Explanation of the Wave Equation Simulation Code

1 Introduction

This document provides an explanation of the Python code used to simulate wave propagation using the 2D wave equation. The code leverages numerical methods to solve the wave equation and visualize the wave's behavior over time.

2 Wave Equation Simulation

The provided code simulates the propagation of a wave using the 2D wave equation. The wave equation describes how waves move through a medium and is essential in many physical systems, including acoustics, electromagnetism, and fluid dynamics.

2.1 Constants and Grid Initialization

```
import numpy as np
import matplotlib.pyplot as plt
import copy
from matplotlib import cm
from matplotlib.animation import FuncAnimation, FFMpegWriter

# Initialize physical and numerical constants
L = 1.0  # set the length of the system
dx = 0.01  # set the discrete spatial stepsize
c = 1.0  # define the wave speed

dt = .707 * dx / c  # choose a time step to satisfy the CFL condition

x = np.arange(0, L + dx, dx)  # define an array to store x position data
y = np.arange(0, L + dx, dx)  # define an array to store y position data

xx, yy = np.meshgrid(x, y)
```

- **L (length of the system)**: This represents the size of the spatial domain in which the wave propagates.
- **dx (spatial step size)**: This is the distance between discrete spatial points in the grid.
- **c (wave speed)**: The speed at which the wave propagates through the medium.
- **dt (time step)**: This is calculated to satisfy the Courant-Friedrichs-Lewy (CFL) condition, which ensures numerical stability. The condition states that the numerical domain of dependence must include the physical domain of dependence.

2.2 Arrays and Initial Conditions

```
npts = len(x)  # this is the number of spatial points along x
nsteps = 900  # set the number of time steps to get a 30 second animation at 30

f = np.zeros((npts, npts, 3))

xc = 0.5  # define the center of the system to locate a Gaussian pulse
w = 0.05  # define the width of the Gaussian wave pulse

f[:, :, 0] = np.exp(-(xx - xc) ** 2 / (w ** 2)) * np.exp(-(yy - xc) ** 2 / (w **
# initial condition for a Gaussian
```

- **npts**: Number of spatial points along the x (and y) direction.
- **nsteps**: Number of time steps for the animation (900 steps for 30 seconds at 30 FPS).
- **f**: This is a 3D array to store the wave amplitude at different time steps.
- **xc and w**: Center and width of the initial Gaussian pulse. The initial condition is set as a Gaussian wave packet centered at (xc, xc) with width w.

2.3 Leapfrog Time Integration

```
# First time step in the leap frog algorithm
f[1:-1, 1:-1, 1] = f[1:-1, 1:-1, 0] + 0.5 * c ** 2 * (f[:-2, 1:-1, 0] + f[2:, 1:
                                                    + 0.5 * c ** 2 * (f[1:-1, :-2, 0] + f[1:-1,
```

This sets up the wave at the first time step using the initial condition and the wave equation. The leapfrog algorithm is used for time integration:

$$f_{i,j}^{n+1} = f_{i,j}^{n-1} + 2 \cdot \Delta t^2 \left(\frac{c^2}{\Delta x^2}(f_{i-1,j}^n + f_{i+1,j}^n - 2f_{i,j}^n) + \frac{c^2}{\Delta y^2}(f_{i,j-1}^n + f_{i,j+1}^n - 2f_{i,j}^n) \right)$$

2.4 Animation Setup

```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')

def update(frame):
    global f
    # For all additional time steps
    for _ in range(1):
        f[1:-1, 1:-1, 2] = -f[1:-1, 1:-1, 0] + 2 * f[1:-1, 1:-1, 1] \
                           + c ** 2 * (f[:-2, 1:-1, 1] + f[2:, 1:-1, 1] - 2. * f
                           + c ** 2 * (f[1:-1, :-2, 1] + f[1:-1, 2:, 1] - 2. * f

        # Push the data back for the leapfrogging
        f[:, :, 0] = f[:, :, 1]
        f[:, :, 1] = f[:, :, 2]

    ax.clear()
    ax.plot_surface(xx, yy, f[:, :, 2], rstride=1, cstride=1, cmap=cm.coolwarm)
    ax.plot_wireframe(xx, yy, f[:, :, 2], rstride=10, cstride=10, color='green')
    plt.title(f't= {frame * dt:.2f}')
    ax.set_zlim(-.25, 1)
```

- **update function**: This updates the wave amplitude at each frame. It uses the leapfrog method to compute the new values of the wave function and updates the plot.
- **ax.clear()**: Clears the previous plot.
- **ax.plot_surface() and ax.plot_wireframe()**: Plot the surface and wireframe of the wave function.

2.5 Animation and Video Saving

```
# Create the animation
ani = FuncAnimation(fig, update, frames=nsteps, repeat=False)

# Save the animation to a video file
writer = FFMpegWriter(fps=30, metadata=dict(artist='Me'), bitrate=1800)
ani.save("wave_animation.mp4", writer=writer)
```

- **FuncAnimation**: Creates the animation by repeatedly calling the `update` function.
- **FFMpegWriter**: Specifies the video writer, frame rate (30 FPS), and bitrate for the output video file.

3 Physical Properties

- **Wave Propagation**: The simulation models the propagation of a wave in a 2D medium, demonstrating how an initial Gaussian pulse evolves over time.
- **Wave Speed (c)**: The parameter `c` represents the speed of the wave, which affects how fast the wave propagates through the medium.
- **Spatial and Temporal Resolution**: The choice of `dx` and `dt` ensures numerical stability and accuracy in capturing the wave dynamics.
- **Gaussian Pulse**: The initial condition represents a localized wave packet that spreads out over time, which is a common scenario in wave physics.

This code, therefore, provides a numerical solution to the 2D wave equation, visualizing how waves propagate in a given medium over time.