Two-Dimensional Wave Simulator

Isabella Pestovski
May 6, 2019

PH 235: Physics Simulations
Professor Raja
The Cooper Union for the Advancement of Science and Art

Table of Contents

Introduction

This paper explains the development of a simulator with an interface that allows the user to visualize solutions to the wave equation. The wave equation, reproduced below, is an important equation in mathematics and physics describing the oscillation of particles in space.

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

Solutions to the wave equation demonstrate the motion of mechanical as well as electromagnetic waves. In order to solve the equation, two initial conditions and four boundary conditions need to be specified. This educational tool allows students to explore the effects of various initial conditions to understand the behavior of wave propagation as well as how to mathematically model physical phenomena. The graphical user interface (GUI) is ideal for classroom demonstrations.

Two simulators were made for both the one and two-dimensional cases of the wave equation. For the one-dimensional case, a string of unit length is considered, with fixed displacement at the ends. For the two-dimensional case, a rectangle of unit area is considered with fixed displacement at the edges. Three initial condition cases are calculated for each simulator using the forward-time centered-space finite difference method. The GUI provides a home screen for the user to select a button to show the requested physical behavior. Selecting a choice generates a new frame showing a looped animation of the resulting wave. The user is able to switch between all the options for as long as they want.

There are other wave equation simulations that allow the user to draw[1] the initial displacement and velocity, but because the user is drawing these the resulting animation cannot represent perfect standing wave behavior or reflected propagation. Furthermore, the project as stated is also educational for learning about computational methods for solving partial differential equations.

Computational Approach

A finite difference method was used since it is fairly straightforward and not computationally expensive. Finite difference methods are good for rectangular shaped domains, as is the case for the two-dimensional equation. The first step is discretizing the time and space points into a grid. Using this grid, derivatives can be approximated through the use of finite differences. For example, the second derivative can be approximated as

$$f_{xx} \approx \frac{f(x+h,y) - 2f(x,y) + f(x-h,y)}{h^2}$$

Where h is the size of each step taken in the x grid. This is an explicit method in that each step is forward in time and based on the previously calculated values, whereas implicit methods rely on solving systems of numerical equations at each time step.

One-Dimensional Model

Utilizing these approximations, the one-dimensional wave equation can be rewritten as

$$\frac{u_j^{n+1} - 2u_j^n + u_j^{n-1}}{\Delta t^2} = c\,\frac{u_{j+1}^n - 2u_j^n + u_{j-1}^n}{\Delta x^2}$$

Here, j is the index for space steps and n is the index for time steps. Rearranging this and defining r as $c\frac{\Delta t}{\Delta x}$, we get the explicit algorithm for approximating the wave equation:

$$u_j^{n+1} = \left(r^2 u_{j+1}^n + 2(1-r^2)u_j^n + r^2 u_{j-1}^n\right) - u_j^{n-1}$$

This works for all values of n except for when n=0, which is the first time-step. Therefore, initial conditions have to be set. This is done by approximating u_j^{-1} using the approximation for the first derivative:

$$g(x,0) = \frac{u_j^1 - u_j^{-1}}{2\Delta t}$$

u_j^1 is solved for here, and is then substituted into the explicit algorithm for when n=0, resulting in an explicit expression for the first time-step:

$$u_j^1 = \frac{1}{2}\left(r^2 u_{j+1}^0 + 2(1-r^2)u_j^0 + r^2 u_{j-1}^0\right) + \Delta t g(x,0)$$

In the program, u is defined as a two-dimensional array with the first axis denoting the x-values and the second axis denoting the t-values. The initial conditions are set by calculating the initial displacement condition for all values of x for when t=0. Three initial conditions are explored in this model: 1) u(x,0) = sin(3πx), 2) u(x,0) = -cos(3πx)sin(πx), 3) u(x,0) = 0, g(x,0) = 1. Each of these initial conditions produces different physical behavior: 1) 3-node standing wave, 2) split wave propagation, 3) single pulse. Figures of these can be seen in the results section.

Two-Dimensional Model

A similar process was followed to develop the two-dimensional model resulting in the following explicit algorithm with i representing x index and j representing y index:

$$u_{i,j}^{n+1} = r^2\left(u_{i+1,j}^n - 2u_{i,j}^n + u_{i-1,j}^n\right) + r^2\left(u_{i,j+1}^n - 2u_{i,j}^n + u_{i,j-1}^n\right) + 2u_{i,j}^n - u_{i,j}^{n-1}$$

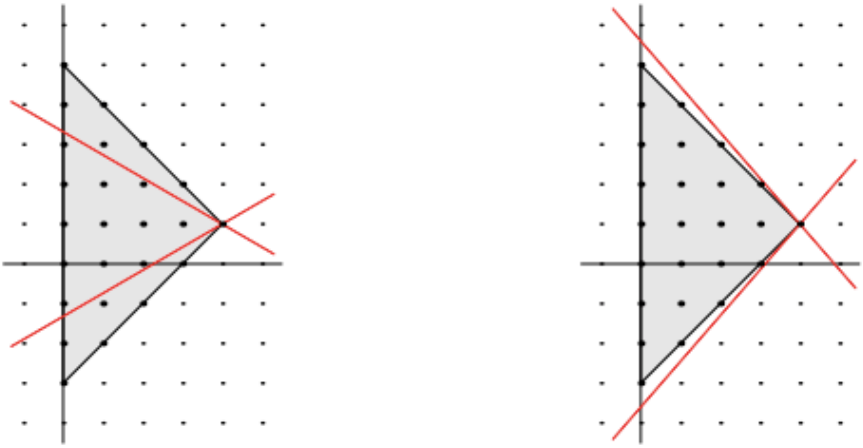
A similar approximation for $u_{i,j}^{-1}$ must be made, resulting in the following explicit expression for the first time-step:

$$u_j^1 = r^2\left(u_{i+1,j}^0 - 2u_{i,j}^0 + u_{i-1,j}^0\right) + r^2\left(u_{i,j+1}^0 - 2u_{i,j}^0 + u_{i,j-1}^0\right) + u_{i,j}^0$$

In the program, u is defined as a three-dimensional array with the first axis denoting the x-values, the second axis denoting the y-values, and the third axis denoting the t-values. The initial conditions are set by calculating the initial displacement condition for all values of x and y for when t=0. Three initial conditions are explored in this model: 1) u(x,y,0) = sin(3πx)sin(3πy), 2) u(x,y,0) = cos(3πx)sin(πx)cos(3πy)sin(πy), 3) u(x,y,0) = $e^{-2\pi x^2}$sin(2πx) $e^{-2\pi y^2}$sin(2πy). Each of these initial conditions produces different physical behavior: 1) 3-node standing wave, 2) split wave propagation, 3) ripple. Figures of these can be seen in the results section.

Method Evaluation

In order to evaluate this method, the scheme's numerical stability was checked using the Courant-Friedrichs-Lewy (CFL) criterion. This criterion is demonstrated in the figure[2] below.



Stable Unstable

The vertical axis represents the x-values, and the horizontal axis represents the time values. The shaded portion represents the numerical domain of dependence, or the mesh values that are being calculated. This domain is bounded by lines of slope c, the wave speed. Specifically, the CFL criterion states that the scheme is numerically stable if the slope of the mesh points being calculated falls within the domain of dependence. So, there is a limit on the step sizes Δt and Δx:

$$0 \leq c \leq \frac{\Delta x}{\Delta t}$$

In other words, the wave speed cannot exceed the speed of calculations, otherwise the model will be numerically unstable. This condition is met in the program by setting c=1, $\Delta x = 0.01$ and $\Delta t = .00707$.

To confirm this, a von Neumann stability analysis has been performed, and is carried out below for the one-dimensional case. The von Neumann stability analysis is based on Fourier series analysis and is done to show that calculated errors are represented by complex exponentials that either decay (stable) or grow (unstable) over time. After each time step, the scheme multiplies a complex exponential by a magnification factor, λ. The first steps from t_{n-1} to t_{n+1} show this:

$$u_j^{n-1} = e^{ikx_j}, \qquad u_j^n = \lambda e^{ikx_j}, \qquad u_j^{n+1} = \lambda^2 e^{ikx_j}$$

These expressions are substituted into the general explicit algorithm for the one-dimensional wave equation and, after cancelling out the exponentials, results in the following quadratic:

$$\lambda^2 = \left(2 - 4r^2 \sin^2 \frac{1}{2} k\Delta x\right)\lambda + 1$$

This is solved for λ, in order to determine under which conditions the exponential terms decay or grow. The resulting values of λ are

$$\lambda = 1 - 2r^2 \sin^2 \frac{1}{2} k\Delta x \pm \sqrt{\left(1 - 2r^2 \sin^2 \frac{1}{2} k\Delta x\right)^2 - 2}$$

In order for the scheme to be stable, both λ values must be less than or equal to one. If the CFL criterion is met and $r \leq 1$, then $|\lambda| = 1$, satisfying the stability criterion. Whereas, if the CFL criterion is not met and $r > 1$ the scheme is unstable. For this program, since the CFL criterion is met, the model is stable.

Results

A GUI was made for each case with three buttons the user can choose between to see different physical behavior. Below, the home screens for each GUI are shown with stills from the animations on each page. The user can switch between each page as many times as they need.

One-Dimensional Simulator

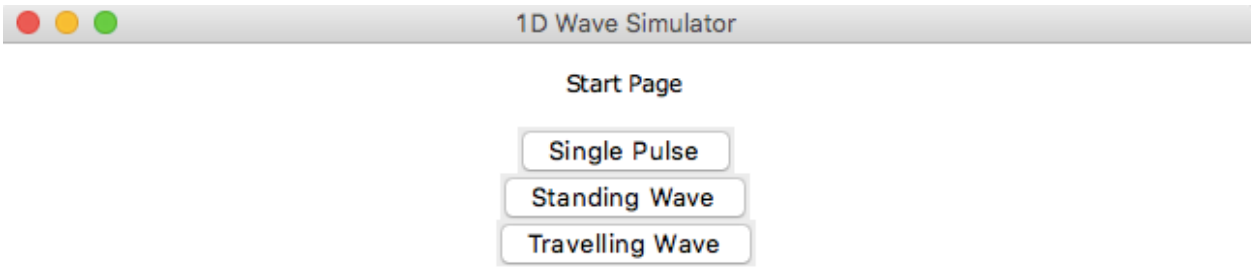


Figure 1. 1D wave simulator GUI home screen

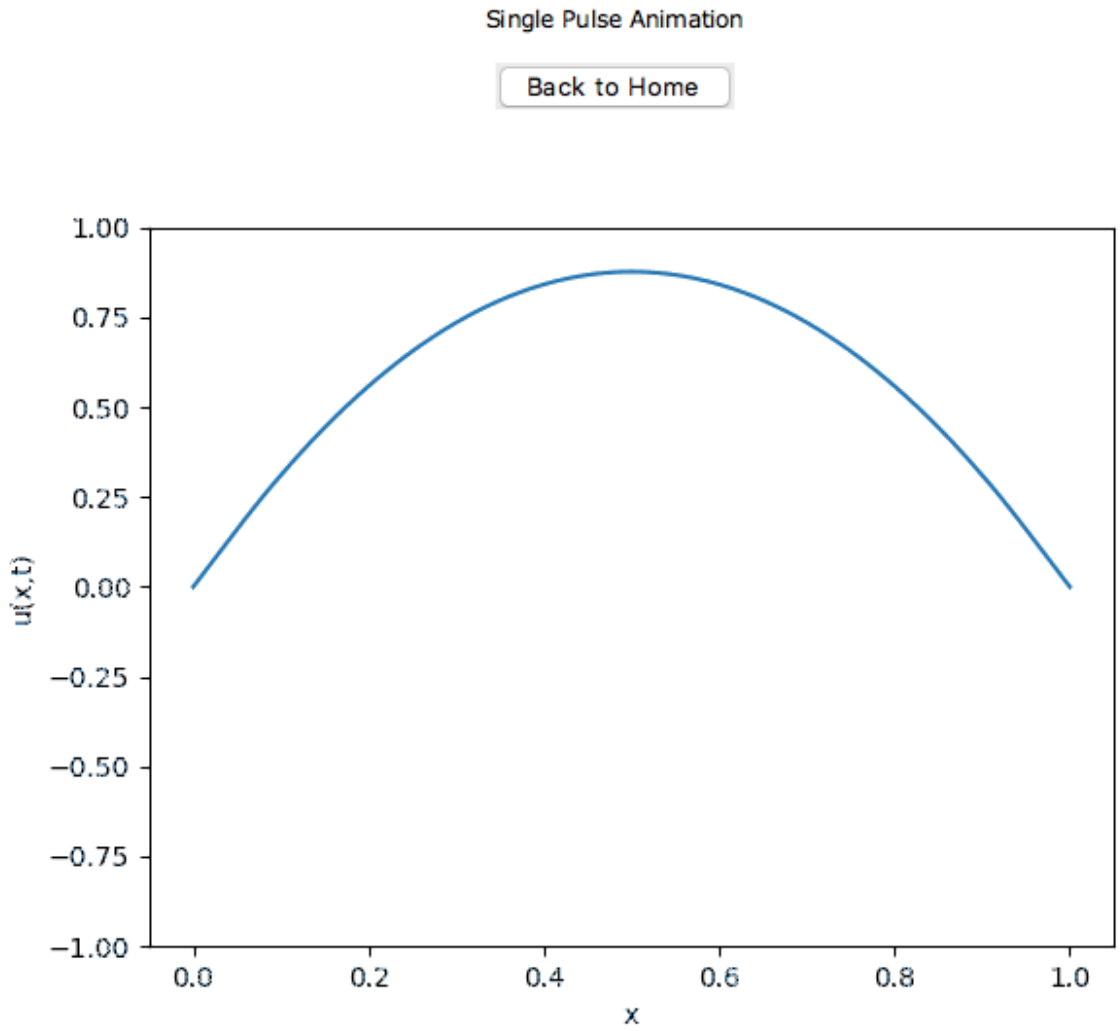


Figure 2. 1D single pulse animation still

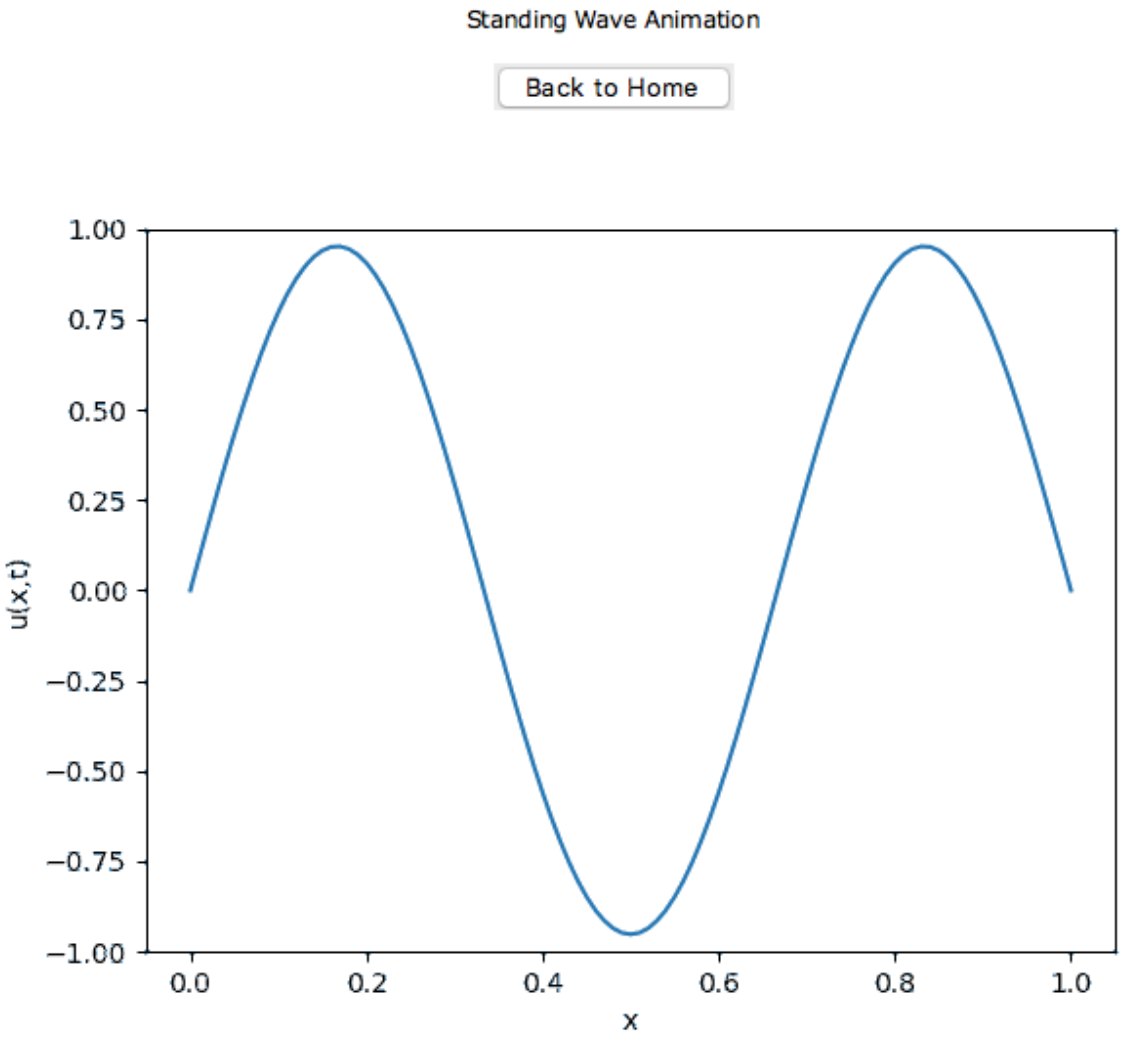


Figure 3. 1D 3-node standing wave animation still

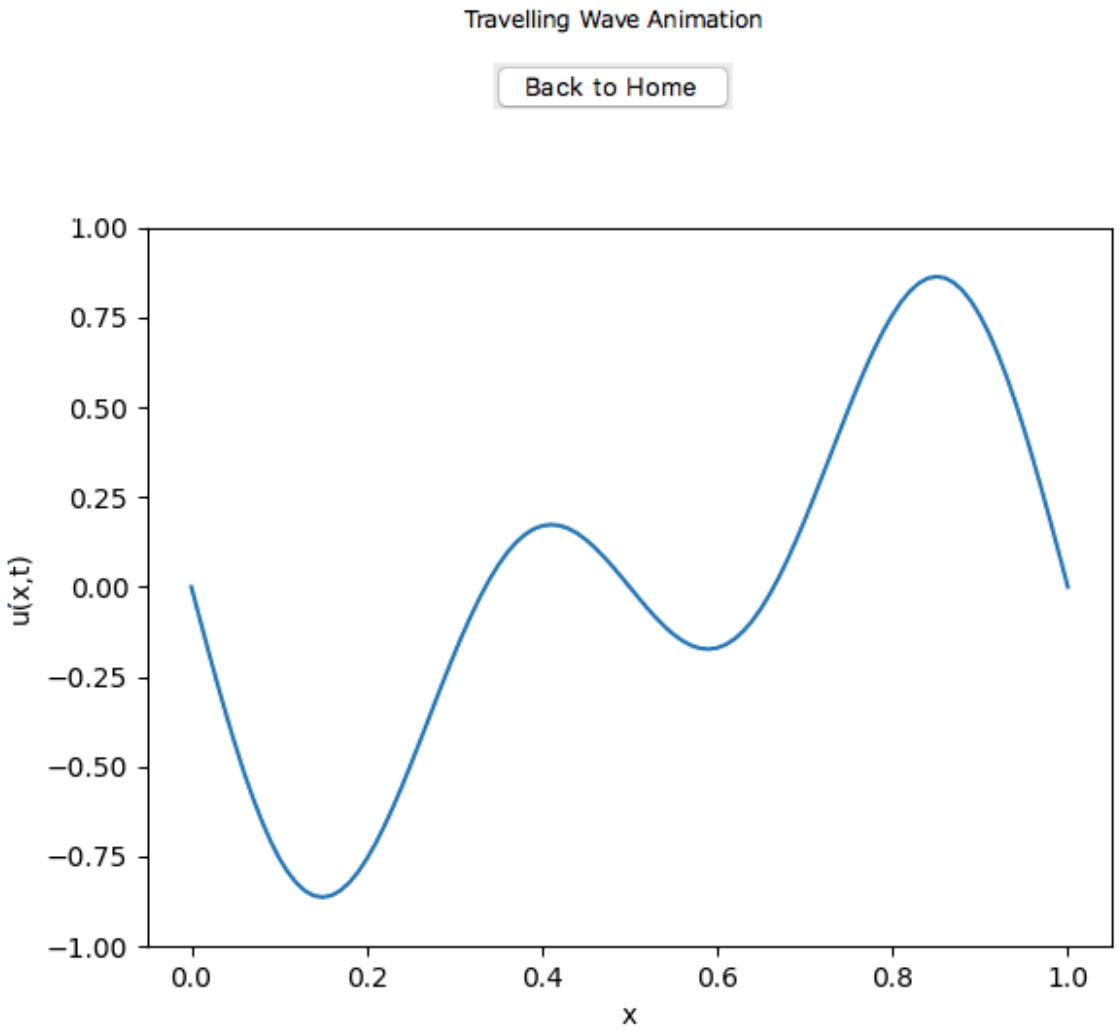


Figure 4. 1D split wave propagation animation still

Two-Dimensional Simulator

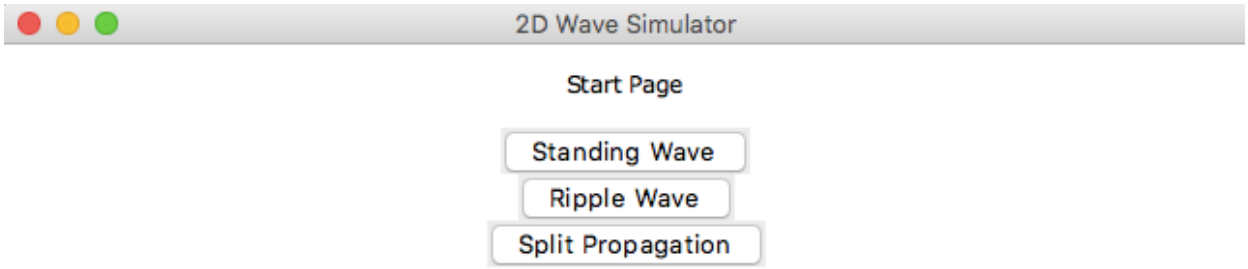


Figure 5. 2D wave simulator GUI home page

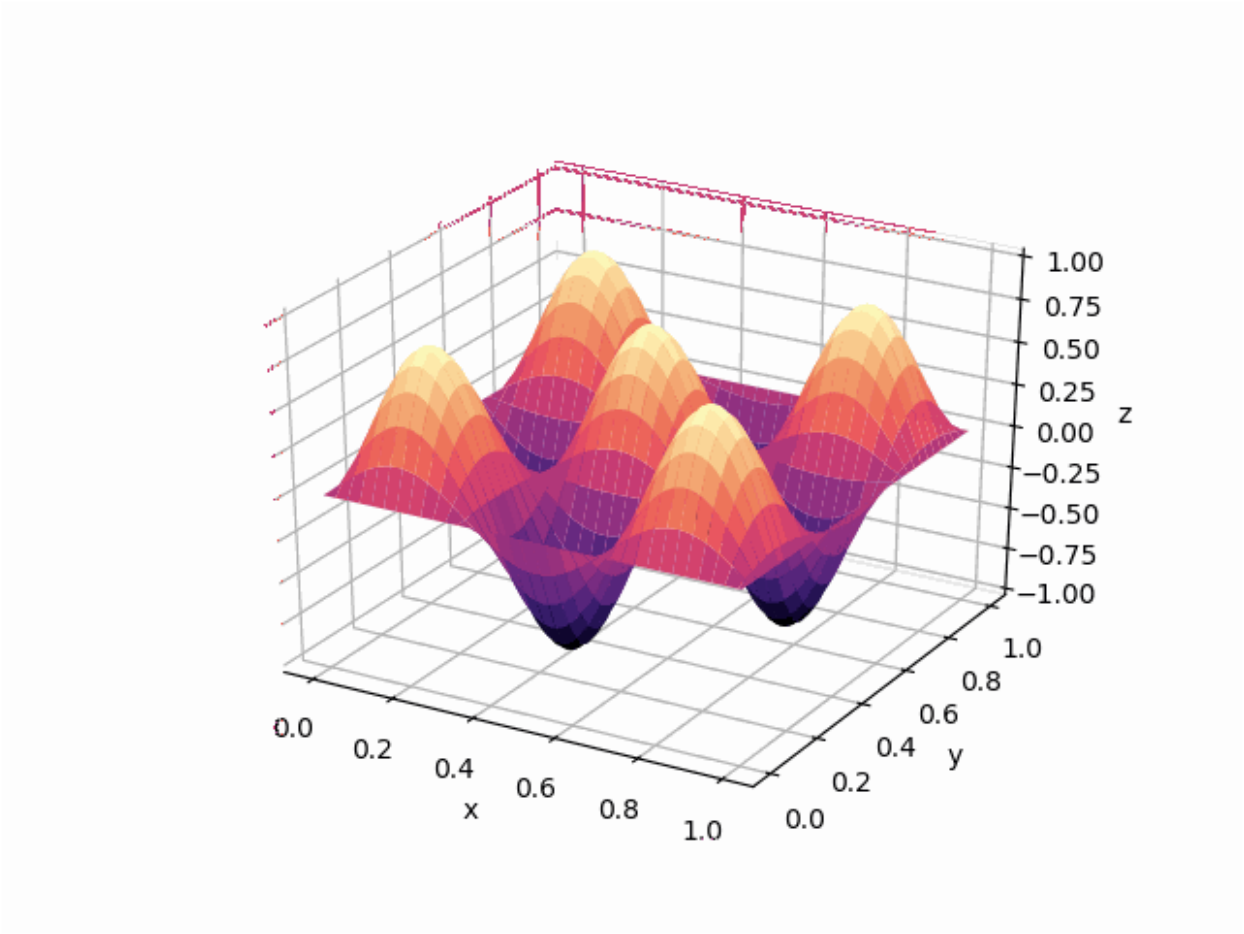


Figure 6. 2D 3-node standing wave animation still

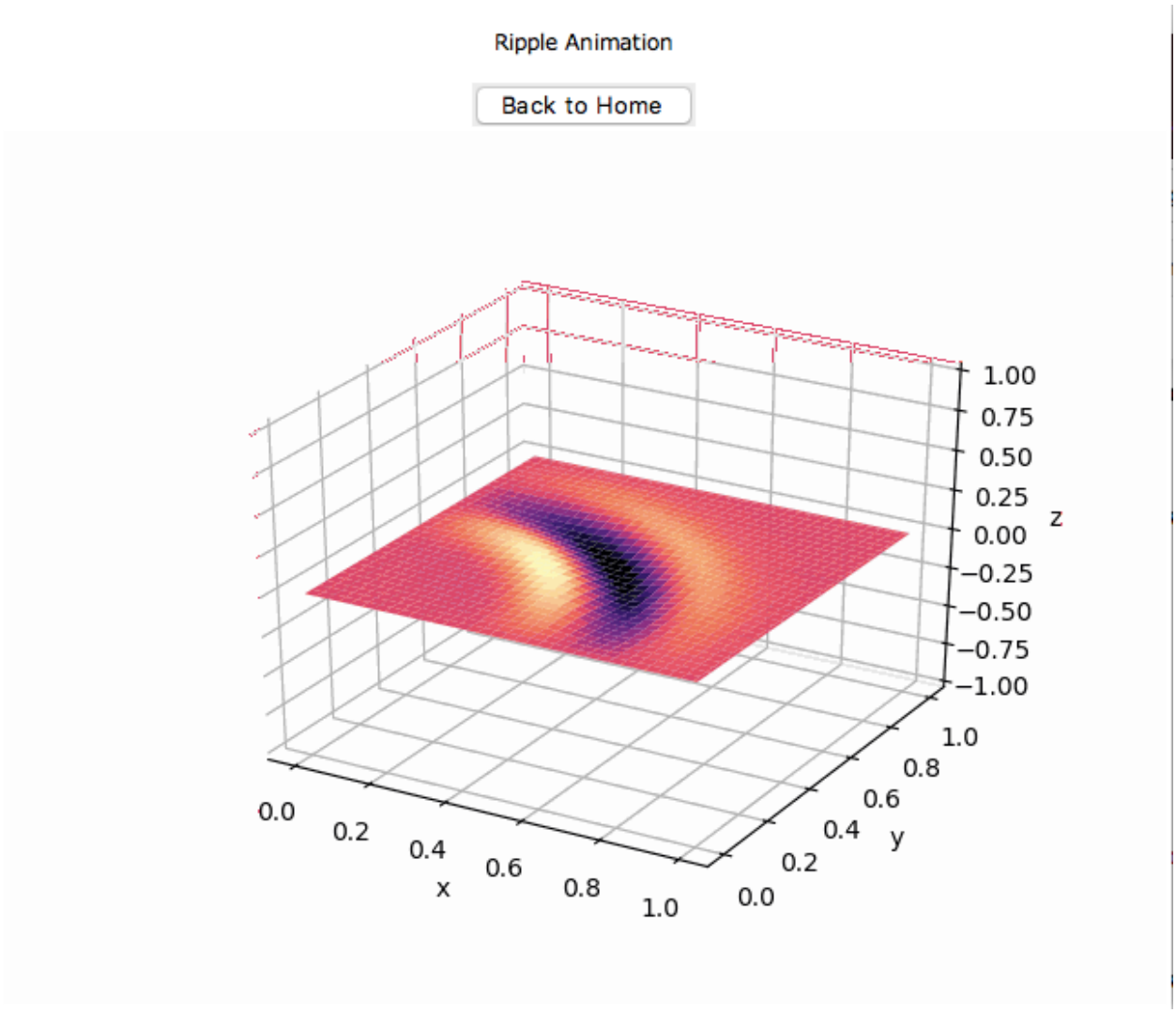


Figure 7. 2D ripple animation still

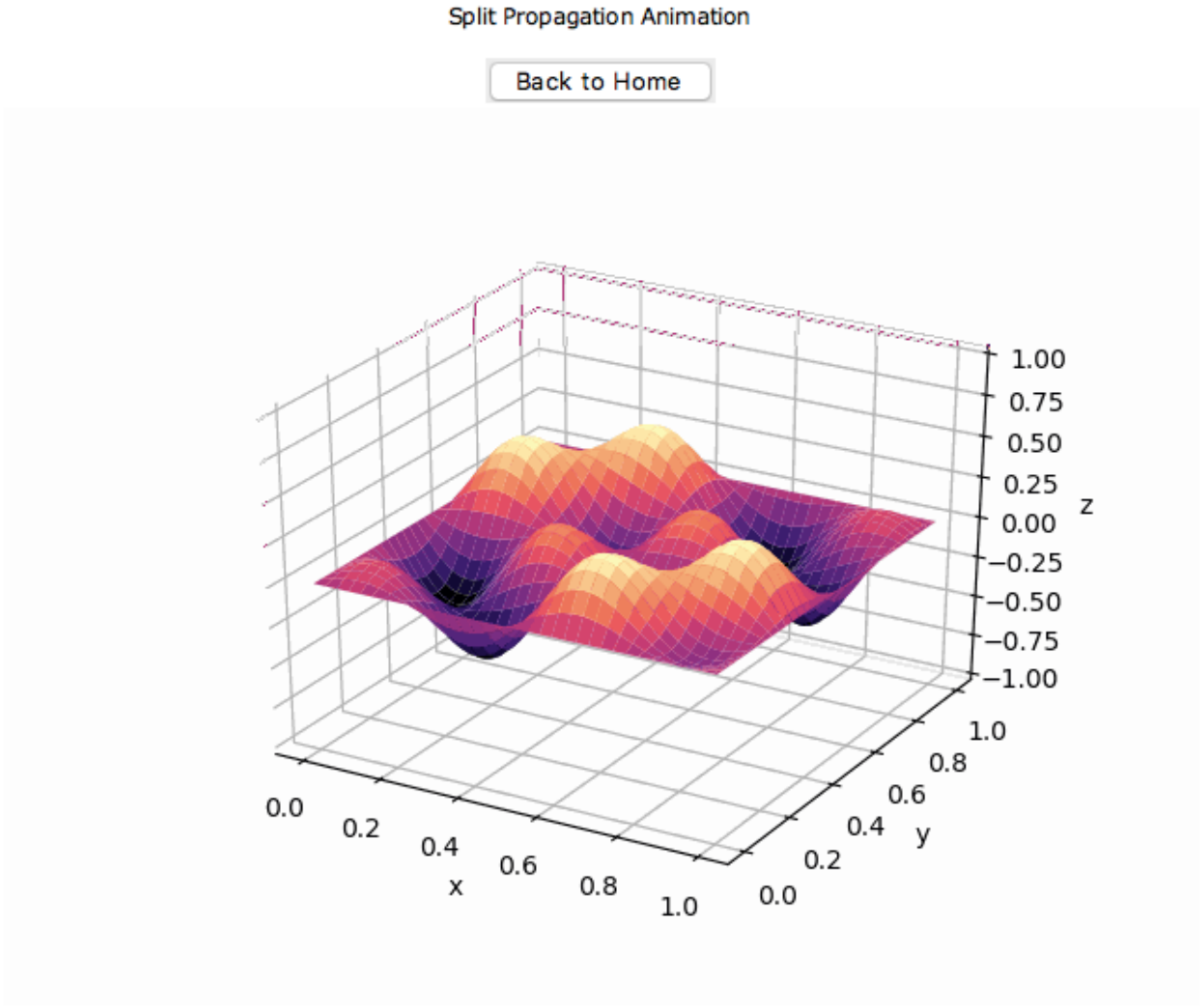


Figure 8. 2D split wave propagation animation still

Future Work

This project can be improved in several ways. While the FTCS method used here is numerically stable, a more robust numerical scheme could be used. The FTCS method as implemented here is conditionally stable for short periods of time, but other schemes are conditionally stable for longer periods of time and still maintain proper physical behavior, which is ideal for the wave equation. A further iteration of this project could include the implementation of the Crank-Nicolson algorithm, which would also be faster than the FTCS method.

In addition, the GUI could be further developed to either include more information on the home screen, or more physical behavior screens. There could also be an added feature for the user to input their desired initial condition and have the solution be provided to them in real-time.

References

1. http://math.uchicago.edu/~luis/pde/wave.html
2. http://www-users.math.umn.edu/~olver/num_/lnp.pdf

Appendix

1D Wave FTCS Method

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 24 12:54:59 2019

@author: isabellapestovski
"""
from numpy import zeros,array,linspace,nan
from math import sin,pi,sqrt,exp,cos
import matplotlib.pyplot as plt
import matplotlib.animation as animation

#sigma = 0.5
#dx = 0.01
#dt = sqrt(dx*dx*sigma)

dx = 0.01
dt = 0.00707
c = 1
r = c*dt/dx

lenx = int(1/dx)+1
lent = int(1/dt)+1

t = array([linspace(0,1,lent)])
x = array([linspace(0,1,lenx)])


u = zeros([lenx,lent])

# initial displacement condition
# for single pulse
def init_fn(x):
    return 0
# standing wave
#def init_fn(x):
#    return sin(3*pi*x)
```

```
# split wave propagation
#def init_fn(x):
#    return -cos(3*pi*x)*sin(pi*x)

# intial velocity condition
def g(x):
  return 1
# initially at rest
#def g(x):
#    return 0

# set up initial displacement condition on x
for i in range(lenx):
    u[i,0]=init_fn(i*dx)

# set up initial velocity condition on x
#for j in range(1,lenx-1):
#     u[j,1] = sigma*(u[j+1,0]-2*u[j,0]+u[j-1,0]) + u[j,0]
for j in range(1,lenx-1):
     u[j,1] = 0.5*(r**2*u[j+1,0]+2*(1-r**2)*u[j,0]+r**2*u[j-1,0]) + dt*g(j*dx)

# algorithm to numerically solve wave equation
#for n in range(1,lent-1):
#    for j in range(1,lenx-1):
#        u[j,n+1] = sigma*(u[j+1,n] - 2*u[j,n] + u[j-1,n]) + 2*u[j,n] - u[j,n-1]

for n in range(1,lent-1):
    for j in range(1,lenx-1):
        u[j,n+1] = r**2*u[j+1,n] + 2*(1-r**2)*u[j,n] + r**2*u[j-1,n] - u[j,n-1]

fig, ax = plt.subplots()

line, = ax.plot(x.transpose(), u[:,0])
line.set_ydata([nan] * lenx)
ax.set_xlabel("x")
ax.set_ylabel("u(x,t)")

def init():  # only required for blitting to give a clean slate.
    line.set_ydata([nan] * lenx)
    ax.set_xlabel("x")
    ax.set_ylabel("u(x,t)")
```

```
    ax.set_ylim(-1,1)
    return line,

def animate(i):
    line.set_ydata(u[:,i])  # update the data.
    ax.set_ylim(-1,1)
    return line,

ani = animation.FuncAnimation(
    fig, animate, init_func=init, interval=20, blit=True, save_count=142)
ani.save('pulse.gif', writer='imagemagick', fps=60)
#plt.show()
```

2D Wave FTCS Method

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 24 15:34:57 2019

@author: isabellapestovski
"""
from numpy import zeros,array,linspace,meshgrid
from math import sqrt,sin,exp,pi,cos
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.animation as animation

sigma = 0.5
dx = 0.01
dy = dx
dt = sqrt(dx*dx*sigma)

lenx = int(1/dx)+1
leny = int(1/dy)+1
lent = int(1/dt)+1

t = array([linspace(0,1,lent)])
x = array([linspace(0,1,lenx)])
y = array([linspace(0,1,leny)])

u = zeros([lenx,leny,lent])

# wave split propagation
#def init_fn(x,y):
#    return -cos(3*pi*x)*sin(pi*x)*-cos(3*pi*y)*sin(pi*y)
# standing wave
#def init_fn(x,y):
#    return sin(3*pi*x)*sin(3*pi*y)
# ripple propagation
def init_fn(x,y):
    return exp(-(2*pi*x)**2)*sin(2*pi*x)*exp(-(2*pi*y)**2)*sin(2*pi*y)

for i in range(lenx):
```

```
    for j in range(leny):
        u[i,j,0] = init_fn(i*dx,j*dy)

for i in range(1,lenx-1):
    for j in range(1,leny-1):
        u[i,j,1] = sigma*(u[i+1,j,0]-2*u[i,j,0]+u[i-1,j,0]) +\
        sigma*(u[i,j+1,0]-2*u[i,j,0]+u[i,j-1,0]) + u[i,j,0]

# algorithm to numerically solve wave equation
for n in range(1,lent-1):
    for i in range(1,lenx-1):
        for j in range(1,leny-1):
            u[i,j,n+1] = sigma*(u[i+1,j,n] - 2*u[i,j,n] + u[i-1,j,n]) +\
            sigma*(u[i,j+1,n]-2*u[i,j,n]+u[i,j-1,n]) + 2*u[i,j,n] - u[i,j,n-1]

fig = plt.figure()
ax = fig.gca(projection='3d')
X, Y = meshgrid(x.transpose(),y.transpose())
Z = u[:,:,0]
plot = Axes3D.plot_surface(ax,X,Y,Z)
ax.set_zlim3d(-1,1)
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")

def update_plot(i,u,plot):
    ax.clear()
    plot = Axes3D.plot_surface(ax,X,Y,u[:,:,i], cmap="magma")
    ax.set_zlim3d(-1,1)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("z")
    return plot

ani = animation.FuncAnimation(fig,update_plot,fargs=(u, plot))
ani.save('ripple2d.gif', writer='imagemagick', fps=60)
```

1D GUI

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
from matplotlib.figure import Figure
from numpy import zeros,array,linspace,nan
from math import sin,pi,sqrt,cos,exp
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from PIL import Image, ImageTk
from itertools import count, cycle

import tkinter as tk
from tkinter import ttk

LARGE_FONT= ("Verdana", 12)

class ImageLabel(tk.Label):
    """
    A Label that displays images, and plays them if they are gifs
    :im: A PIL Image instance or a string filename
    """
    def load(self, im):
        if isinstance(im, str):
            im = Image.open(im)
        frames = []

        try:
            for i in count(1):
                frames.append(ImageTk.PhotoImage(im.copy()))
                im.seek(i)
        except EOFError:
            pass
        self.frames = cycle(frames)

        try:
            self.delay = im.info['duration']
        except:
            self.delay = 100

        if len(frames) == 1:
```

```
            self.config(image=next(self.frames))
        else:
            self.next_frame()

    def unload(self):
        self.config(image=None)
        self.frames = None

    def next_frame(self):
        if self.frames:
            self.config(image=next(self.frames))
            self.after(self.delay, self.next_frame)

class OneDWave(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "1D Wave Simulator")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand = True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, PageOne, PageTwo, PageThree):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()

class StartPage(tk.Frame):
```

```
    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        label = tk.Label(self, text="Start Page", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Single Pulse",
                            command=lambda: controller.show_frame(PageOne))
        button1.pack()

        button2 = ttk.Button(self, text="Standing Wave",
                            command=lambda: controller.show_frame(PageTwo))
        button2.pack()

        button3 = ttk.Button(self, text="Travelling Wave",
                            command=lambda: controller.show_frame(PageThree))
        button3.pack()

class PageOne(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Single Pulse Animation", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
                            command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('pulse.gif')

class PageTwo(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Standing Wave Animation",
font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
```

```
                    command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('standing.gif')

class PageThree(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Travelling Wave Animation",
font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
                    command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('traveling.gif')

app = OneDWave()
app.mainloop()
```

2D GUI

```
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg,
NavigationToolbar2Tk
from matplotlib.figure import Figure
from numpy import zeros,array,linspace,nan
from math import sin,pi,sqrt,cos,exp
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from PIL import Image, ImageTk
from itertools import count, cycle

import tkinter as tk
from tkinter import ttk

LARGE_FONT= ("Verdana", 12)

class ImageLabel(tk.Label):
    """
    A Label that displays images, and plays them if they are gifs
    :im: A PIL Image instance or a string filename
    """
    def load(self, im):
        if isinstance(im, str):
            im = Image.open(im)
        frames = []

        try:
            for i in count(1):
                frames.append(ImageTk.PhotoImage(im.copy()))
                im.seek(i)
        except EOFError:
            pass
        self.frames = cycle(frames)

        try:
            self.delay = im.info['duration']
        except:
            self.delay = 100

        if len(frames) == 1:
```

```
            self.config(image=next(self.frames))
        else:
            self.next_frame()

    def unload(self):
        self.config(image=None)
        self.frames = None

    def next_frame(self):
        if self.frames:
            self.config(image=next(self.frames))
            self.after(self.delay, self.next_frame)

class TwoDWave(tk.Tk):
    def __init__(self, *args, **kwargs):
        tk.Tk.__init__(self, *args, **kwargs)
        tk.Tk.wm_title(self, "2D Wave Simulator")

        container = tk.Frame(self)
        container.pack(side="top", fill="both", expand = True)
        container.grid_rowconfigure(0, weight=1)
        container.grid_columnconfigure(0, weight=1)

        self.frames = {}

        for F in (StartPage, PageOne, PageTwo, PageThree):

            frame = F(container, self)

            self.frames[F] = frame

            frame.grid(row=0, column=0, sticky="nsew")

        self.show_frame(StartPage)

    def show_frame(self, cont):
        frame = self.frames[cont]
        frame.tkraise()

class StartPage(tk.Frame):
```

```
    def __init__(self, parent, controller):
        tk.Frame.__init__(self,parent)
        label = tk.Label(self, text="Start Page", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Standing Wave",
                            command=lambda: controller.show_frame(PageOne))
        button1.pack()

        button2 = ttk.Button(self, text="Ripple Wave",
                            command=lambda: controller.show_frame(PageTwo))
        button2.pack()

        button3 = ttk.Button(self, text="Split Propagation",
                            command=lambda: controller.show_frame(PageThree))
        button3.pack()

class PageOne(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Standing Wave Animation",
font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
                            command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('standing2d.gif')

class PageTwo(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Ripple Animation", font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
```

```
                    command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('ripple2d.gif')

class PageThree(tk.Frame):

    def __init__(self, parent, controller):
        tk.Frame.__init__(self, parent)
        label = tk.Label(self, text="Split Propagation Animation",
font=LARGE_FONT)
        label.pack(pady=10,padx=10)

        button1 = ttk.Button(self, text="Back to Home",
                    command=lambda: controller.show_frame(StartPage))
        button1.pack()

        lbl = ImageLabel(self)
        lbl.pack()
        lbl.load('split2d.gif')

app = TwoDWave()
app.mainloop()
```