5/28/2024

Project 3 Report

MATH 453

Cameron Robinson

Table of Contents

Table of Figures

Project 2

Part 1

The objective of part one was to implement a centered finite difference method to solve the 1D wave equation with c = sqrt(2) from t = 0 seconds to t = 8 seconds. Figure 1 shows the boundary conditions and initial conditions of the PDE.

$$\textit{Wave Equation}:$$

$$u_{tt} = c^2 u_{xx}$$

$$u(x,0) = e^{-4x^2}$$

$$u_t(x,0) = 0$$

$$u(-4,t) = u(4,t) = 0$$

$$0 \le t \le 8$$

Figure 1: Wave Equation PDE for Part 1

Figure 2 shows the exact solution of the PDE using d'Alembert's solution formula for a 1D wave. The values from Figure 1 can be directly substituted into the equation. Since there is no initial velocity, the integral portion of the solution can be ignored.

$$d'Alembert's\ Solution:$$

$$u(x,t) = \frac{1}{2}\big(f(x+ct) + f(x-ct)\big) + \frac{1}{2c}\int_{x-ct}^{x+ct} G(s)ds$$

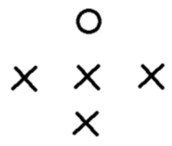
$$f(x) = u(x,t) - Initial\ Condition\ for\ Position\ of\ Wave$$

$$G(x) = u_t(x,0) - Initial\ Condition\ for\ Velocity\ of\ Wave$$

$$c - Physical\ Wave\ Speed$$

Figure 2: d'Alembert's Solution to 2D Wave Equation

Figure 3 illustrates the stencil and update formula for this method. This method always has four known values and one unknown value, which means that each point can be solved individually. This is due to there being two initial conditions, u(x,0) and $u_t(x,0)$ (velocity), which can be used to solve for the position of the wave at the first time-step. One limitation of this method is that the time steps are limited by the 1D CFL condition, which says that the computational speed ($\Delta x/\Delta t$) must be greater than or equal to the physical speed (c). The code implementing this method can be seen in Figure 35 in the appendix.



$$u(x,t+k) = 2u(x,t) - u(x,t-k) + \frac{k^2c^2}{h^2}(u(x+h,t) - 2u(x,t) + u(x-h,t))$$

Figure 3: 1D Wave Equation Finite Difference Stencil and Update Equation

Figure 4 and Figure 5 are graphs of the solutions calculated using centered finite differences with 50 and 100 x-steps respectively. Each time the wave encounters one of the boundaries, the direction of the wave's velocity and its amplitude flip. Figure 6 is a graph of the exact solution found using d'Alembert's equation. Based on the solution graphs, the numerical approximations appear to be good estimates of the actual solution.

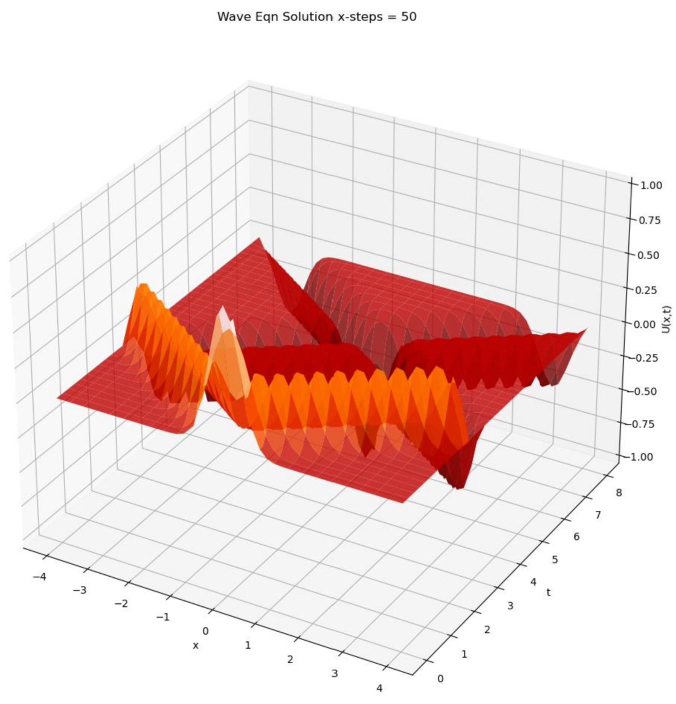


Figure 4: Part 1 Solution Graph with 50 Steps

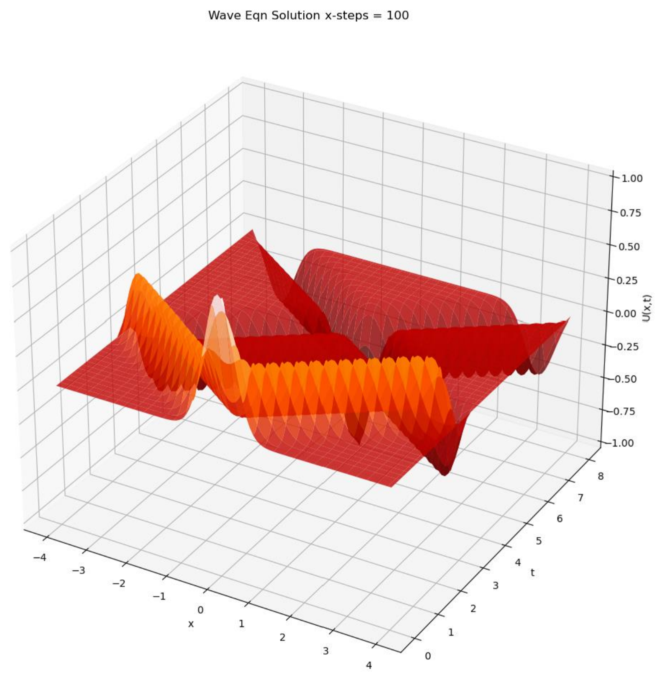


Figure 5: Part 1 Solution Graph with 100 Steps

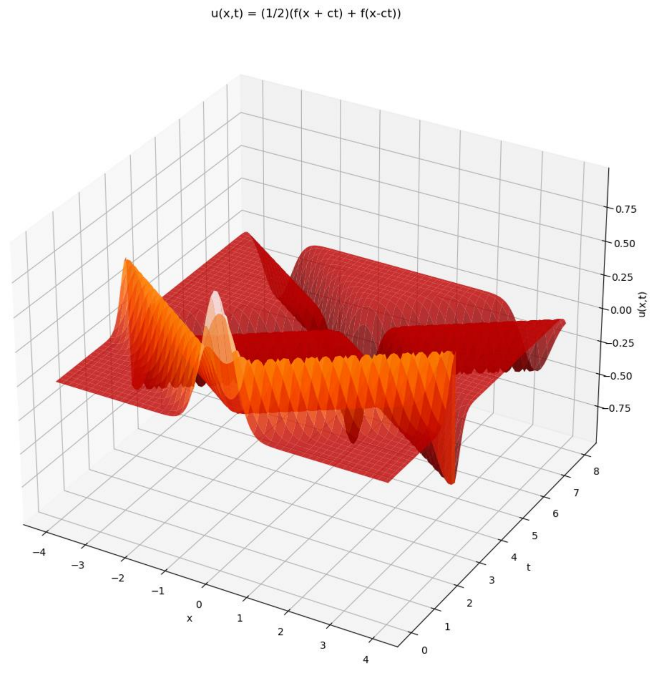


Figure 6: Part 1 Exact Solution Graph

Figure 7 shows sequential 2D snapshots of the numerical solution at different times. After the initial disturbance, two waves begin to move both left and right on the medium. The two waves are identical and have about half the amplitude of the original disturbance. Like in the 3D graphs, the waves' amplitude and velocity flip when they reflect off the fixed boundaries.

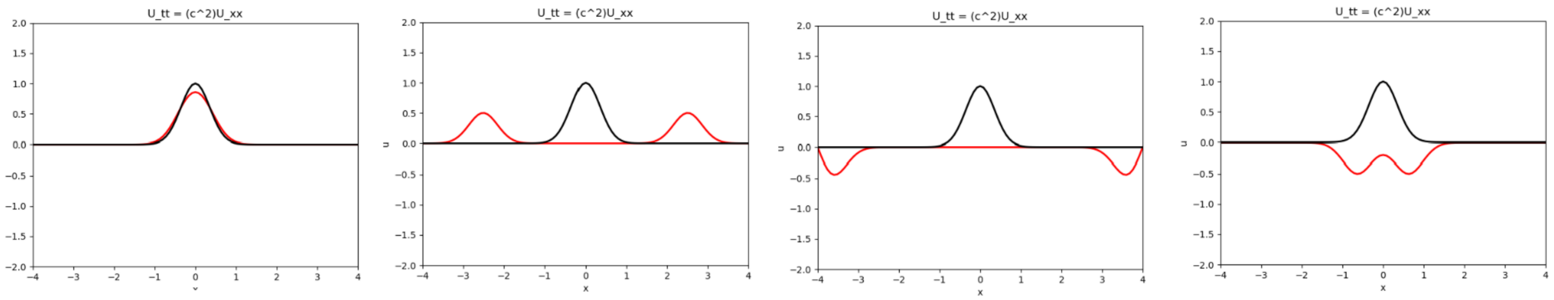


Figure 7: Part 1 Snapshots

Figure 8 and Figure 9 both contain error graphs comparing the 50 and 100-step numerical solutions with the exact solution. Since d'Alembert's solution does not have a finite domain, the exact solution does not naturally reflect off the boundaries like the numerical solutions do. The reflections in the exact solution were added by calculating where the wave should hit the boundary and then flipping the values. This is why the graphs in Figure 8 have large spikes of error; the velocity and amplitude flips performed on the exact solution do not perfectly match the numerical solution. Figure 9 on the other hand, compares the solutions before any boundaries are hit, so there is much smaller error. Regardless, the numerical approximations do converge towards the solution as the number of steps increase. Based on the max errors shown in Figure 9, when the number of steps double, the numerical solution becomes around two-times more accurate, which seems reasonable considering the method has $O((\Delta t)^2) + O((\Delta x)^2)$ error. The code to create all the graphs in this part is shown in Figure 36.

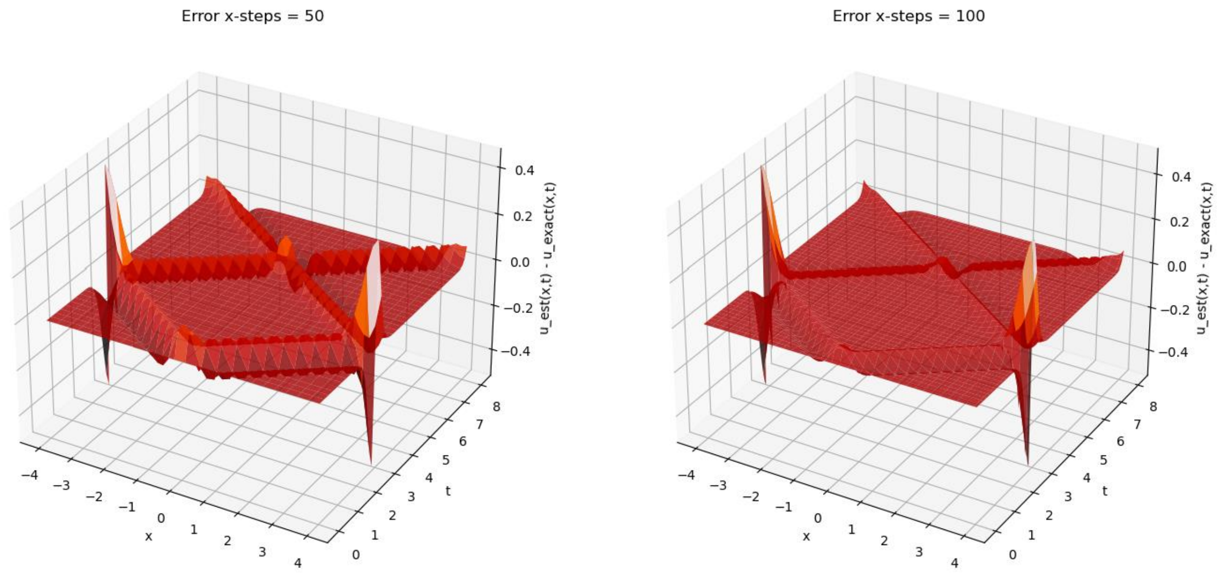


Figure 8: Part 1 Error Graphs

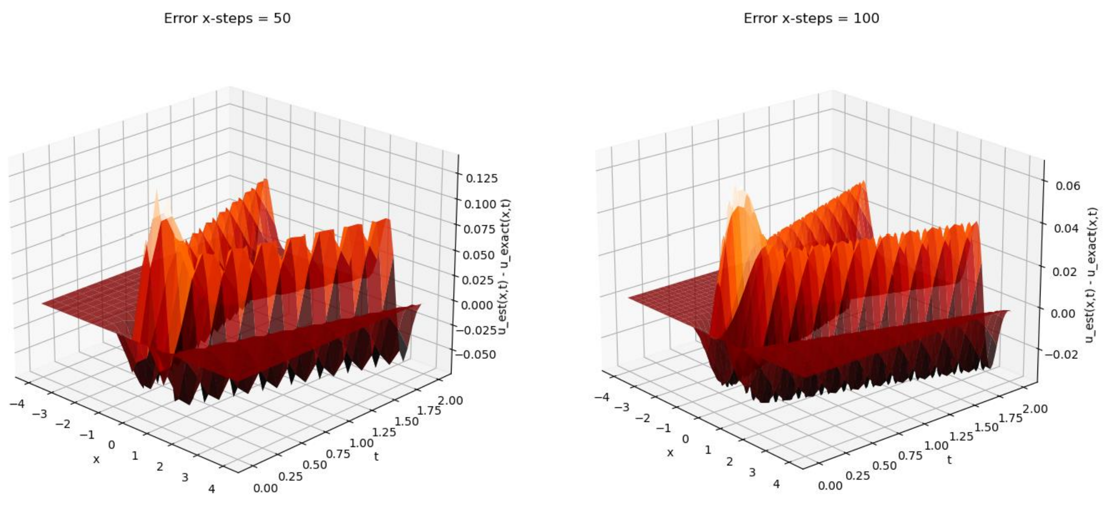


Figure 9: Part 1 Error Graphs w/o Reflections

Part 2

The objective of part two was the same as part one, only the initial conditions for the wave were different. The wave speed remained the same and the time window was still from 0s to 8s. The PDE solved in this part is shown in Figure 10. Figure 37 shows the code for the PDE and method implementation.

$Wave\ Equation$:

$u_{tt} = c^2 u_{xx}$

$u(x,0) = -x^2 + 1\ for\ |x| \leq 1, otherwise\ u(x,0) = 0$

$u_t(x,0) = 0$

$u(-4,t) = u(4,t) = 0$

$0 \leq t \leq 8$

Figure 10: PDE for Part 2

Another goal of this problem was to define numerical dispersion and diffusion. Numerical dispersion and diffusion are errors in numerical solutions caused by discretizing continuous functions. Although not a problem in some cases, the errors can become noticeable when trying to discretize functions with sharp corners. Dispersion can appear as oscillations in the numerical solution that do not appear in the actual solution, and diffusion tends to cause these oscillations to spread out as the numerical errors are used in further calculations. Both diffusion and dispersion are applicable to this problem because the wave is a parabola and has sharp corners at both edges of the wave.

Figure 11 and Figure 12 are graphs of the numerical solutions with 50 and 100 x-steps, while Figure 13 is a graph of the exact solution. The exact solution was found using d'Alembert's solution. Overall, the results are very similar to the first part, only the shape of the wave is slightly different. Neither dispersion nor diffusion are very obvious by looking at the graphs of the numerical solutions.

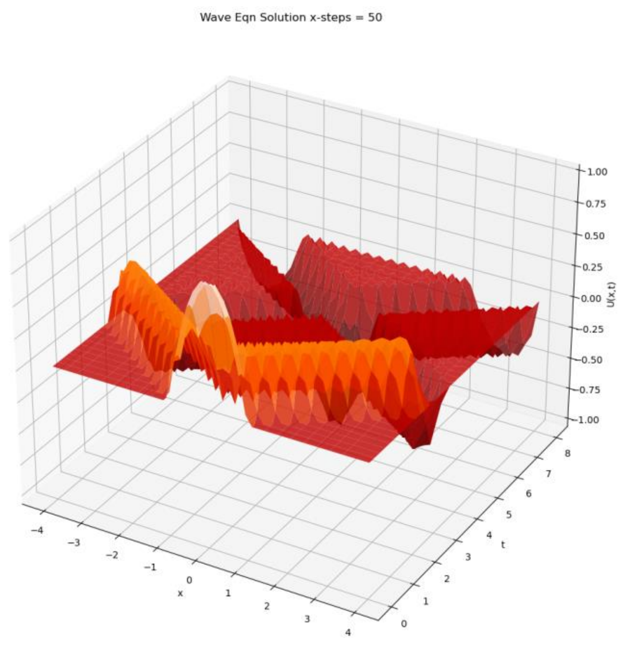


Figure 11: Part 2 Solution Graph with 50 Steps

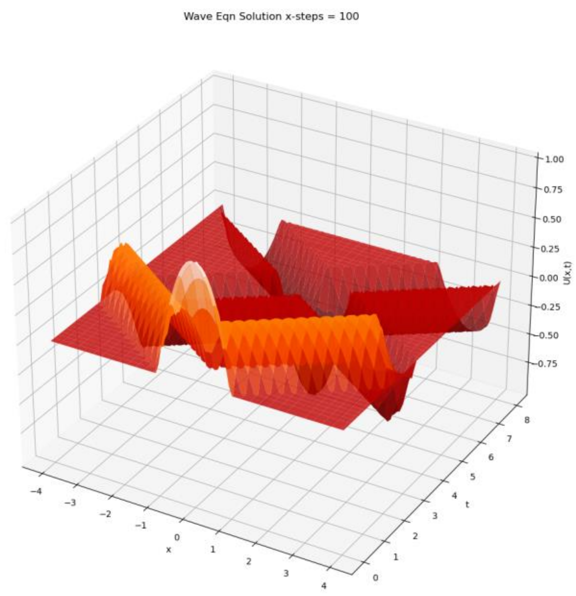


Figure 12: Part 2 Solution Graphs with 100 Steps

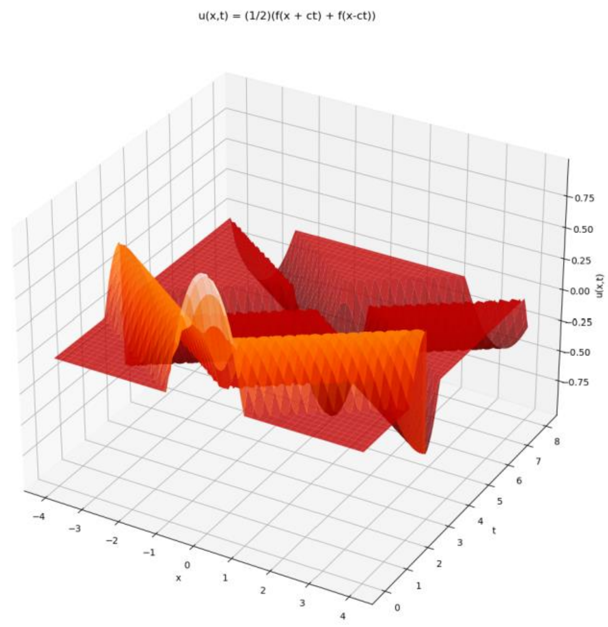


Figure 13: Part 2 Exact Solution Graph

Figure 14 shows a series of 2D snapshots of the numerical solution. Like in part one, the initial disturbance splits into two waves travelling in opposite directions with half the amplitude of the original wave. The velocity and amplitude also flip when it encounters a boundary since the BCs are still fixed.

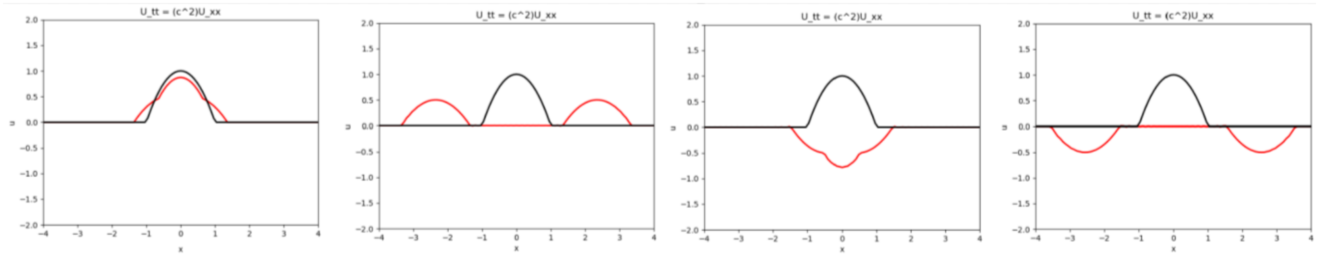


Figure 14: Part 2 2D Snapshots

Figure 15 shows an example of numerical dispersion and diffusion. The top image shows small oscillations in the numerical solution that do not appear in the exact solution; these oscillations are a result of numerical dispersion. The bottom image shows that the oscillations tend to spread out as more time-steps are calculated, which is the effect of numerical diffusion.

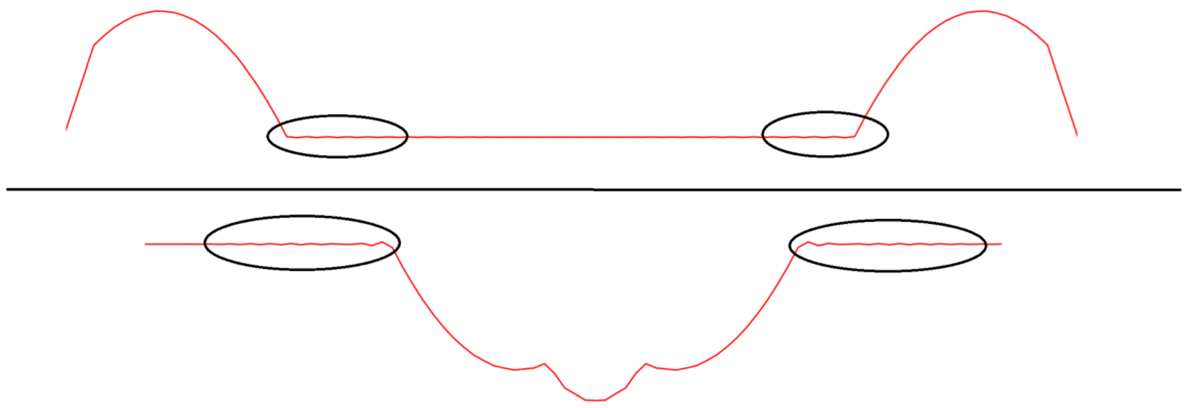


Figure 15: Numerical Dispersion and Diffusion

Figure 16 and Figure 17 are error graphs showing the differences between the 50 and 100-step numerical solutions and the exact solution. The error graphs that include the wave reflections also have large error spikes like in part one. The error graphs without the reflections do a better job of illustrating the convergence of the centered finite difference method. When the number of steps double, the error once again decreases by about half. Also visible in the error graphs are the oscillations caused by numerical dispersion, which are shown by the larger errors at the ends of each wave (the corners where the wave meets the undisturbed medium). The code to create all the graphs in this part is shown in Figure 38.

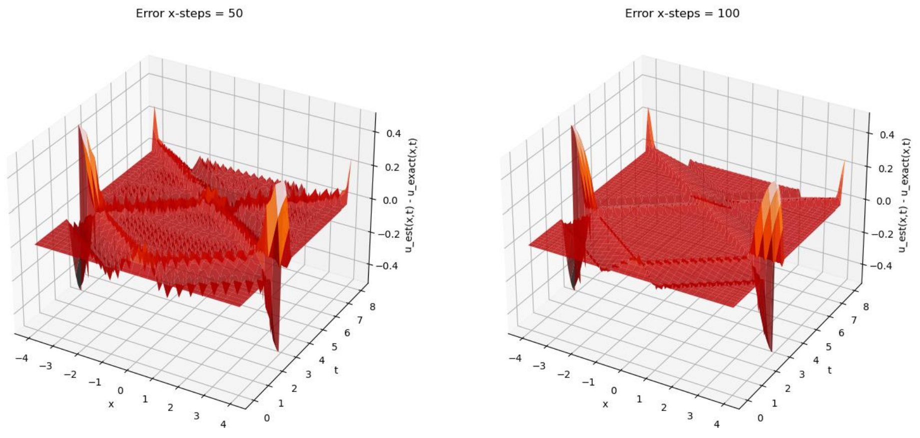


Figure 16: Part 2 Error Graphs

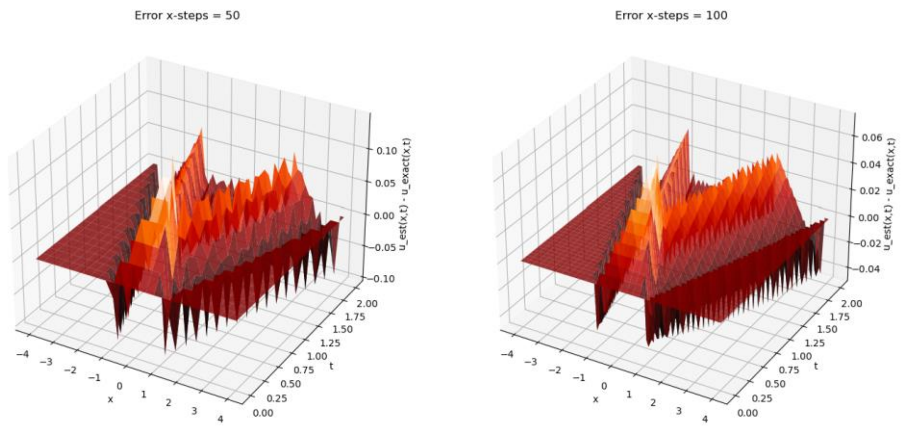


Figure 17: Part 2 Error Graphs w/o Reflections

Part 3

The goal of part three was the same as parts one and two, the only difference was that there are different initial conditions, and the time window is from 0s to 20s. The wave speed is still sqrt(2), however, the initial velocity is no longer zero. Additionally, the wave has sharp corners at the top and left side, which should result in numerical diffusion and dispersion in the numerical solutions. The specific PDE solved in this problem is shown in Figure 18. The code for the method implementation is shown in Figure 39 in the appendix.

$Wave\ Equation$:

$u_{tt} = c^2 u_{xx}$

$u(x,0) = \frac{x}{2}\ for\ 0 \leq x \leq 2$

$u(x,0) = (x-3)^2\ \ for\ 2 < x \leq 3$

$u_t(x,0) = -\frac{1}{2}\ for\ 0 \leq x \leq 2$

$u_t(x,0) = -2x + 6\ for\ 2 < x \leq 3$

$u(0,t) = u(10,t) = 0$

$0 \leq t \leq 20$

Figure 18: Part 3 PDE

Figure 19 and Figure 20 are graphs of the numerical solutions with 50 and 100 x-steps. Figure 21 is a graph of the exact solution, once again found using d'Alembert's solution for the 1D wave equation. It is important to note that the exact solution was not reflected, so the graphs can only be compared up to about 7.5 seconds when the waves reach the first boundary. Overall, the numerical solutions turned out to be fairly accurate, though some numerical dispersion is visible on both graphs. There are small ripples and a large undershoot on the left side of the waves. The size of the error due to dispersion seems to decrease as the number of steps increase since it is less obvious on the 100-step graph.

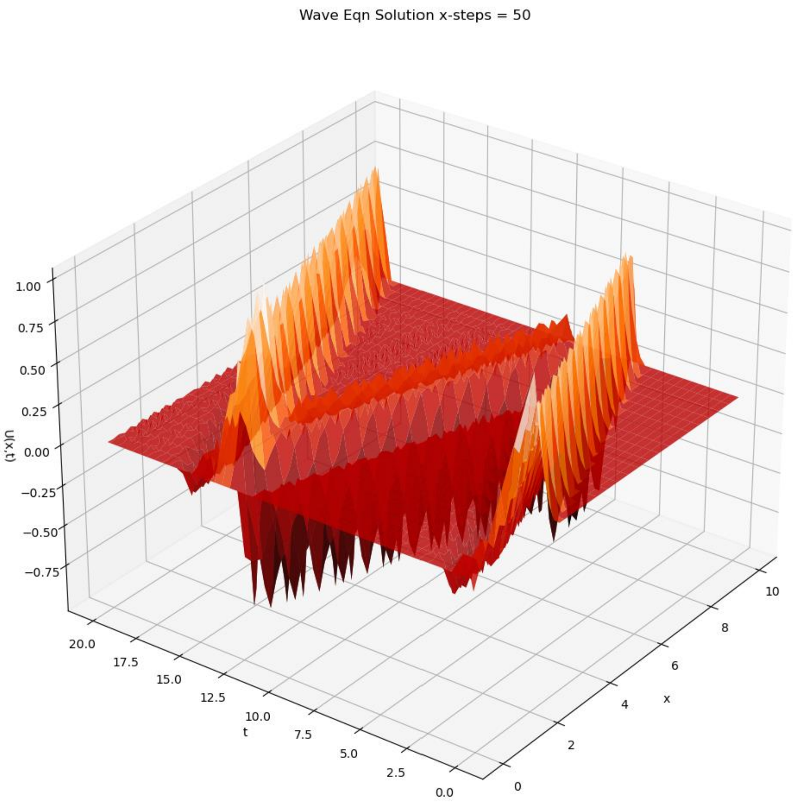


Figure 19: Part 3 Solution Graph with 50 Steps

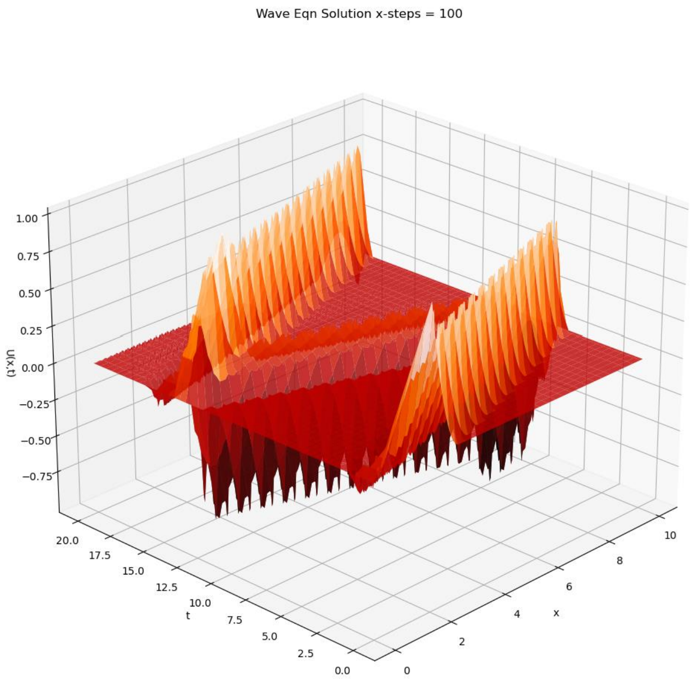


Figure 20: Part 3 Solution Graph with 100 Steps

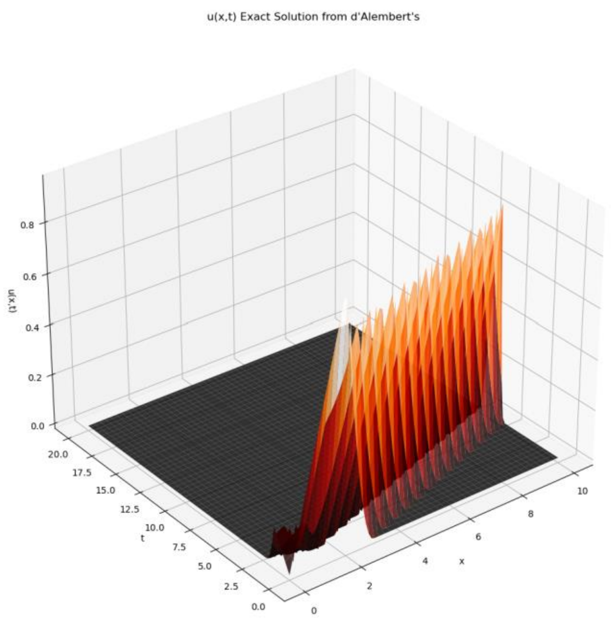


Figure 21: Part 3 Exact Solution

Figure 22 shows 2D snapshots of the numerical solution. Unlike before, the wave has an initial velocity which caused it to travel as a single wave rather than as two waves moving in opposite directions. Due to the sharp corners at the top of the wave and at the trailing edge, there is a lot of error due to dispersion. The large undershoot and small oscillations on the left edge match what is seen in the 3D graphs.

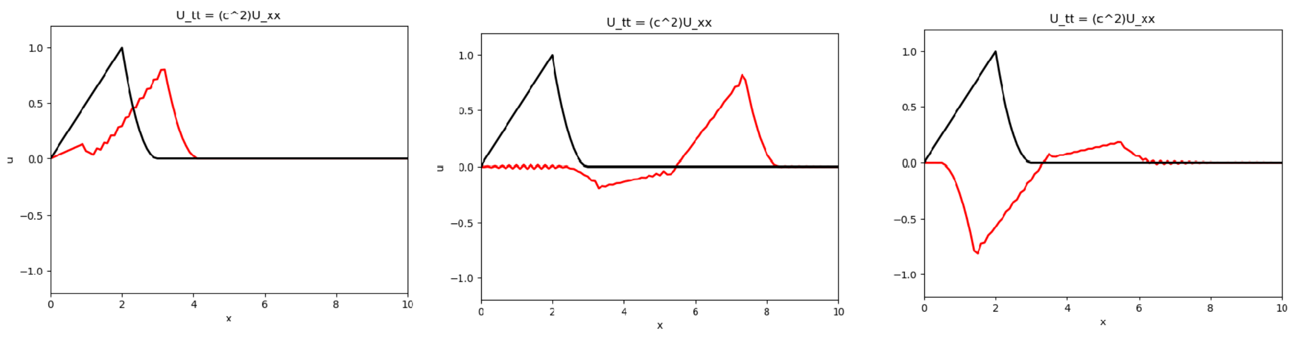


Figure 22: Part 3 2D Snapshots

Figure 23 shows the error graphs between the numerical solutions and the exact solution. In both the 50 and 100-step graphs, the numerical solutions tended to undershoot the exact solution in nearly every spot. As before, the numerical solution does converge towards the exact solution as the number of steps increases. With twice as many steps, the numerical solution becomes about twice as accurate. The code to create all of the graphs in this part is shown in Figure 40.

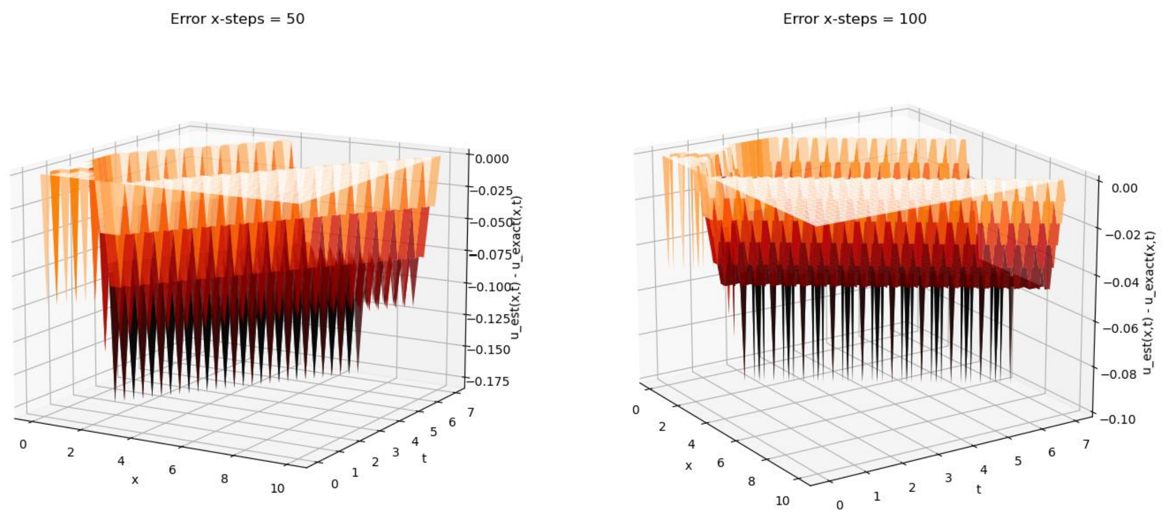


Figure 23: Part 3 Error Graphs w/o Reflections

Part 4

The objective of part four was to solve the 2D wave equation shown in Figure 24 using the centered finite difference (CFD) method. For this problem, c = 1, the boundaries were fixed at zero, and the region was π units long in x and y. The update equation and stencil are shown in Figure 25; the method is almost identical to the CFD method for the 1D wave, but it includes some extra terms for the y-axis points. Like before, there is only one unknown point, which means that each unknown can be solved individually. Like the 1D CFD method, this method must meet the 2D CFL condition which makes sure that the physics do not outrun the computation speed ($\Delta t \leq \Delta x/2c$). The code for this method is shown in Figure 41 in the appendix.

$2D\ Wave\ Equation$:

$$u_{tt} = c^2(u_{xx} + u_{yy})$$

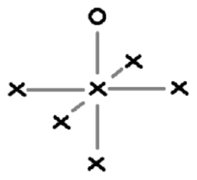
$$u(x,y,0) = \sin(2x)\sin(2y)$$

$$u_t(x,y,0) = 0$$

$$u(0,y,t) = u(\pi,y,t) = u(x,0,t) = u(x,\pi,t) = 0$$

$$0 \leq t \leq 6$$

Figure 24: 2D Wave PDE for Part 4



$$u(x,y,t+\Delta t) = 2u(x,y,t) - u(x,y,t-\Delta t) + c^2(\Delta t)^2\left(\frac{u(x+\Delta x,y,t)-2u(x,y,t)+u(x-\Delta x,y,t)}{(\Delta x)^2} + \frac{u(x,y+\Delta y,t)-2u(x,y,t)+u(x,y-\Delta y,t)}{(\Delta y)^2}\right)$$

Figure 25: 2D Wave Finite Difference Stencil and Update Equation

Figure 26 shows the exact solution to the PDE and verifies that it is correct by checking the BCs, IVs, and by substituting the derivatives directly into the PDE.

$Solution$

$u(x,y,t) = \cos(2\sqrt{2}\,t)\sin(2x)\sin(2y)$

$Verification$

$\sin(2\pi) = \sin(0) = 0, \textit{so BCs are met}$

$u(x,y,0) = 1 * \sin(2x)\sin(2y)\,, \textit{so IC for position is met}$

$u_t(x,y,0) = 0\ \textit{since}\ \cos\left(2\sqrt{2}\,t\right)\ \textit{becomes sine and}\ \sin(0) = 0$

$u_{tt} = 1^2(u_{xx} + u_{yy})$

$u_{tt} = -8\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y)$

$u_{xx} = -4\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y)$

$u_{yy} = -4\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y)$

$-8\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y) = 1^2(-4\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y) + -4\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y))$

$-8\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y) = \sin(2x)\sin(2y)(-4\cos(2\sqrt{2}\,t) + -4\cos(2\sqrt{2}\,t))$

$-8\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y) = -8\cos(2\sqrt{2}\,t)\sin(2x)\sin(2y)$

Figure 26: Part 4 Solution and Verification

Figure 27 shows snapshots of the numerical solution with 50 steps in the x and y-directions. Figure 28 shows similar snapshots but for the exact solution. The numerical and exact solution look nearly identical.

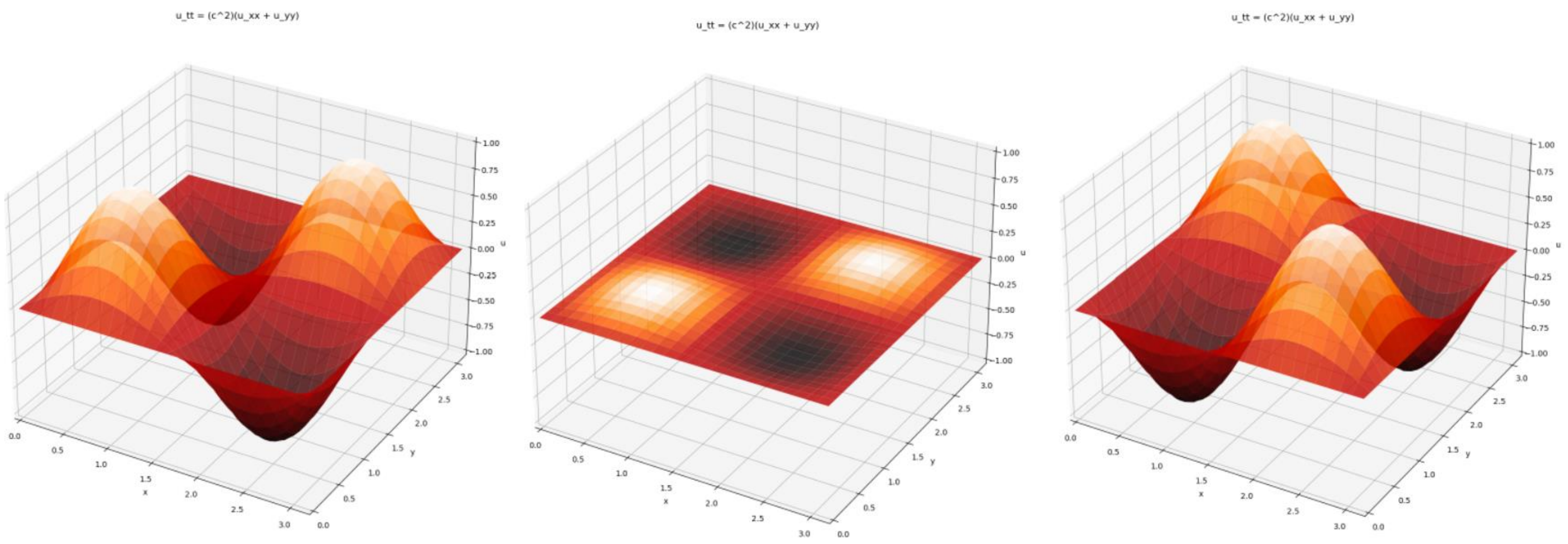


Figure 27: Part 4 Numerical Solution with 50 X and Y Steps

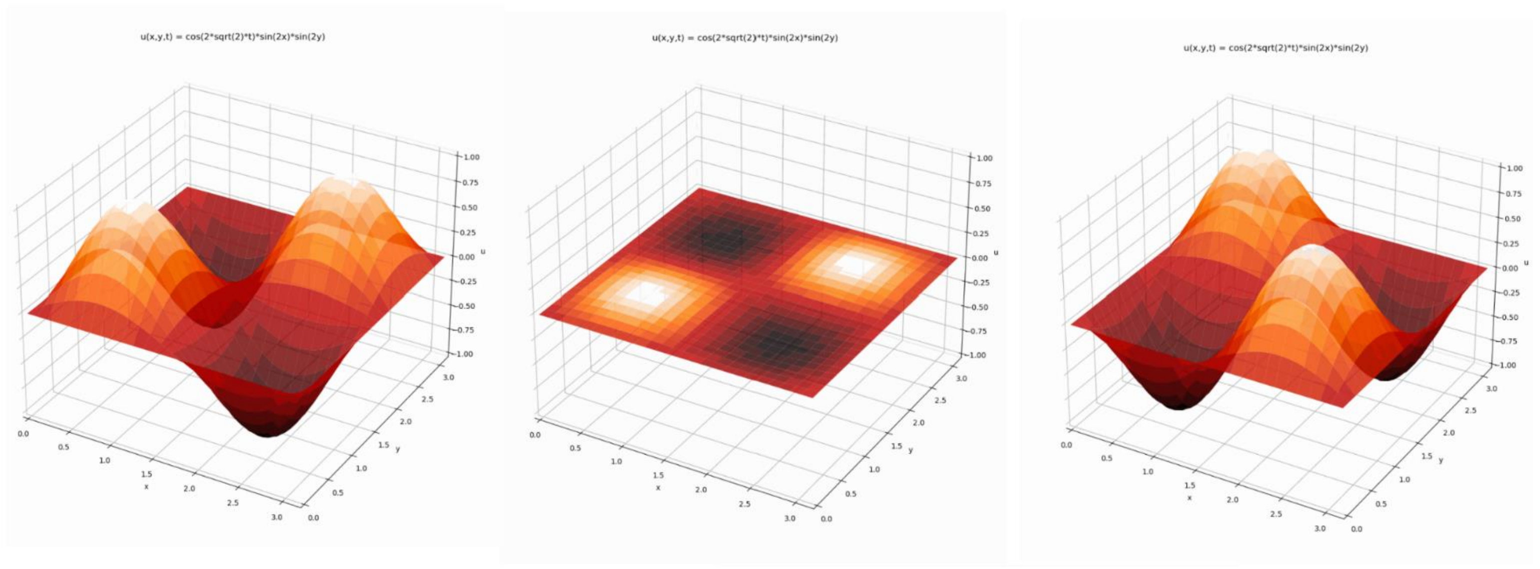


Figure 28: Part 4 Exact Solution Graphs

Figure 29 shows two error graphs, one for a 25-step numerical solution and the other for a 50-step numerical solution. The snapshots show the maximum error of each solution since the amount of error changed with time. It is interesting that the errors tended to match the solution graphs almost exactly with the only difference being the amplitude of the waves. The error of the method has $O((\Delta t)^2)$ + $O((\Delta x)^2)$ + $O((\Delta y)^2)$, which is similar to the 1D CFD method, just with an extra error term for the y-variable. The numerical solutions do converge towards the exact solution with more steps. When the number of steps doubled, the maximum error decreased by about half. The code to create all of the graphs in this part is shown in Figure 42 in the appendix.

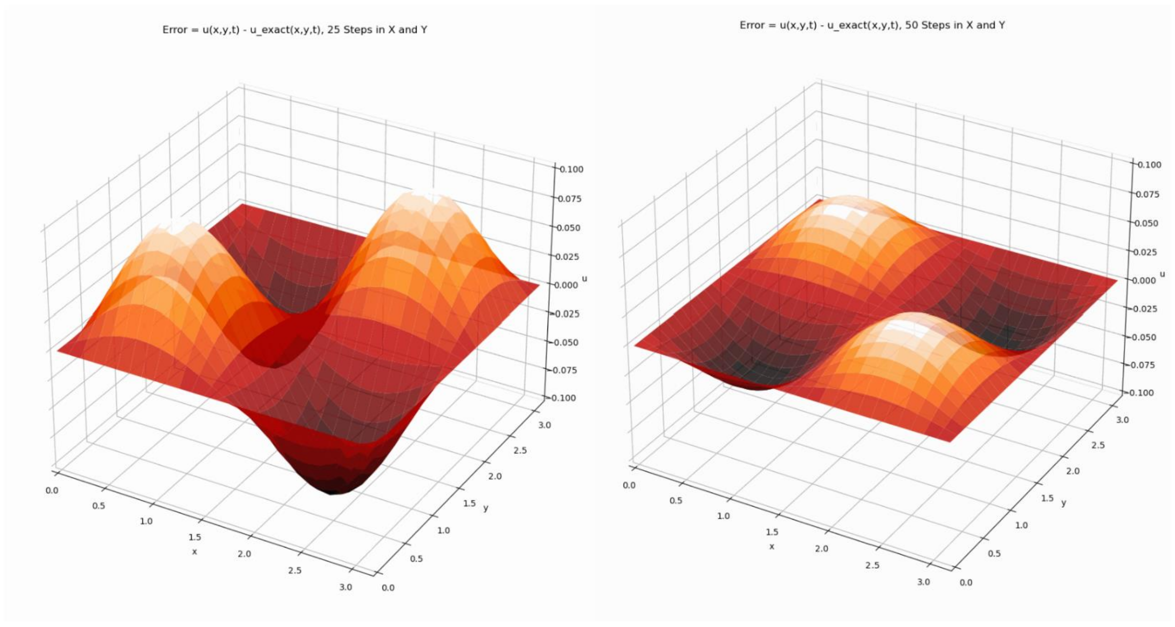


Figure 29: Part 4 Max Error Graphs

Part 5

The goal of part five was the same as part four, only with the PDE shown in Figure 30. Additionally, the wave speed, c, was two instead of one. Figure 31 shows the analytical solution to the PDE. The code for this method is shown in Figure 43.

$2D\ Wave\ Equation:$

$u_{tt} = c^2(u_{xx} + u_{yy})$

$u(x, y, 0) = \mathrm{xy}(1 - \mathrm{x})(1 - \mathrm{y})$

$u_t(x, y, 0) = 0$

$u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(x, 1, t) = 0$

$0 \leq t \leq 10$

Figure 30: Part 5 2D Wave PDE

$$u(x,y,t) = \frac{64}{\pi^6} \sum_{n=0}^{\infty} \sum_{m=0}^{\infty} \frac{1}{(2n+1)^3(2m+1)^3} *$$
$$\sin((2n+1)\pi x)\sin((2m+1)\pi y)\cos(c\pi t\sqrt{(2n+1)^2+(2m+1)^2})$$

Figure 31: Part 5 Exact Solution

Figure 32 shows snapshots of the numerical solution calculated with 15 steps in x and y. Figure 33 shows the exact solution calculated with values of n and m from 0 – 49. Although the exact solution has some error, it should be extremely small since the terms in the solution quickly converge to zero as n and m increase. Overall, the exact solution and numerical solution appear to be nearly identical.

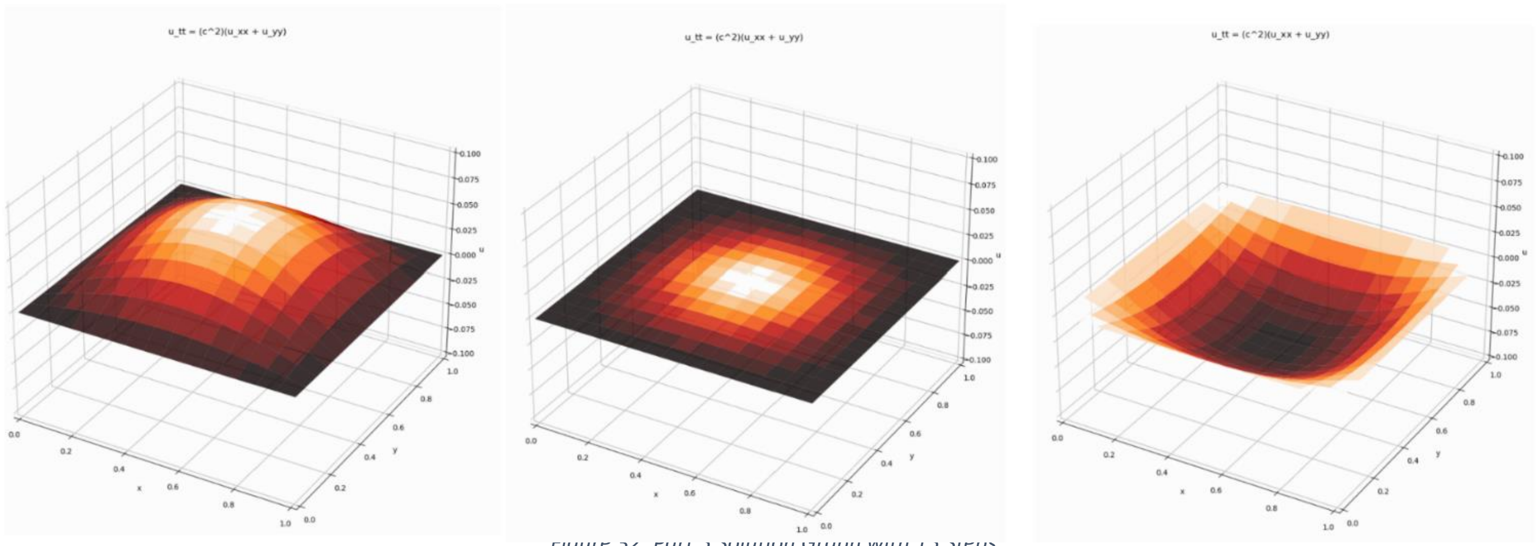


Figure 32: Part 5 Solution Graph with 15 Steps

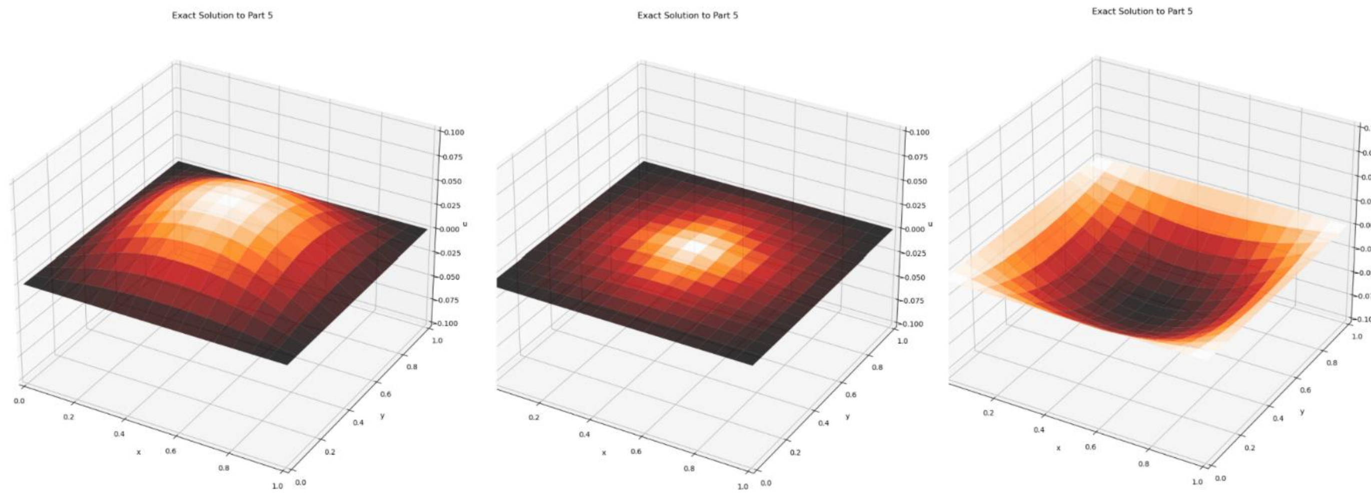
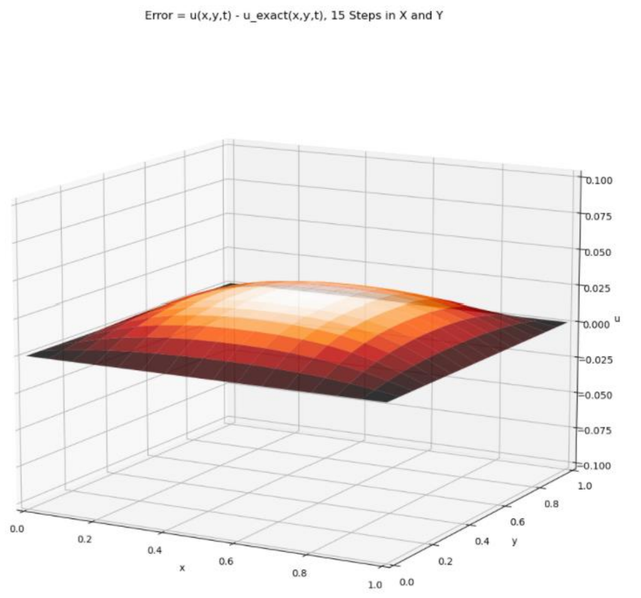


Figure 33: Part 5 Exact Solution Graphs

Figure 34 shows the error graph of the 15-step numerical solution at two different times when the error oscillations reached their peak. Unlike the other numerical solutions, the error for this one had a lot of variation with respect to time. As time increased, the amount of error also increased. As shown in the graphs, the max error was initially about 0.025, but later increased to about 0.1. This may be due to compounding errors, but I am not sure why it was much more visible in this problem than in part four. The code to create all of the graphs in this part is shown in Figure 44.



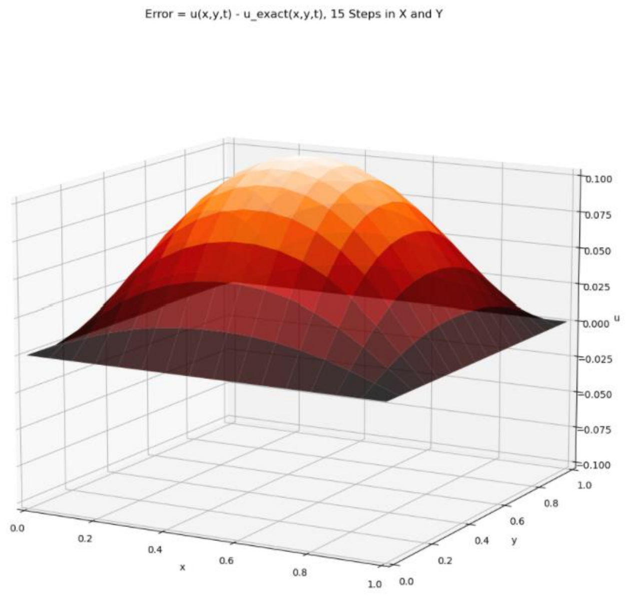
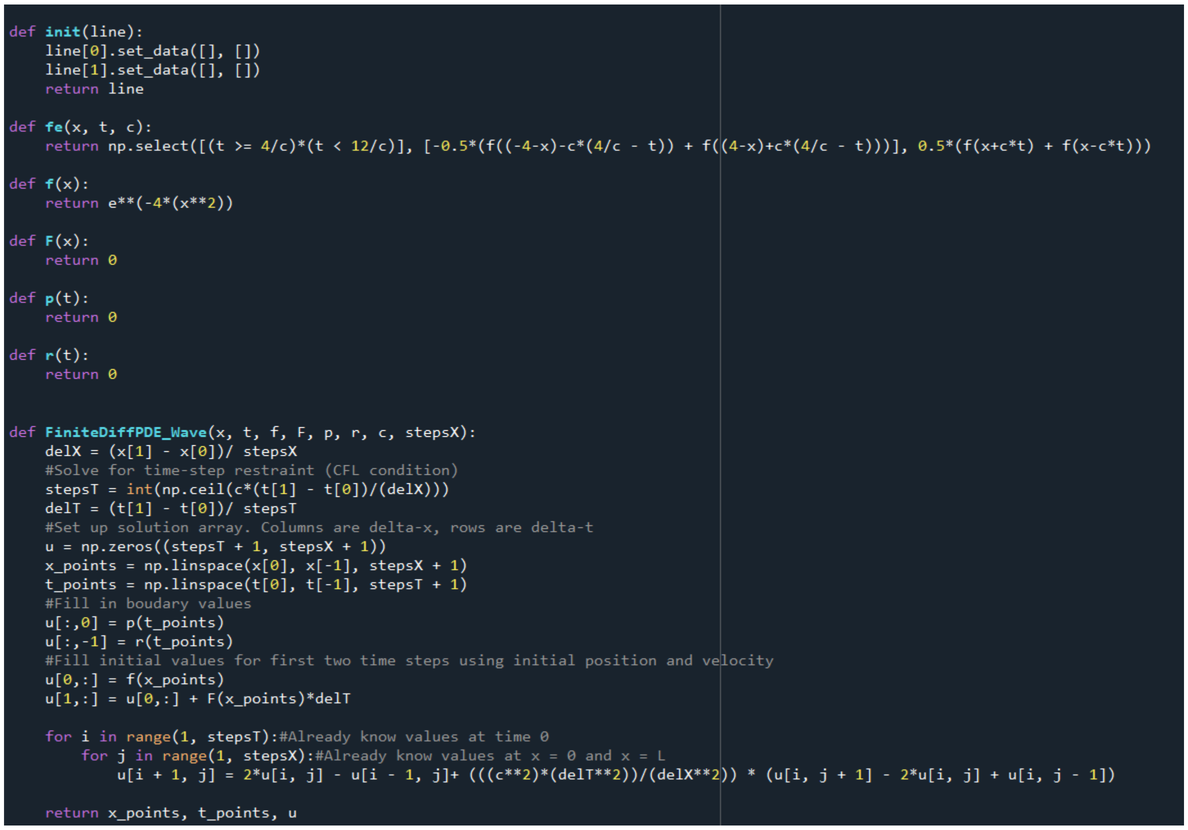


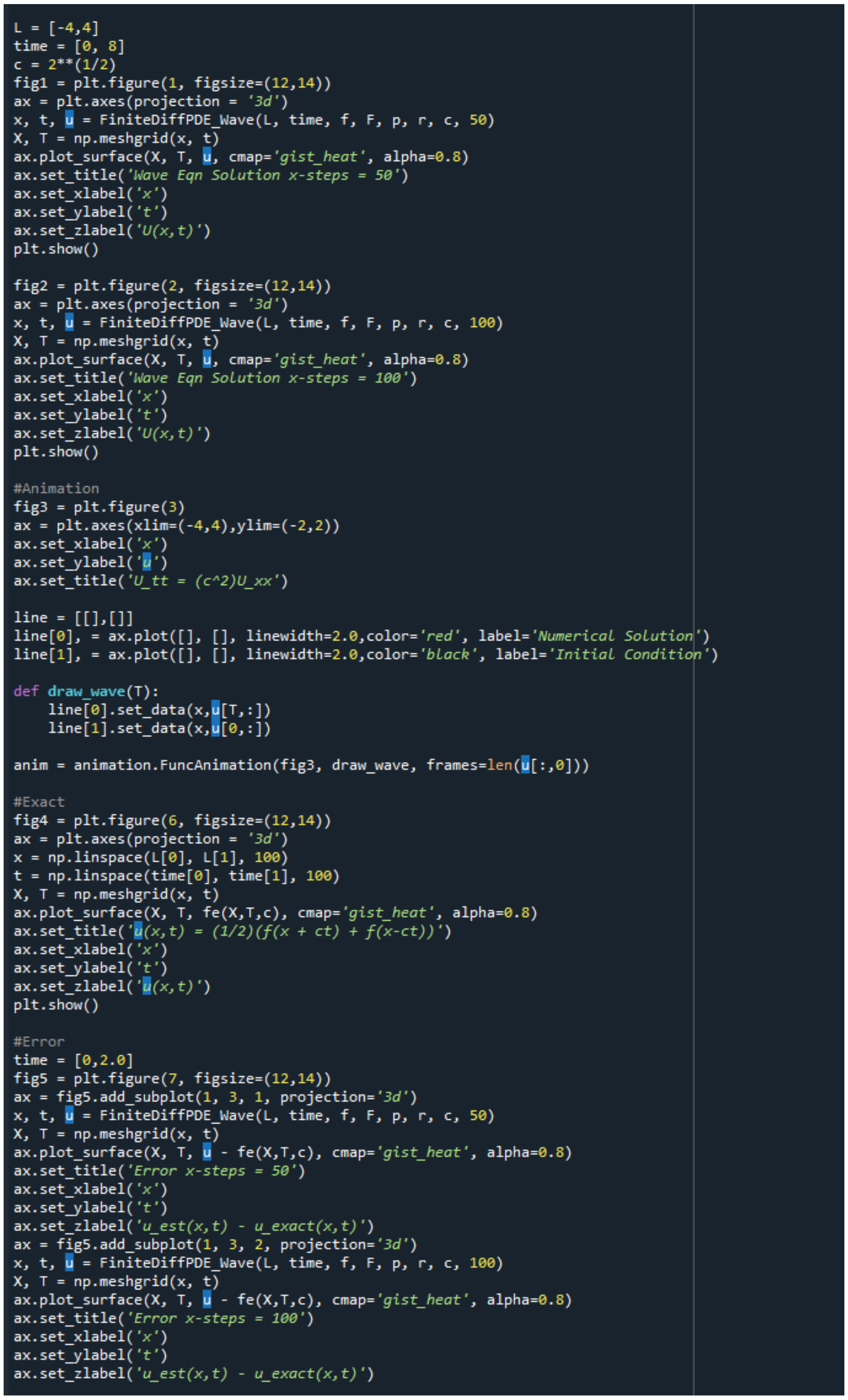
Figure 34: Part 5 Error Graphs

Appendix



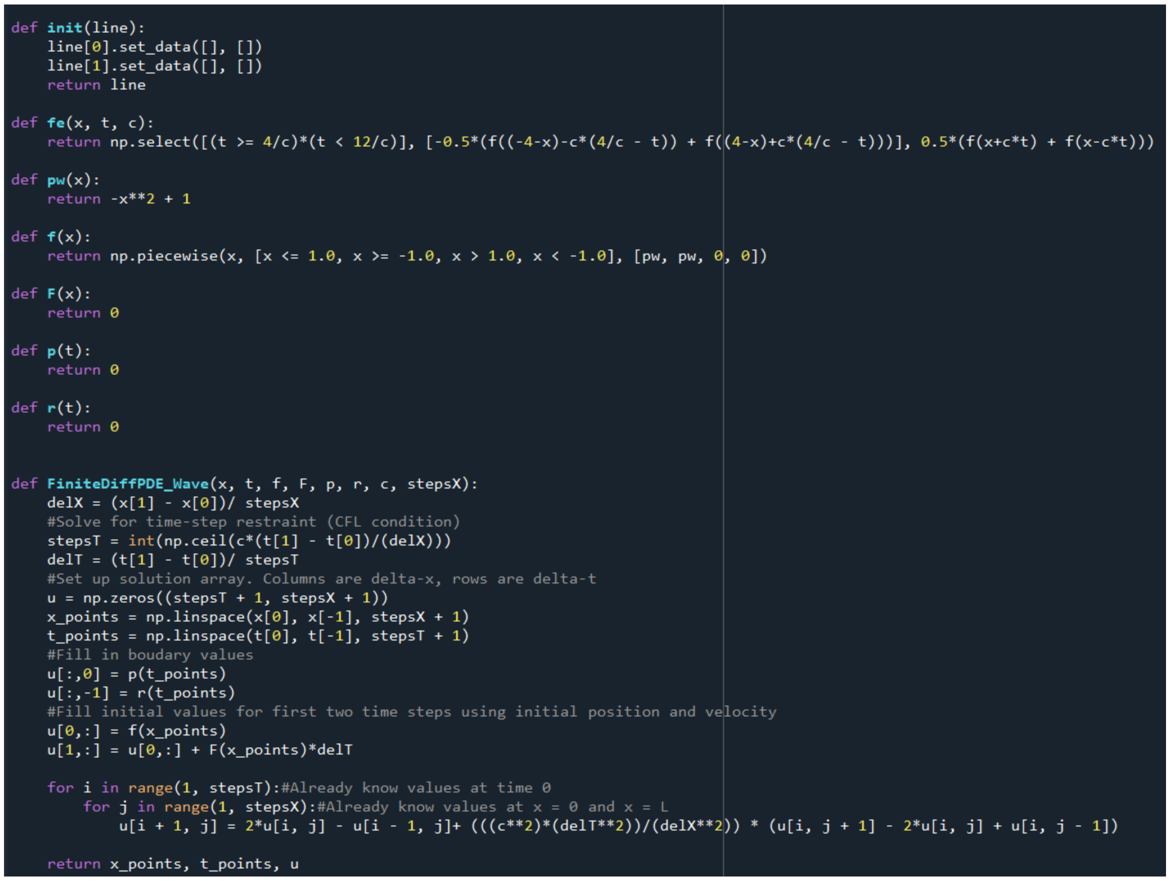
```python
def init(line):
    line[0].set_data([], [])
    line[1].set_data([], [])
    return line

def fe(x, t, c):
    return np.select([(t >= 4/c)*(t < 12/c)], [-0.5*(f((-4-x)-c*(4/c - t)) + f((4-x)+c*(4/c - t)))], 0.5*(f(x+c*t) + f(x-c*t)))

def f(x):
    return e**(-4*(x**2))

def F(x):
    return 0

def p(t):
    return 0

def r(t):
    return 0


def FiniteDiffPDE_Wave(x, t, f, F, p, r, c, stepsX):
    delX = (x[1] - x[0])/ stepsX
    #Solve for time-step restraint (CFL condition)
    stepsT = int(np.ceil(c*(t[1] - t[0])/(delX)))
    delT = (t[1] - t[0])/ stepsT
    #Set up solution array. Columns are delta-x, rows are delta-t
    u = np.zeros((stepsT + 1, stepsX + 1))
    x_points = np.linspace(x[0], x[-1], stepsX + 1)
    t_points = np.linspace(t[0], t[-1], stepsT + 1)
    #Fill in boudary values
    u[:,0] = p(t_points)
    u[:,-1] = r(t_points)
    #Fill initial values for first two time steps using initial position and velocity
    u[0,:] = f(x_points)
    u[1,:] = u[0,:] + F(x_points)*delT

    for i in range(1, stepsT):#Already know values at time 0
        for j in range(1, stepsX):#Already know values at x = 0 and x = L
            u[i + 1, j] = 2*u[i, j] - u[i - 1, j]+ (((c**2)*(delT**2))/(delX**2)) * (u[i, j + 1] - 2*u[i, j] + u[i, j - 1])

    return x_points, t_points, u
```

Figure 35: Part 1 Method Code



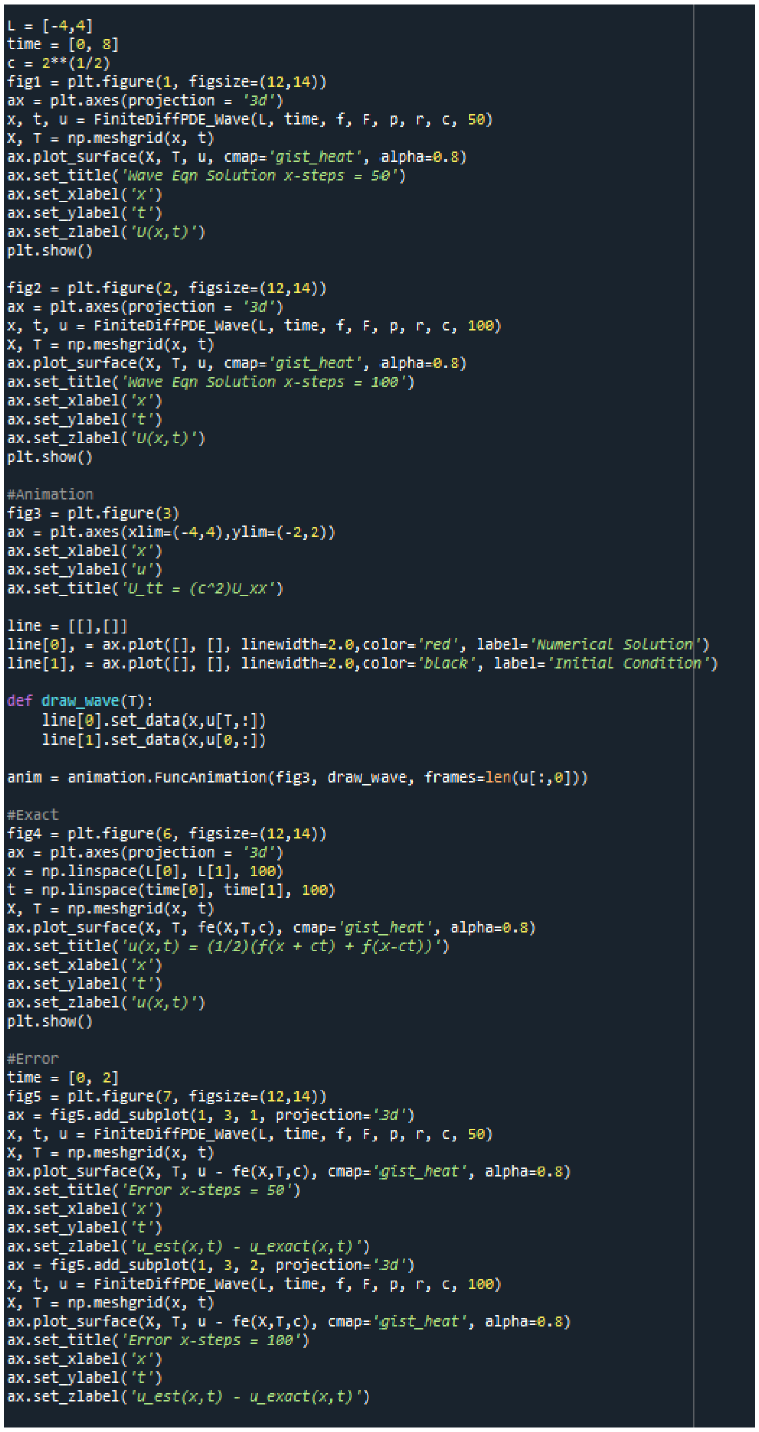
```python
L = [-4,4]
time = [0, 8]
c = 2**(1/2)
fig1 = plt.figure(1, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

fig2 = plt.figure(2, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

#Animation
fig3 = plt.figure(3)
ax = plt.axes(xlim=(-4,4),ylim=(-2,2))
ax.set_xlabel('x')
ax.set_ylabel('u')
ax.set_title('U_tt = (c^2)U_xx')

line = [[],[]]
line[0], = ax.plot([], [], linewidth=2.0,color='red', label='Numerical Solution')
line[1], = ax.plot([], [], linewidth=2.0,color='black', label='Initial Condition')

def draw_wave(T):
    line[0].set_data(x,u[T,:])
    line[1].set_data(x,u[0,:])

anim = animation.FuncAnimation(fig3, draw_wave, frames=len(u[:,0]))

#Exact
fig4 = plt.figure(6, figsize=(12,14))
ax = plt.axes(projection = '3d')
x = np.linspace(L[0], L[1], 100)
t = np.linspace(time[0], time[1], 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('u(x,t) = (1/2)(f(x + ct) + f(x-ct))')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
plt.show()

#Error
time = [0,2.0]
fig5 = plt.figure(7, figsize=(12,14))
ax = fig5.add_subplot(1, 3, 1, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
ax = fig5.add_subplot(1, 3, 2, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
```

Figure 36: Part 1 Graphing Code



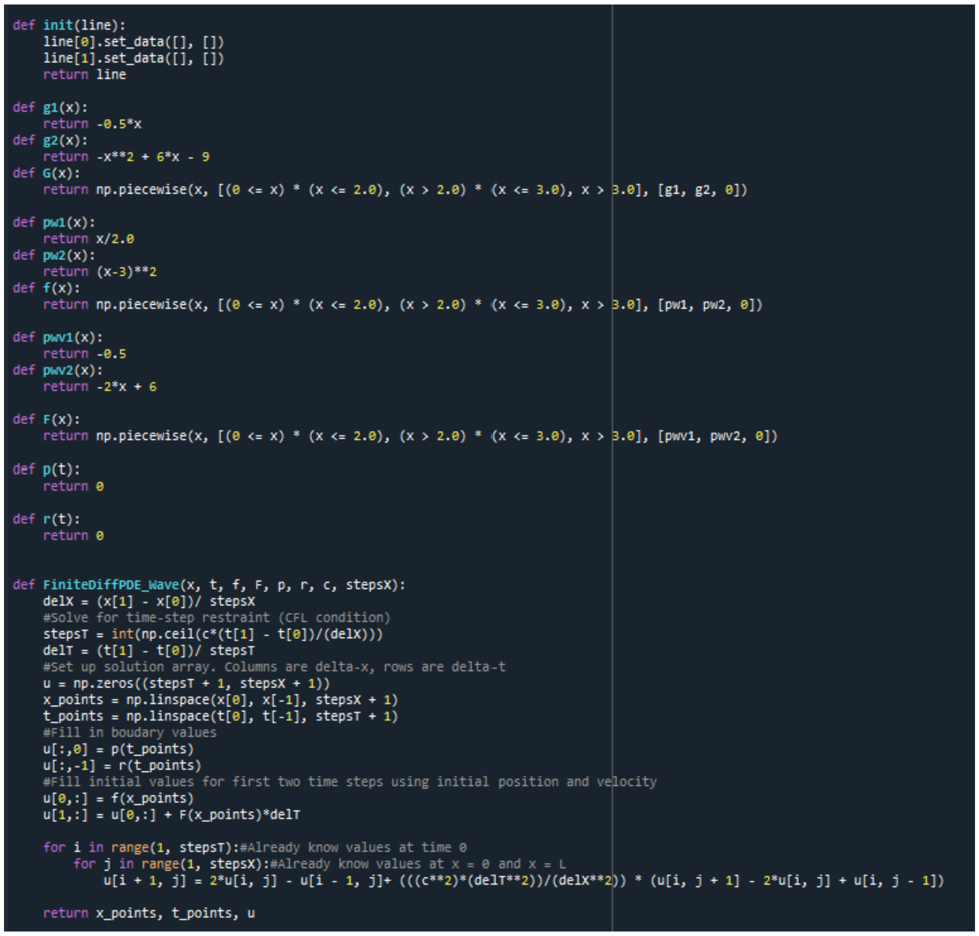
```python
def init(line):
    line[0].set_data([], [])
    line[1].set_data([], [])
    return line

def fe(x, t, c):
    return np.select([(t >= 4/c)*(t < 12/c)], [-0.5*(f((-4-x)-c*(4/c - t)) + f((4-x)+c*(4/c - t)))], 0.5*(f(x+c*t) + f(x-c*t)))

def pw(x):
    return -x**2 + 1

def f(x):
    return np.piecewise(x, [x <= 1.0, x >= -1.0, x > 1.0, x < -1.0], [pw, pw, 0, 0])

def F(x):
    return 0

def p(t):
    return 0

def r(t):
    return 0


def FiniteDiffPDE_Wave(x, t, f, F, p, r, c, stepsX):
    delX = (x[1] - x[0])/ stepsX
    #Solve for time-step restraint (CFL condition)
    stepsT = int(np.ceil(c*(t[1] - t[0])/(delX)))
    delT = (t[1] - t[0])/ stepsT
    #Set up solution array. Columns are delta-x, rows are delta-t
    u = np.zeros((stepsT + 1, stepsX + 1))
    x_points = np.linspace(x[0], x[-1], stepsX + 1)
    t_points = np.linspace(t[0], t[-1], stepsT + 1)
    #Fill in boudary values
    u[:,0] = p(t_points)
    u[:,-1] = r(t_points)
    #Fill initial values for first two time steps using initial position and velocity
    u[0,:] = f(x_points)
    u[1,:] = u[0,:] + F(x_points)*delT

    for i in range(1, stepsT):#Already know values at time 0
        for j in range(1, stepsX):#Already know values at x = 0 and x = L
            u[i + 1, j] = 2*u[i, j] - u[i - 1, j]+ (((c**2)*(delT**2))/(delX**2)) * (u[i, j + 1] - 2*u[i, j] + u[i, j - 1])

    return x_points, t_points, u
```

Figure 37: Part 2 Method Code



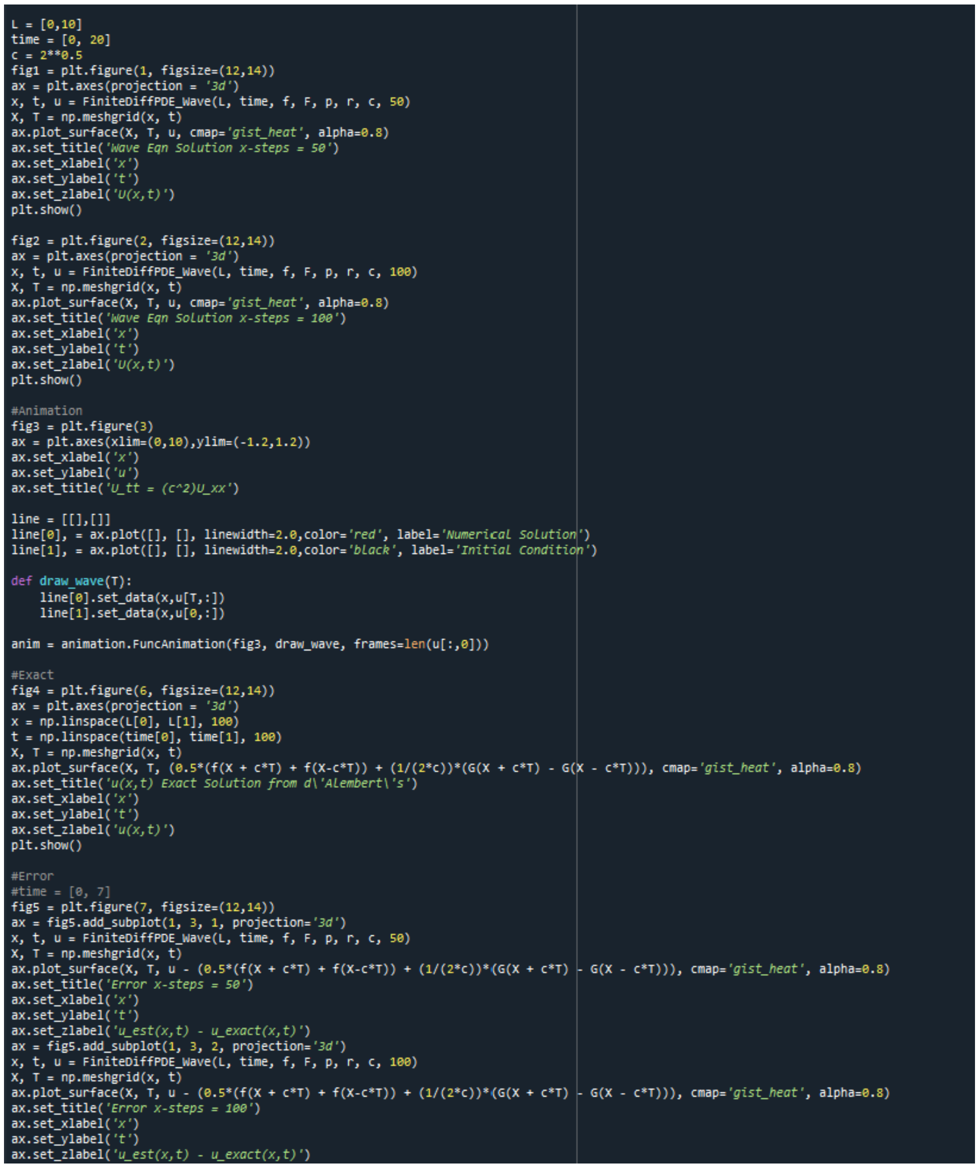
```python
L = [-4,4]
time = [0, 8]
c = 2**(1/2)
fig1 = plt.figure(1, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

fig2 = plt.figure(2, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

#Animation
fig3 = plt.figure(3)
ax = plt.axes(xlim=(-4,4),ylim=(-2,2))
ax.set_xlabel('x')
ax.set_ylabel('u')
ax.set_title('U_tt = (c^2)U_xx')

line = [[],[]]
line[0], = ax.plot([], [], linewidth=2.0,color='red', label='Numerical Solution')
line[1], = ax.plot([], [], linewidth=2.0,color='black', label='Initial Condition')

def draw_wave(T):
    line[0].set_data(x,u[T,:])
    line[1].set_data(x,u[0,:])

anim = animation.FuncAnimation(fig3, draw_wave, frames=len(u[:,0]))

#Exact
fig4 = plt.figure(6, figsize=(12,14))
ax = plt.axes(projection = '3d')
x = np.linspace(L[0], L[1], 100)
t = np.linspace(time[0], time[1], 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('u(x,t) = (1/2)(f(x + ct) + f(x-ct))')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
plt.show()

#Error
time = [0, 2]
fig5 = plt.figure(7, figsize=(12,14))
ax = fig5.add_subplot(1, 3, 1, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
ax = fig5.add_subplot(1, 3, 2, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - fe(X,T,c), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
```

Figure 38: Part 2 Graphing Code



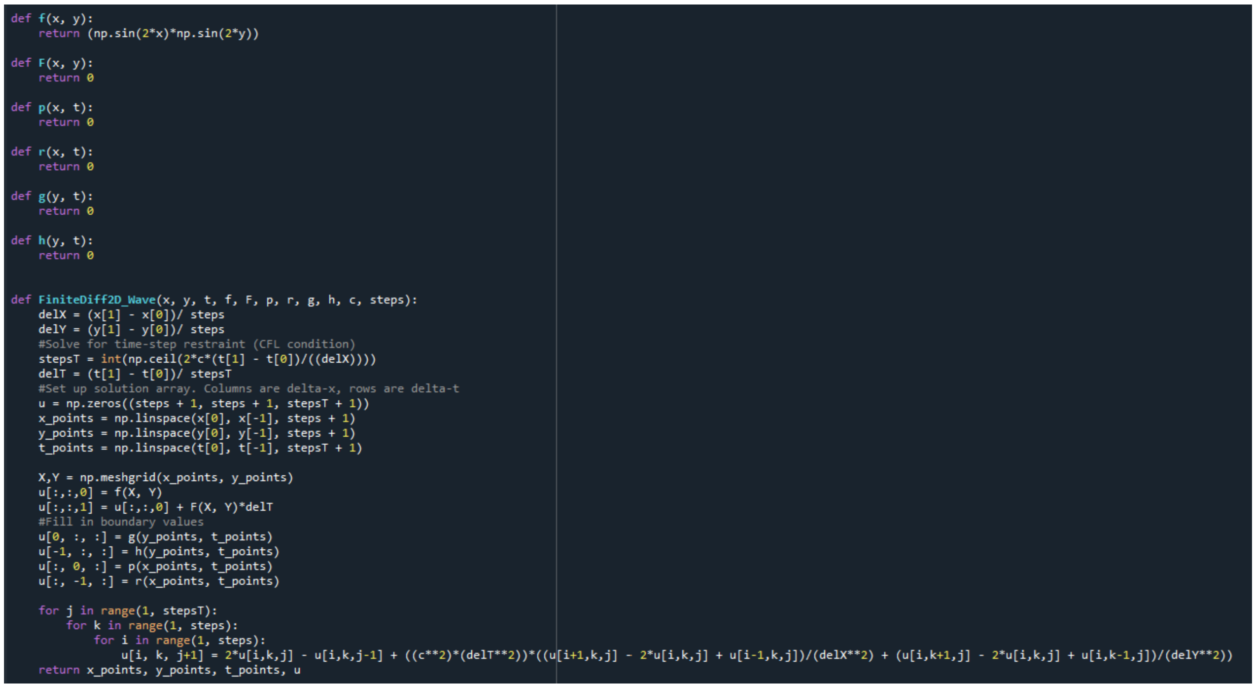
```python
def init(line):
    line[0].set_data([], [])
    line[1].set_data([], [])
    return line

def g1(x):
    return -0.5*x
def g2(x):
    return -x**2 + 6*x - 9
def G(x):
    return np.piecewise(x, [(0 <= x) * (x <= 2.0), (x > 2.0) * (x <= 3.0), x > 3.0], [g1, g2, 0])

def pw1(x):
    return x/2.0
def pw2(x):
    return (x-3)**2
def f(x):
    return np.piecewise(x, [(0 <= x) * (x <= 2.0), (x > 2.0) * (x <= 3.0), x > 3.0], [pw1, pw2, 0])

def pwv1(x):
    return -0.5
def pwv2(x):
    return -2*x + 6

def F(x):
    return np.piecewise(x, [(0 <= x) * (x <= 2.0), (x > 2.0) * (x <= 3.0), x > 3.0], [pwv1, pwv2, 0])

def p(t):
    return 0

def r(t):
    return 0


def FiniteDiffPDE_Wave(x, t, f, F, p, r, c, stepsX):
    delX = (x[1] - x[0])/ stepsX
    #Solve for time-step restraint (CFL condition)
    stepsT = int(np.ceil(c*(t[1] - t[0])/(delX)))
    delT = (t[1] - t[0])/ stepsT
    #Set up solution array. Columns are delta-x, rows are delta-t
    u = np.zeros((stepsT + 1, stepsX + 1))
    x_points = np.linspace(x[0], x[-1], stepsX + 1)
    t_points = np.linspace(t[0], t[-1], stepsT + 1)
    #Fill in boudary values
    u[:,0] = p(t_points)
    u[:,-1] = r(t_points)
    #Fill initial values for first two time steps using initial position and velocity
    u[0,:] = f(x_points)
    u[1,:] = u[0,:] + F(x_points)*delT

    for i in range(1, stepsT):#Already know values at time 0
        for j in range(1, stepsX):#Already know values at x = 0 and x = L
            u[i + 1, j] = 2*u[i, j] - u[i - 1, j]+ (((c**2)*(delT**2))/(delX**2)) * (u[i, j + 1] - 2*u[i, j] + u[i, j - 1])

    return x_points, t_points, u
```

Figure 39: Part 3 Method Code



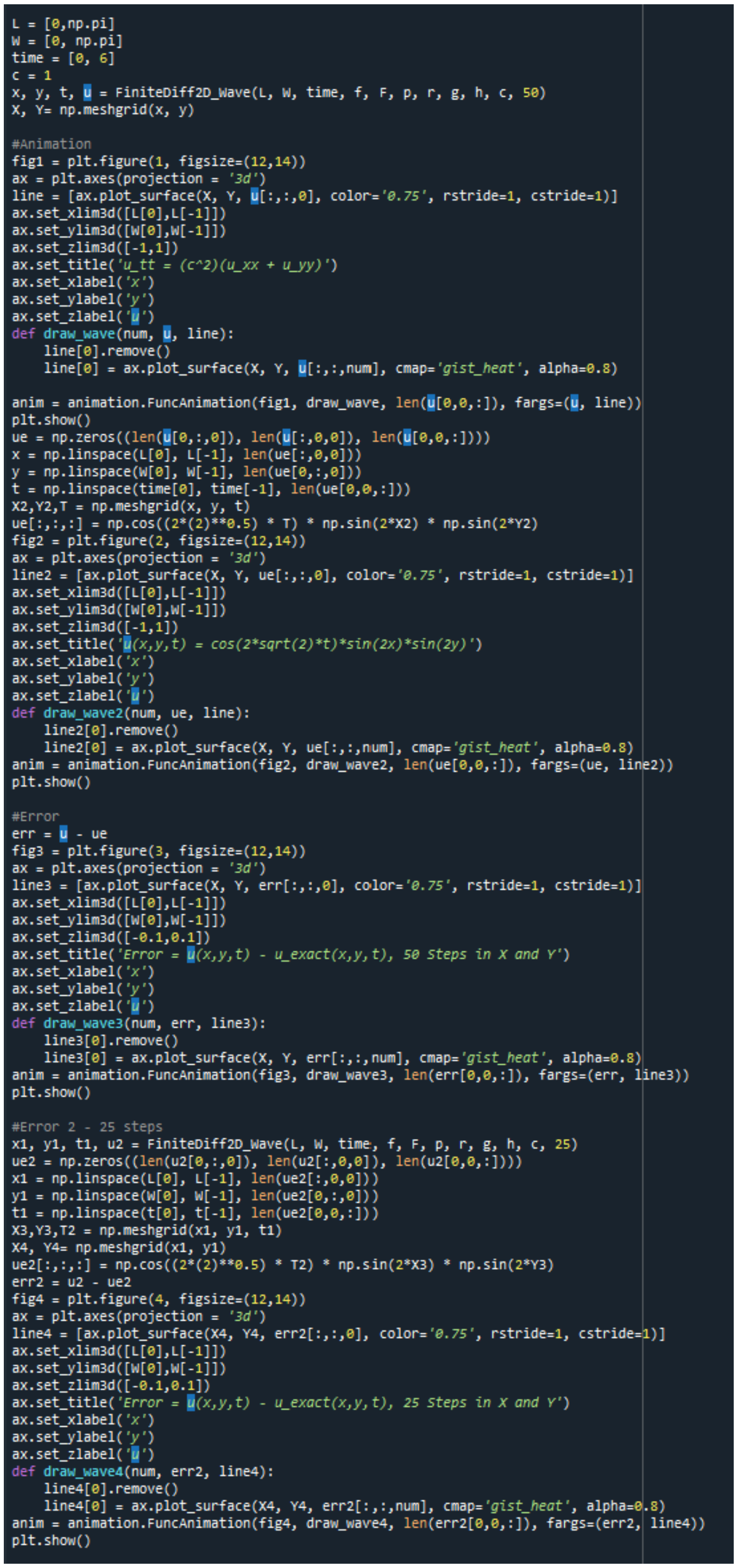
```python
L = [0,10]
time = [0, 20]
c = 2**0.5
fig1 = plt.figure(1, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

fig2 = plt.figure(2, figsize=(12,14))
ax = plt.axes(projection = '3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u, cmap='gist_heat', alpha=0.8)
ax.set_title('Wave Eqn Solution x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('U(x,t)')
plt.show()

#Animation
fig3 = plt.figure(3)
ax = plt.axes(xlim=(0,10),ylim=(-1.2,1.2))
ax.set_xlabel('x')
ax.set_ylabel('u')
ax.set_title('U_tt = (c^2)U_xx')

line = [[],[]]
line[0], = ax.plot([], [], linewidth=2.0,color='red', label='Numerical Solution')
line[1], = ax.plot([], [], linewidth=2.0,color='black', label='Initial Condition')

def draw_wave(T):
    line[0].set_data(x,u[T,:])
    line[1].set_data(x,u[0,:])

anim = animation.FuncAnimation(fig3, draw_wave, frames=len(u[:,0]))

#Exact
fig4 = plt.figure(6, figsize=(12,14))
ax = plt.axes(projection = '3d')
x = np.linspace(L[0], L[1], 100)
t = np.linspace(time[0], time[1], 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, (0.5*(f(X + c*T) + f(X-c*T)) + (1/(2*c))*(G(X + c*T) - G(X - c*T))), cmap='gist_heat', alpha=0.8)
ax.set_title('u(x,t) Exact Solution from d\'Alembert\'s')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u(x,t)')
plt.show()

#Error
#time = [0, 7]
fig5 = plt.figure(7, figsize=(12,14))
ax = fig5.add_subplot(1, 3, 1, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 50)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - (0.5*(f(X + c*T) + f(X-c*T)) + (1/(2*c))*(G(X + c*T) - G(X - c*T))), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 50')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
ax = fig5.add_subplot(1, 3, 2, projection='3d')
x, t, u = FiniteDiffPDE_Wave(L, time, f, F, p, r, c, 100)
X, T = np.meshgrid(x, t)
ax.plot_surface(X, T, u - (0.5*(f(X + c*T) + f(X-c*T)) + (1/(2*c))*(G(X + c*T) - G(X - c*T))), cmap='gist_heat', alpha=0.8)
ax.set_title('Error x-steps = 100')
ax.set_xlabel('x')
ax.set_ylabel('t')
ax.set_zlabel('u_est(x,t) - u_exact(x,t)')
```

Figure 40: Part 3 Graphing Code



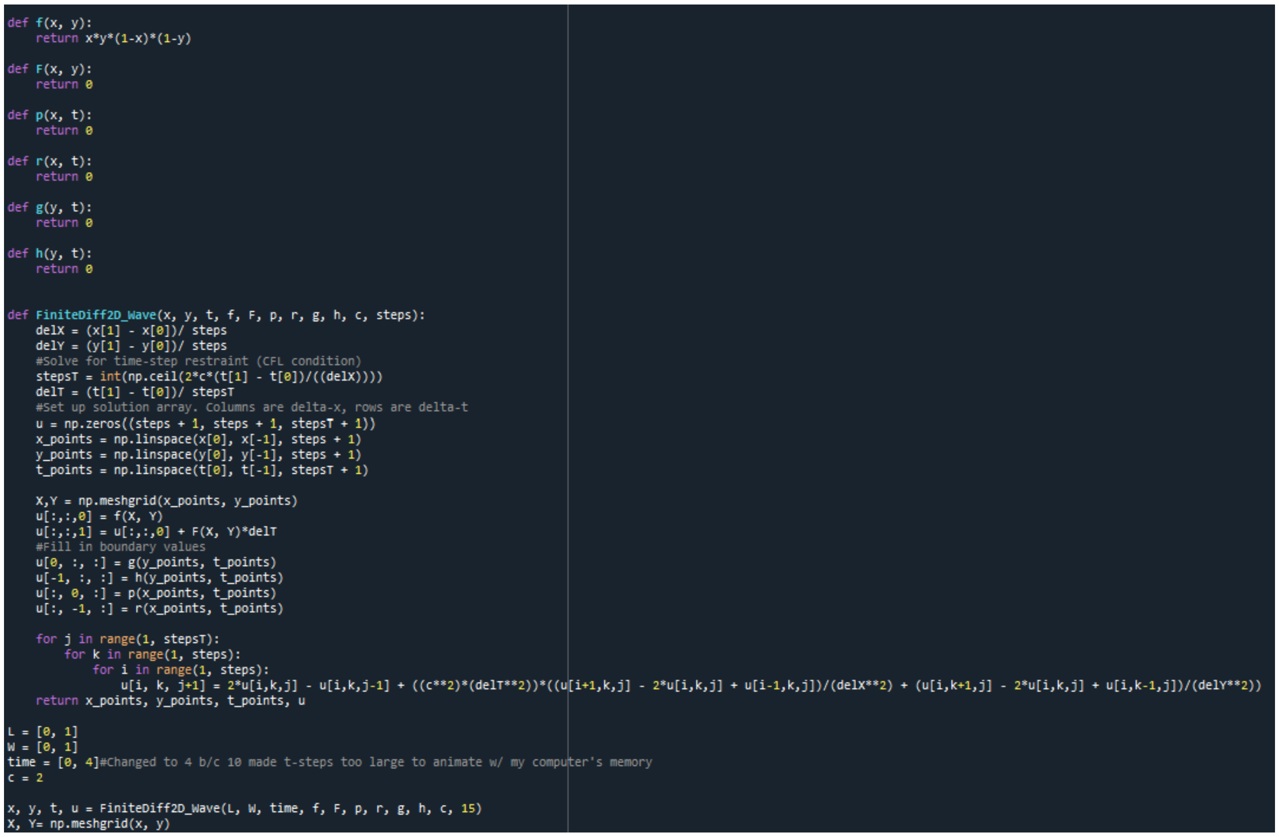
```python
def f(x, y):
    return (np.sin(2*x)*np.sin(2*y))

def F(x, y):
    return 0

def p(x, t):
    return 0

def r(x, t):
    return 0

def g(y, t):
    return 0

def h(y, t):
    return 0


def FiniteDiff2D_Wave(x, y, t, f, F, p, r, g, h, c, steps):
    delX = (x[1] - x[0])/ steps
    delY = (y[1] - y[0])/ steps
    #Solve for time-step restraint (CFL condition)
    stepsT = int(np.ceil(2*c*(t[1] - t[0])/((delX))))
    delT = (t[1] - t[0])/ stepsT
    #Set up solution array. Columns are delta-x, rows are delta-t
    u = np.zeros((steps + 1, steps + 1, stepsT + 1))
    x_points = np.linspace(x[0], x[-1], steps + 1)
    y_points = np.linspace(y[0], y[-1], steps + 1)
    t_points = np.linspace(t[0], t[-1], stepsT + 1)

    X,Y = np.meshgrid(x_points, y_points)
    u[:,:,0] = f(X, Y)
    u[:,:,1] = u[:,:,0] + F(X, Y)*delT
    #Fill in boundary values
    u[0, :, :] = g(y_points, t_points)
    u[-1, :, :] = h(y_points, t_points)
    u[:, 0, :] = p(x_points, t_points)
    u[:, -1, :] = r(x_points, t_points)

    for j in range(1, stepsT):
        for k in range(1, steps):
            for i in range(1, steps):
                u[i, k, j+1] = 2*u[i,k,j] - u[i,k,j-1] + ((c**2)*(delT**2))*((u[i+1,k,j] - 2*u[i,k,j] + u[i-1,k,j])/(delX**2) + (u[i,k+1,j] - 2*u[i,k,j] + u[i,k-1,j])/(delY**2))
    return x_points, y_points, t_points, u
```

Figure 41: Part 4 Method Code



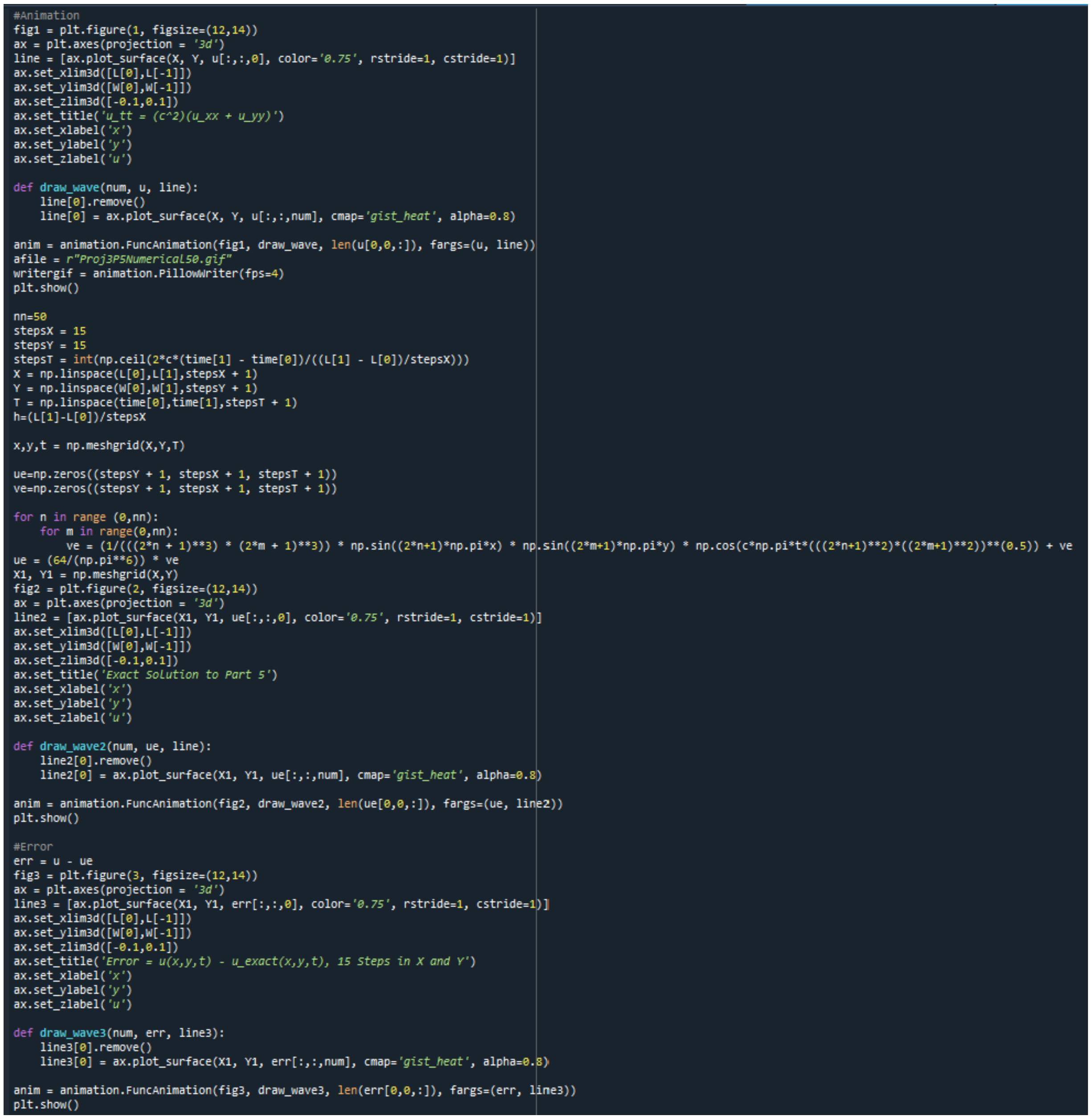
```python
L = [0,np.pi]
W = [0, np.pi]
time = [0, 6]
c = 1
x, y, t, u = FiniteDiff2D_Wave(L, W, time, f, F, p, r, g, h, c, 50)
X, Y= np.meshgrid(x, y)

#Animation
fig1 = plt.figure(1, figsize=(12,14))
ax = plt.axes(projection = '3d')
line = [ax.plot_surface(X, Y, u[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-1,1])
ax.set_title('u_tt = (c^2)(u_xx + u_yy)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')
def draw_wave(num, u, line):
    line[0].remove()
    line[0] = ax.plot_surface(X, Y, u[:,:,num], cmap='gist_heat', alpha=0.8)

anim = animation.FuncAnimation(fig1, draw_wave, len(u[0,0,:]), fargs=(u, line))
plt.show()
ue = np.zeros((len(u[0,:,0]), len(u[:,0,0]), len(u[0,0,:])))
x = np.linspace(L[0], L[-1], len(ue[:,0,0]))
y = np.linspace(W[0], W[-1], len(ue[0,:,0]))
t = np.linspace(time[0], time[-1], len(ue[0,0,:]))
X2,Y2,T = np.meshgrid(x, y, t)
ue[:,:,:] = np.cos((2*(2)**0.5) * T) * np.sin(2*X2) * np.sin(2*Y2)
fig2 = plt.figure(2, figsize=(12,14))
ax = plt.axes(projection = '3d')
line2 = [ax.plot_surface(X, Y, ue[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-1,1])
ax.set_title('u(x,y,t) = cos(2*sqrt(2)*t)*sin(2x)*sin(2y)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')
def draw_wave2(num, ue, line):
    line2[0].remove()
    line2[0] = ax.plot_surface(X, Y, ue[:,:,num], cmap='gist_heat', alpha=0.8)
anim = animation.FuncAnimation(fig2, draw_wave2, len(ue[0,0,:]), fargs=(ue, line2))
plt.show()

#Error
err = u - ue
fig3 = plt.figure(3, figsize=(12,14))
ax = plt.axes(projection = '3d')
line3 = [ax.plot_surface(X, Y, err[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-0.1,0.1])
ax.set_title('Error = u(x,y,t) - u_exact(x,y,t), 50 Steps in X and Y')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')
def draw_wave3(num, err, line3):
    line3[0].remove()
    line3[0] = ax.plot_surface(X, Y, err[:,:,num], cmap='gist_heat', alpha=0.8)
anim = animation.FuncAnimation(fig3, draw_wave3, len(err[0,0,:]), fargs=(err, line3))
plt.show()

#Error 2 - 25 steps
x1, y1, t1, u2 = FiniteDiff2D_Wave(L, W, time, f, F, p, r, g, h, c, 25)
ue2 = np.zeros((len(u2[0,:,0]), len(u2[:,0,0]), len(u2[0,0,:])))
x1 = np.linspace(L[0], L[-1], len(ue2[:,0,0]))
y1 = np.linspace(W[0], W[-1], len(ue2[0,:,0]))
t1 = np.linspace(t[0], t[-1], len(ue2[0,0,:]))
X3,Y3,T2 = np.meshgrid(x1, y1, t1)
X4, Y4= np.meshgrid(x1, y1)
ue2[:,:,:] = np.cos((2*(2)**0.5) * T2) * np.sin(2*X3) * np.sin(2*Y3)
err2 = u2 - ue2
fig4 = plt.figure(4, figsize=(12,14))
ax = plt.axes(projection = '3d')
line4 = [ax.plot_surface(X4, Y4, err2[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-0.1,0.1])
ax.set_title('Error = u(x,y,t) - u_exact(x,y,t), 25 Steps in X and Y')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')
def draw_wave4(num, err2, line4):
    line4[0].remove()
    line4[0] = ax.plot_surface(X4, Y4, err2[:,:,num], cmap='gist_heat', alpha=0.8)
anim = animation.FuncAnimation(fig4, draw_wave4, len(err2[0,0,:]), fargs=(err2, line4))
plt.show()
```

Figure 42: Part 4 Graphing Code



```python
def f(x, y):
    return x*y*(1-x)*(1-y)

def F(x, y):
    return 0

def p(x, t):
    return 0

def r(x, t):
    return 0

def g(y, t):
    return 0

def h(y, t):
    return 0


def FiniteDiff2D_Wave(x, y, t, f, F, p, r, g, h, c, steps):
    delX = (x[1] - x[0])/ steps
    delY = (y[1] - y[0])/ steps
    #Solve for time-step restraint (CFL condition)
    stepsT = int(np.ceil(2*c*(t[1] - t[0])/((delX))))
    delT = (t[1] - t[0])/ stepsT
    #Set up solution array. Columns are delta-x, rows are delta-t
    u = np.zeros((steps + 1, steps + 1, stepsT + 1))
    x_points = np.linspace(x[0], x[-1], steps + 1)
    y_points = np.linspace(y[0], y[-1], steps + 1)
    t_points = np.linspace(t[0], t[-1], stepsT + 1)

    X,Y = np.meshgrid(x_points, y_points)
    u[:,:,0] = f(X, Y)
    u[:,:,1] = u[:,:,0] + F(X, Y)*delT
    #Fill in boundary values
    u[0, :, :] = g(y_points, t_points)
    u[-1, :, :] = h(y_points, t_points)
    u[:, 0, :] = p(x_points, t_points)
    u[:, -1, :] = r(x_points, t_points)

    for j in range(1, stepsT):
        for k in range(1, steps):
            for i in range(1, steps):
                u[i, k, j+1] = 2*u[i,k,j] - u[i,k,j-1] + ((c**2)*(delT**2))*((u[i+1,k,j] - 2*u[i,k,j] + u[i-1,k,j])/(delX**2) + (u[i,k+1,j] - 2*u[i,k,j] + u[i,k-1,j])/(delY**2))
    return x_points, y_points, t_points, u

L = [0, 1]
W = [0, 1]
time = [0, 4]#Changed to 4 b/c 10 made t-steps too large to animate w/ my computer's memory
c = 2

x, y, t, u = FiniteDiff2D_Wave(L, W, time, f, F, p, r, g, h, c, 15)
X, Y= np.meshgrid(x, y)
```

Figure 43: Part 5 Method Code



```python
#Animation
fig1 = plt.figure(1, figsize=(12,14))
ax = plt.axes(projection = '3d')
line = [ax.plot_surface(X, Y, u[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-0.1,0.1])
ax.set_title('u_tt = (c^2)(u_xx + u_yy)')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')

def draw_wave(num, u, line):
    line[0].remove()
    line[0] = ax.plot_surface(X, Y, u[:,:,num], cmap='gist_heat', alpha=0.8)

anim = animation.FuncAnimation(fig1, draw_wave, len(u[0,0,:]), fargs=(u, line))
afile = r"Proj3P5Numerical50.gif"
writergif = animation.PillowWriter(fps=4)
plt.show()

nn=50
stepsX = 15
stepsY = 15
stepsT = int(np.ceil(2*c*(time[1] - time[0])/((L[1] - L[0])/stepsX)))
X = np.linspace(L[0],L[1],stepsX + 1)
Y = np.linspace(W[0],W[1],stepsY + 1)
T = np.linspace(time[0],time[1],stepsT + 1)
h=(L[1]-L[0])/stepsX

x,y,t = np.meshgrid(X,Y,T)

ue=np.zeros((stepsY + 1, stepsX + 1, stepsT + 1))
ve=np.zeros((stepsY + 1, stepsX + 1, stepsT + 1))

for n in range (0,nn):
    for m in range(0,nn):
        ve = (1/(((2*n + 1)**3) * (2*m + 1)**3)) * np.sin((2*n+1)*np.pi*x) * np.sin((2*m+1)*np.pi*y) * np.cos(c*np.pi*t*(((2*n+1)**2)*((2*m+1)**2))**(0.5)) + ve
ue = (64/(np.pi**6)) * ve
X1, Y1 = np.meshgrid(X,Y)
fig2 = plt.figure(2, figsize=(12,14))
ax = plt.axes(projection = '3d')
line2 = [ax.plot_surface(X1, Y1, ue[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-0.1,0.1])
ax.set_title('Exact Solution to Part 5')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')

def draw_wave2(num, ue, line):
    line2[0].remove()
    line2[0] = ax.plot_surface(X1, Y1, ue[:,:,num], cmap='gist_heat', alpha=0.8)

anim = animation.FuncAnimation(fig2, draw_wave2, len(ue[0,0,:]), fargs=(ue, line2))
plt.show()

#Error
err = u - ue
fig3 = plt.figure(3, figsize=(12,14))
ax = plt.axes(projection = '3d')
line3 = [ax.plot_surface(X1, Y1, err[:,:,0], color='0.75', rstride=1, cstride=1)]
ax.set_xlim3d([L[0],L[-1]])
ax.set_ylim3d([W[0],W[-1]])
ax.set_zlim3d([-0.1,0.1])
ax.set_title('Error = u(x,y,t) - u_exact(x,y,t), 15 Steps in X and Y')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('u')

def draw_wave3(num, err, line3):
    line3[0].remove()
    line3[0] = ax.plot_surface(X1, Y1, err[:,:,num], cmap='gist_heat', alpha=0.8)

anim = animation.FuncAnimation(fig3, draw_wave3, len(err[0,0,:]), fargs=(err, line3))
plt.show()
```

Figure 44: Part 5 Graphing Code