)
                '''
                # Set parameters
                self.L, self.T = L, T # initialize domain lengths
                self.x, self.t = np.linspace(0, L, 100), np.linspace(0, T, 100) # initialize domains
                self.Nx, self.Nt = len(self.x)-1, len(self.t)-1 # length for domains - 1
                self.dx, self.dt = self.x[1] - self.x[0], self.t[1]-self.t[0] # spacing of domains
                self.A, self.m, self.c, self.C2 = A, m, c, (c*self.dt/self.dx)**2 # physical parameters
                # compute first step
                self.first_step()

            def set_initial(self, x):
                '''Sets the inital state of the system
                Parameters:
                    x: x values to compute the list over (array)
                '''
                return self.A*np.sin((np.pi/self.L)*self.m*x) # intial state of system

            def analytical_soln(self, time=0):
                ''' Computes the analytical solution of the wave at a given time
                Parameters:
                    time: given time to comput the analytical solution at (float or int)
                '''
                self.analytic = self.A*np.sin((np.pi/self.L)*self.m*self.x) \
                                        *np.cos((np.pi/self.L)*self.m*self.c*time)

            def first_step(self):
                '''Computes the first step to the solution
                Parameters:
                    None
                '''
                # set grids for past present and future and initialize
                self.u_nm1, self.u, self.u_n, = np.zeros(len(self.x)), np.zeros(len(self.x)), np.zeros(
        len(self.x))
                self.u_n = self.set_initial(self.x[:]) # sets new grid to initial condition
                # compute centered difference
                self.u[1:-1] = self.u_n[1:-1] + 0.5*self.C2*(self.u_n[2:]-2*self.u_n[1:-1]+self.u_n[:-2
        ])
                self.u_nm1[:], self.u_n[:] = self.u_n, self.u # move forward in time
                # get analytical solution
                self.analytical_soln(time=0)

            def stepper(self):
                '''Computes the grids
                Parameters:
                    None
                '''
                # steps the function
                self.u[1:-1] = -self.u_nm1[1:-1] + 2*self.u_n[1:-1] + self.C2*(self.u_n[2:]-2*self.u_n[
        1:-1]+self.u_n[:-2])# equation (*)
                self.u_nm1[:], self.u_n[:] = self.u_n, self.u # move forward in time

            def step(self, plot=False):
                '''steps the grid and could plot
                Parameters:
                    plot: boolian (if true then plots)
                '''
                if plot:
                    plt.figure(figsize=(15, 5))
                    plt.plot(self.x, self.u_nm1,'k--', label='first step', linewidth=2)## plot the Firs
        t step
                    plt.title('Propagation of Wave to time $T$'); plt.xlabel('Position (m)'); plt.ylabe
        l('Wave Displacement (m)')
                    plt.grid(linestyle='--')
                    for times in range(1, self.Nt):
                        self.stepper()
                        if times%10 == 0 or times==self.Nt-1:
                            plt.plot(self.x, self.u, label='time='+str('%.2f'%(times*0.01*self.T)))## P
        lot Eevery tenth time step
                    plt.xlim(0,self.L)
                    plt.legend(title='Step',loc='upper right')
                    plt.show()
                else:
                    for times in range(1, self.Nt):
                        self.stepper()
                self.analytical_soln(time=self.T)
```

Using the parameters:

$$L = 12_m \quad m = 3 \quad c = 1_{m/s} \quad A = 1$$
$$\lambda = \frac{2L}{m} = \frac{2(12)}{3} = 8_m = P$$

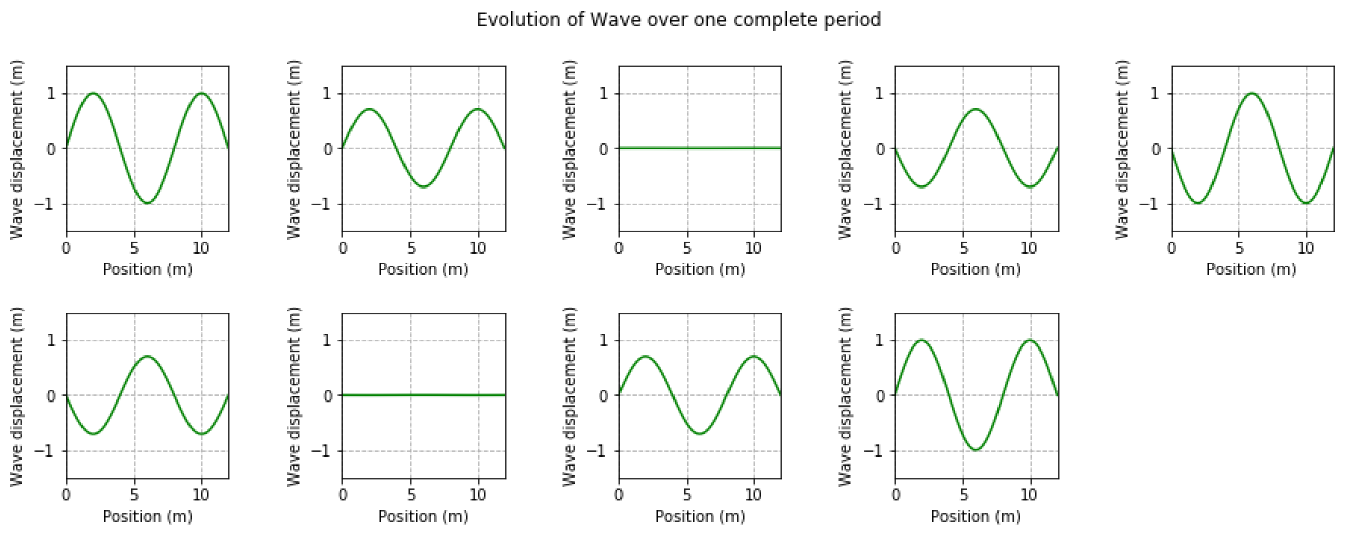
The wave length is the period only works as $c = 1$ as found in part 1. We can display a quarter period of $(8/4 = 2)$, so T$= 2$. These number make sure that the Courant number is $C < 1$ so the numeric solution does not go up in flames.

```
In [3]: WAVE = Wave(L=12, T=2, A=1, m=3, c=1)
        WAVE.step(plot=True)
```



To better see the evolution of the wave for each time step, a series of plots are used.

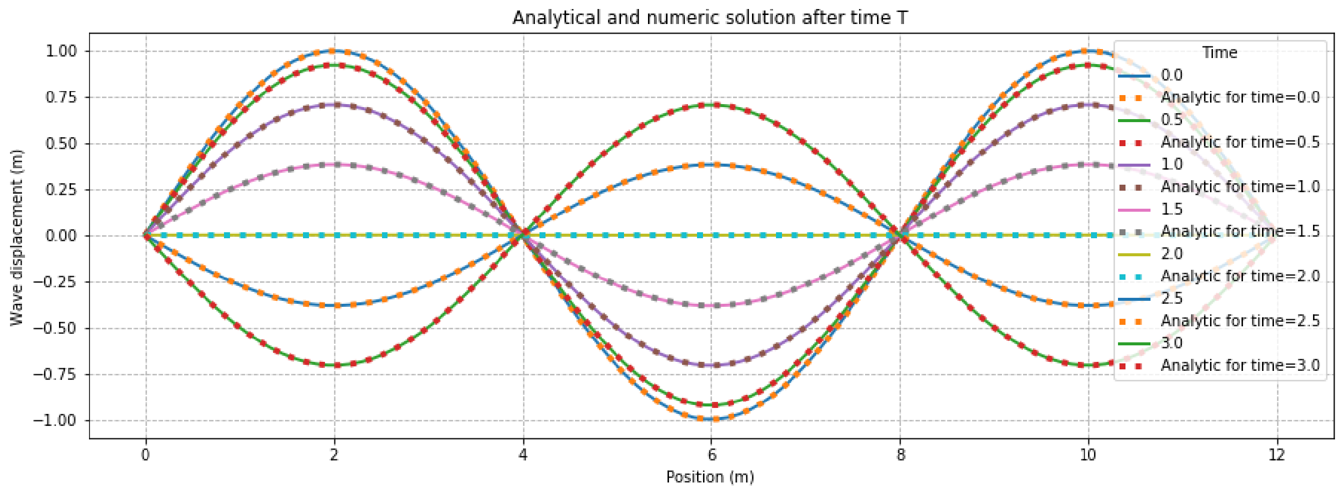
```
In [4]: plt.figure(figsize=(15,5))
        plt.suptitle("Evolution of Wave over one complete period")

        for time in range(9):
            ax = plt.subplot(2, 5, time+1)
            ax.set_ylim(-1.5,1.5); ax.set_xlim(0,12)
            WAVE = Wave(L=12, T=time, A=1, m=3, c=1) # (re)instantiate wave
            WAVE.step() # step over 1/8th of a period
            plt.plot(WAVE.x, WAVE.u,'g-') # plot wave
            plt.xlabel('Position (m)'); plt.ylabel('Wave displacement (m)'); plt.grid(linestyle='--')
        plt.subplots_adjust(wspace=0.7, hspace=0.5)
```



Look at the difference between the analytical and the numeric solutions

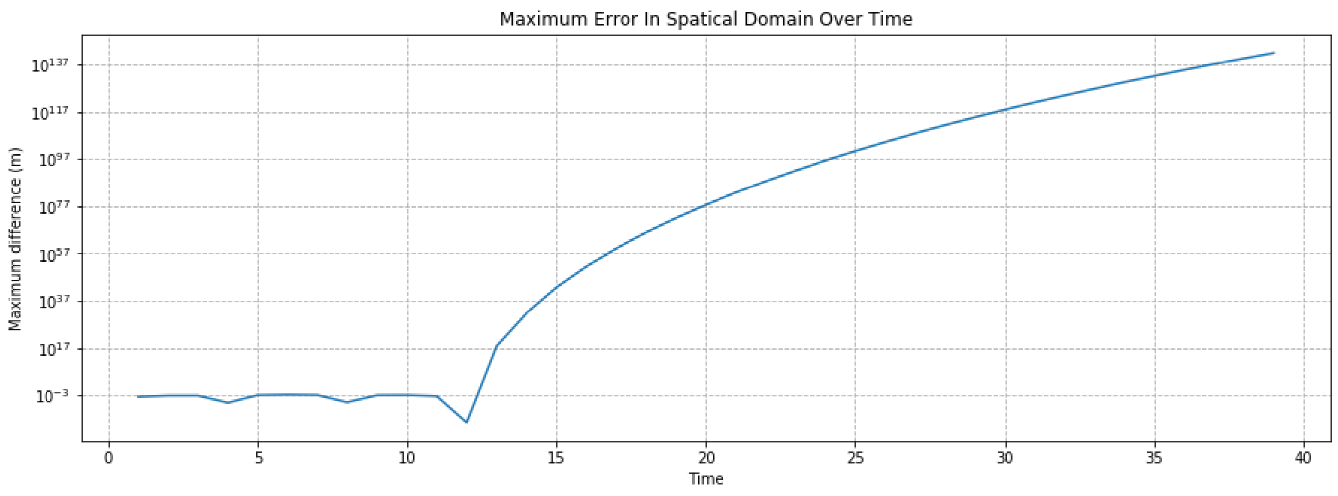
Now we check to make sure that the proposed solution at least roughly agrees with the analytical solution.

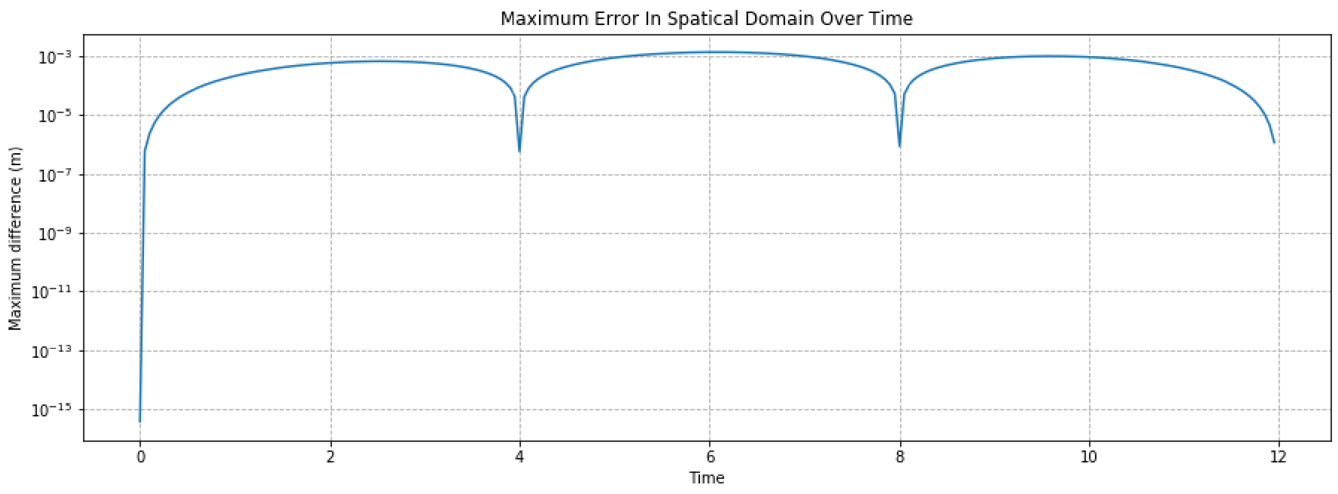
```
In [5]: plt.figure(figsize=(15,5))

        for time in np.arange(0,3.5,0.5):
            WAVE = WAVE = Wave(L=12, T=time, A=1, m=3, c=1)
            WAVE.step()
            plt.plot(WAVE.x, WAVE.u, linewidth=2, linestyle='-', label=str(time))
            plt.plot(WAVE.x, WAVE.analytic, linewidth=4, linestyle=':', label='Analytic for time='+str(
        time))
        plt.title('Analytical and numeric solution after time T'); plt.xlabel('Position (m)'); plt.ylab
        el('Wave displacement (m)')
        plt.legend(title='Time', loc='upper right'); plt.grid(linestyle='--')
```



to quantify this agreement of the analytical and numerical solutions, we can look at the point of maximum difference between the analytical solution and the numerical solution.

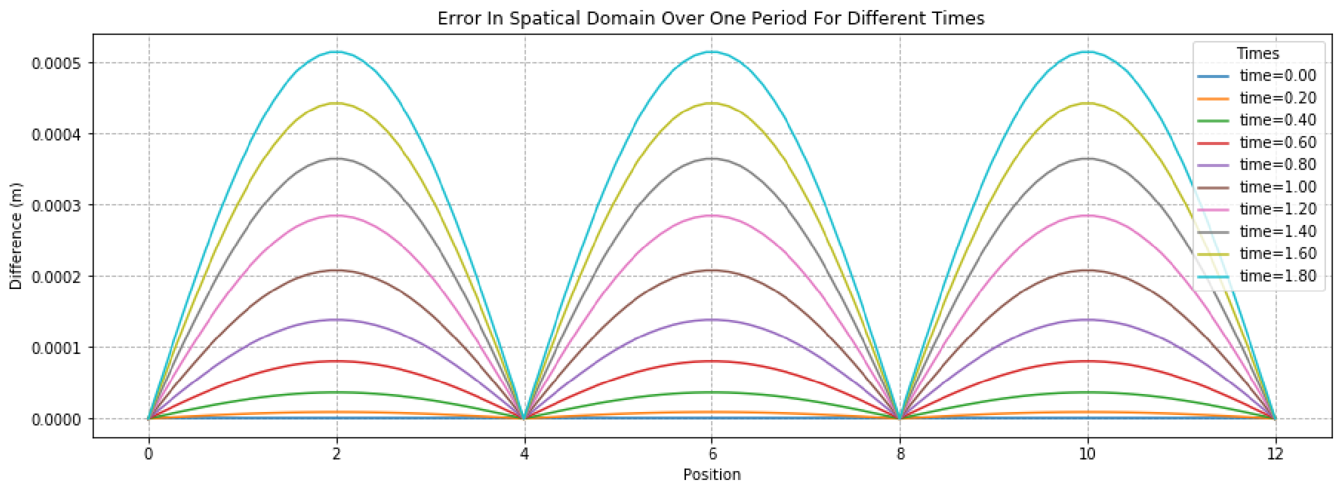
```
In [6]: def errors_T(T_values):
            '''Plots the maximum difference error in the spatial domain for different amount of time.
            Parameters:
                T_values: times to plot the difference between the analytical soln and numeric soln
            '''
            plt.figure(figsize=(15, 5))
            plt.title("Maximum Error In Spatical Domain Over Time")

            errors = []
            for time in T_values:
                WAVE = Wave(L=12, T=time, A=1, m=3, c=1)
                WAVE.step()
                errors.append(max(np.abs(WAVE.u-WAVE.analytic)))
            plt.semilogy(T_values, errors)
            plt.grid(ls='--')
            plt.xlabel('Time'); plt.ylabel('Maximum difference (m)')
            plt.show()

        errors_T(np.arange(1,40,1))
        errors_T(np.arange(0,12,0.05))
```

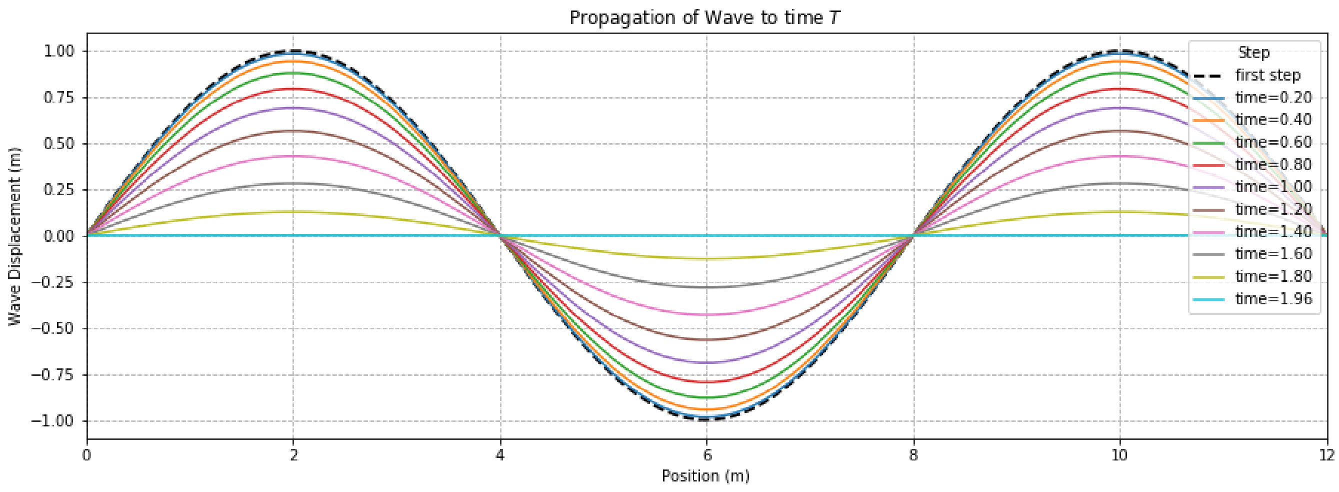




Look at the difference error between different positions

```
In [7]: def errors_x():
            '''Plots the difference error in the spatial domain over one period and plots the numerical
        solution
            Parameters:
                None
            '''
            plt.figure(figsize=(15, 5))
            plt.title("Error In Spatical Domain Over One Period For Different Times")
            for t in np.arange(0,2,0.2):
                WAVE = Wave(L=12, T=t, A=1, m=3, c=1)
                WAVE.step()
                plt.plot(WAVE.x, np.abs(WAVE.u-WAVE.analytic),label='time='+str('%.2f'%t))
            plt.grid(ls='--')
            plt.xlabel('Position'); plt.ylabel('Difference (m)'); plt.legend(title='Times')
            plt.show()

            WAVE = Wave(L=12, T=2, A=1, m=3, c=1)
            WAVE.step(plot=True)
            plt.show()

        errors_x()
```





This difference error looks to be same trend as if the wave was moving over time and seems to oscillate with proportional to the wave displacement.

Conclusion

Using a centered difference approximation to the one dimensional wave equation was seen to provide sufficiently accurate results to the analytical solution given the proper parameters. The parameters, both physical and numerical, had to be choosen with great care as not to make the Courant number be greater than 1. In a practicle or more general setting, this may not be possible and thus is not a prefect implimentation of central differences.

The behaviour of the model was important to begin with and thus the parameters for looking at the trends in the error must be taken into account when solving the system. Keeping in this in mind, the error had a very intresting trend. When the numerical solution passed the intial condtions, the error droped suddenly. This may be due to the initial conditions of the system. After the courant number reaches a point greater than 1, the solution is seen to drastically diverge from the the analytical solution.