Steven Wray

Physics 4610

4/9/12

**The Advection Equation**

***Introduction***

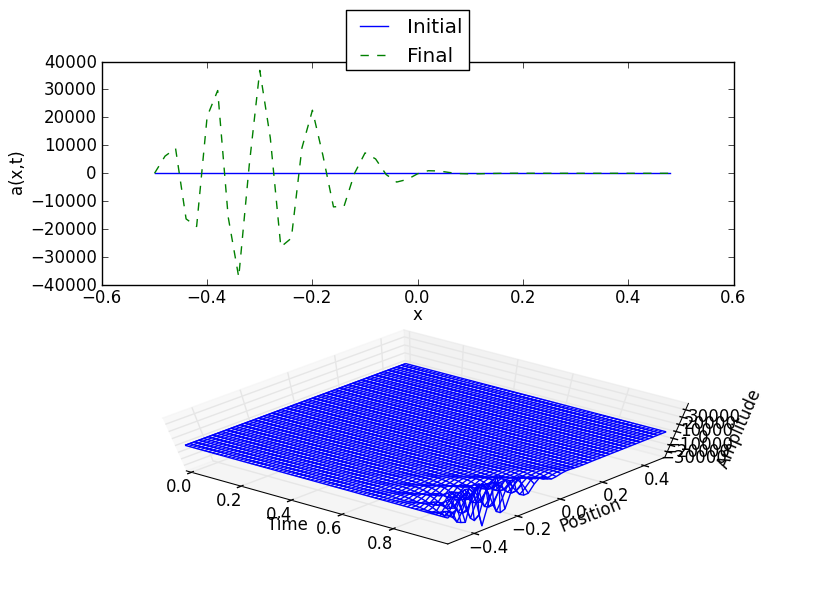
This project used numerical methods to solve the partial differential equation describing advection. The forward time centered space (FTCS) scheme, Lax scheme, Lax-Wendroff scheme, upwind scheme and leap-frog scheme were evaluated and compared.

***Activities***

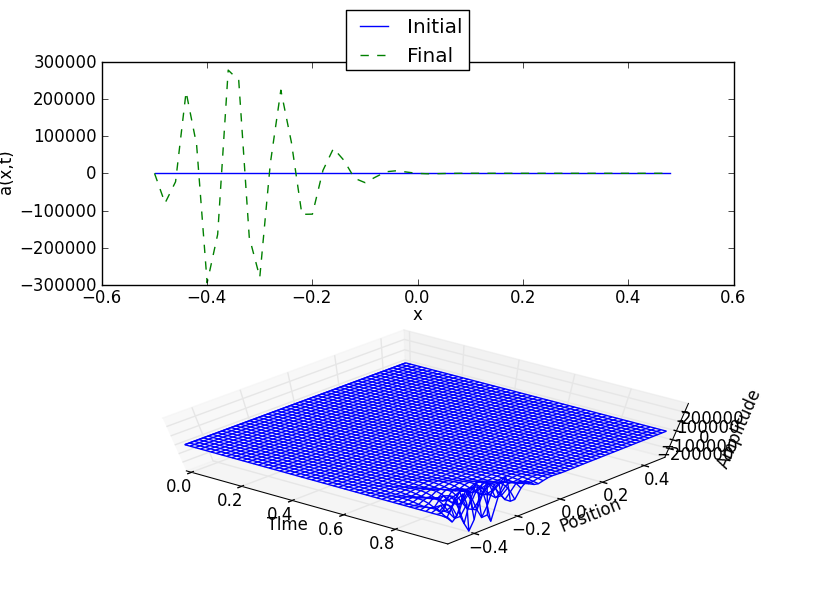
7.2: Modify **advect** to use the Dirichlet boundary conditions:

Run the program so that the wave generated at reaches the other side of the system. Use grid points, a frequency , and time steps .

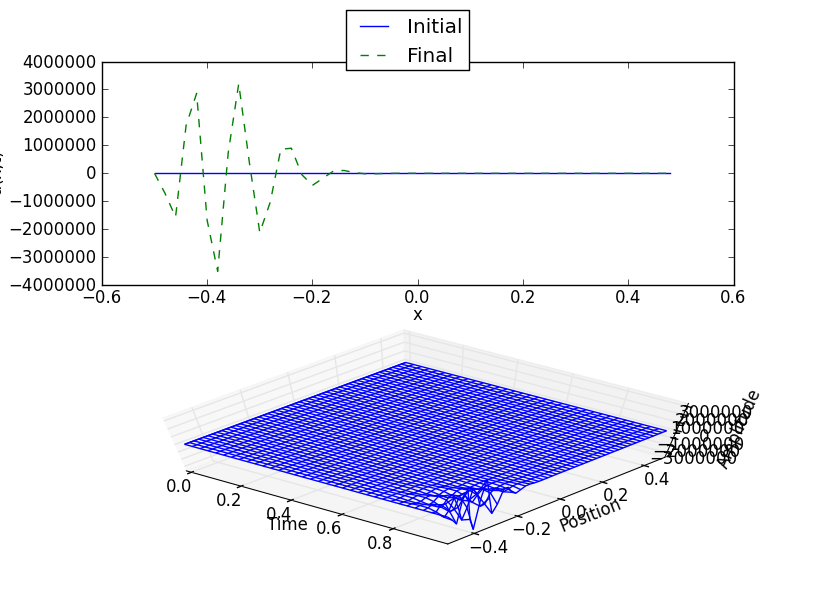
Using the FTCS scheme with gives:



With :

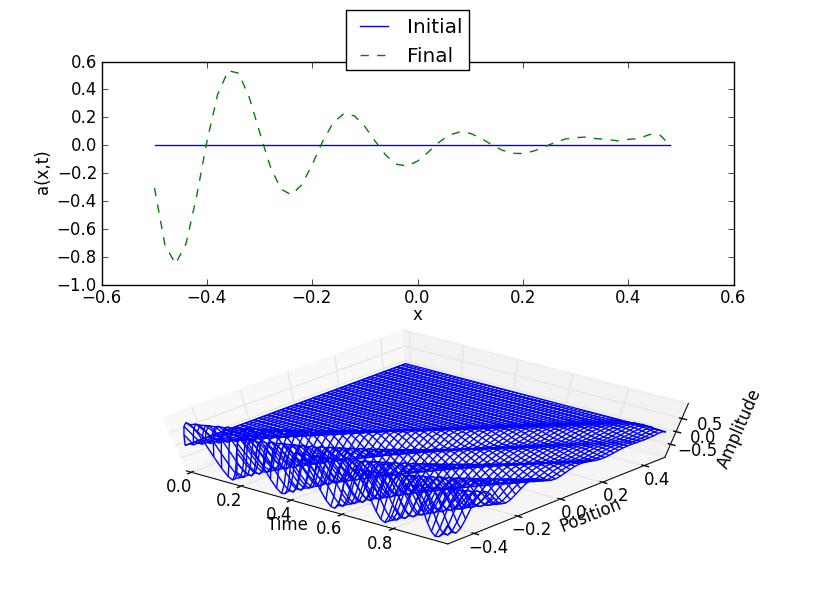


And with :

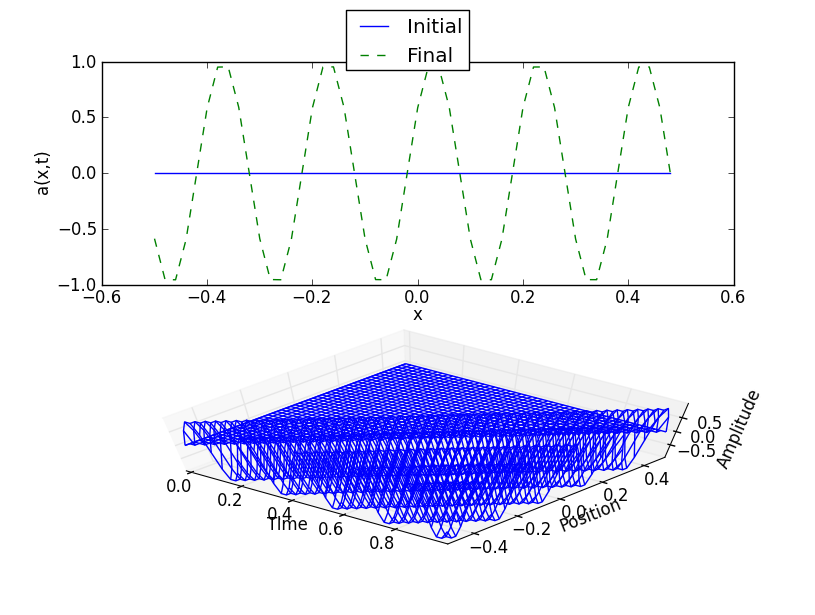


For all three values of , the FTCS method is unstable. The amplitude of the driving force is only 31.42, but the model produces waves with amplitudes greater than .

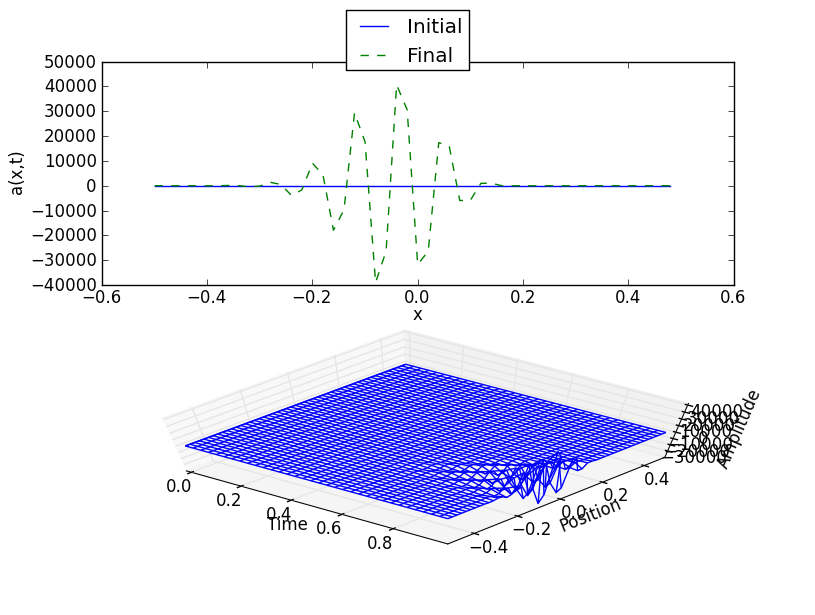
Using the Lax scheme with :



For :

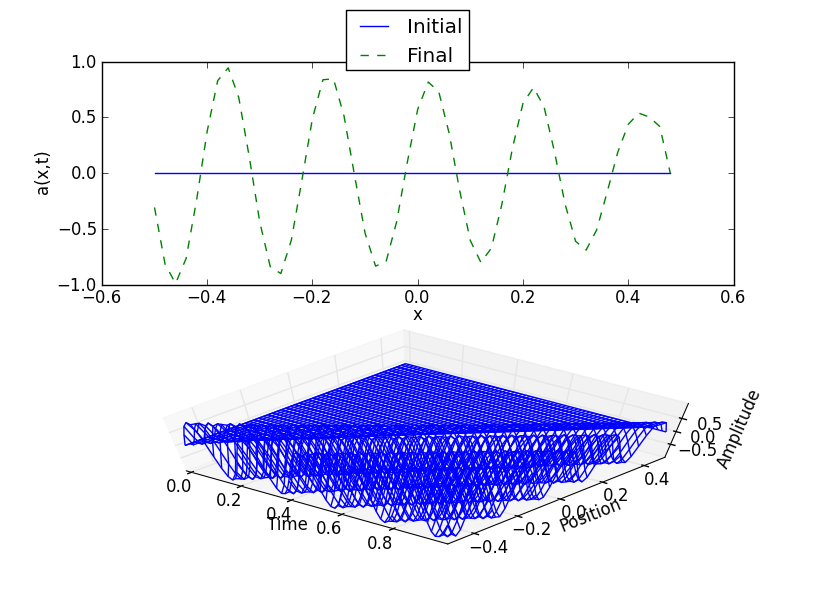


And for :

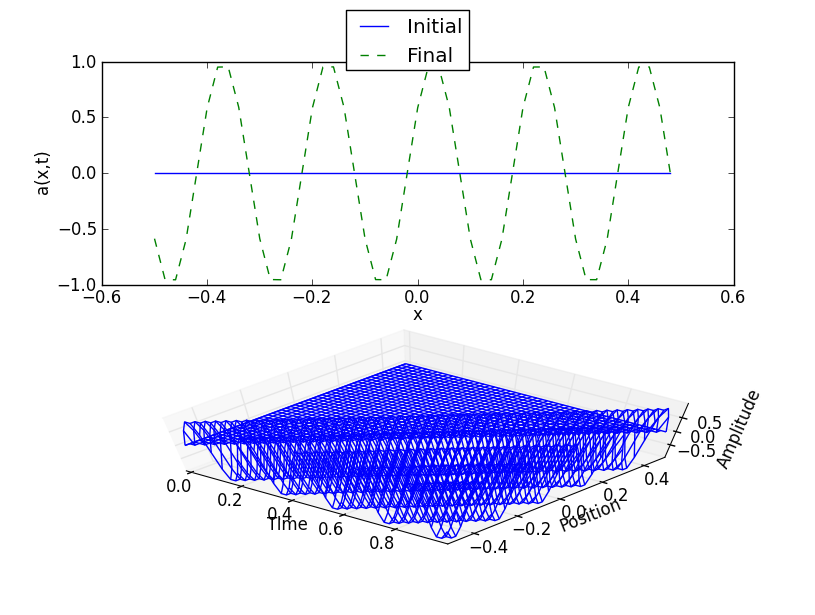


The Lax method is stable for , but the wave dies out quickly as it moves through space. When , the method is stable and accurate. The Lax method is unstable for .

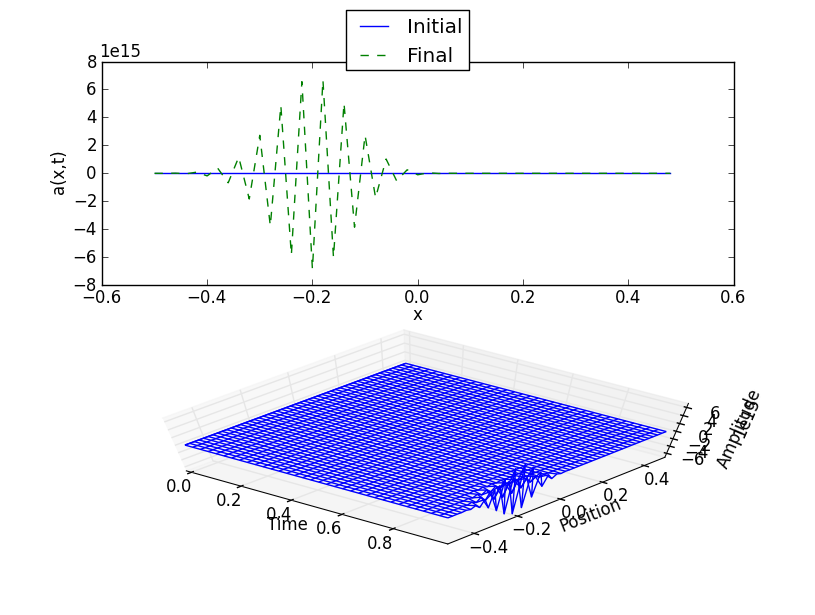
Using the Lax-Wendroff scheme with :



With :



And with :



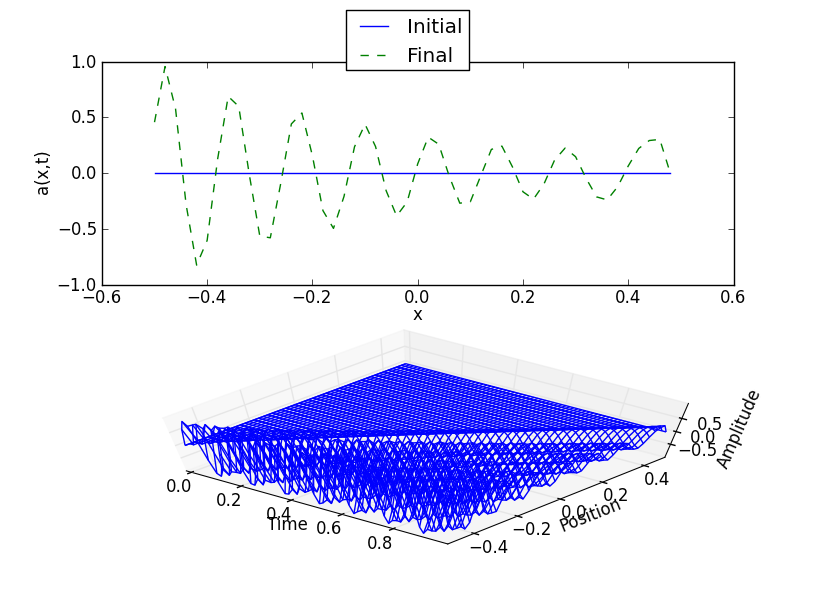
The Lax-Wendroff scheme is stable for . Although the wave is damped for , the effect is much less noticeable than under the Lax scheme. The Lax-Wendroff scheme is unstable for .

In general, the FTCS method is unstable for all time steps. For the other two schemes, the stability is determined by the Courant-Friedrichs-Lewy (CFL) condition , where is the grid spacing and is the wave speed. If the time step , the result will be stable. If

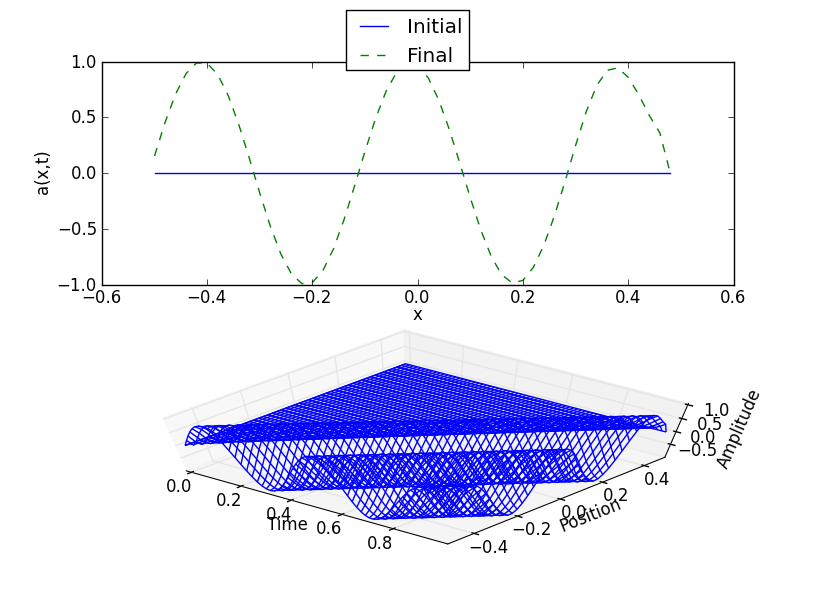
, the result is unstable.

How do the results change when you vary the frequency?

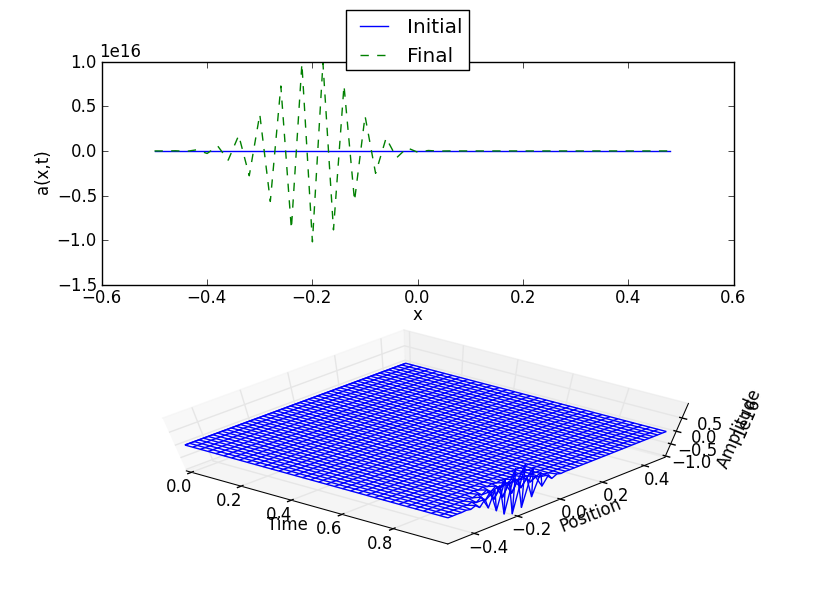
Since the CFL condition does not depend on frequency, the stability of the results will be the same regardless of which frequency is chosen. For example, using the Lax-Wendroff scheme with and gives:



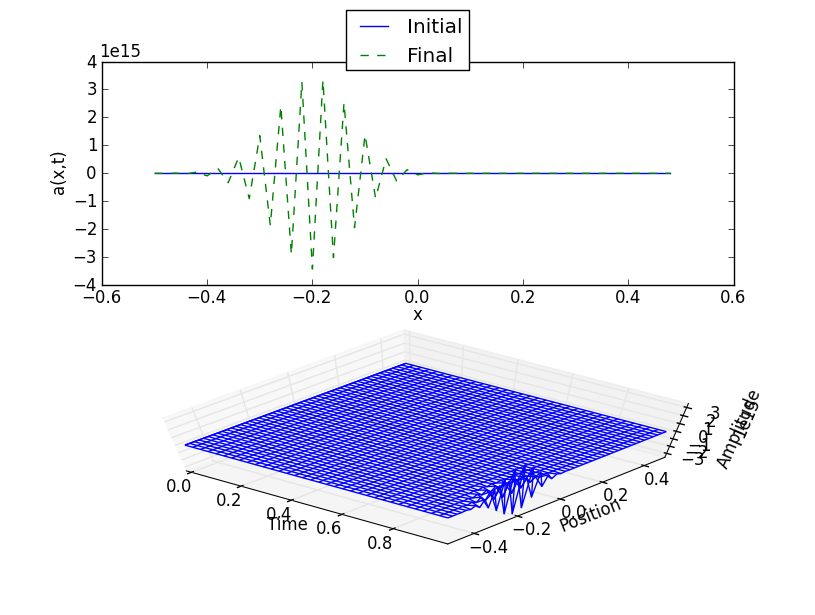
Using the same time step with :



In both cases, the solution is stable. Both waves show damping as they move through space since . Using the Lax-Wendroff scheme with time step and gives:



The same time step with :



Since the time step , the scheme is unstable for any choice of frequency.

7.3: The combination of advection and diffusion is described by the transport equation

Find the solution of this equation for the initial condition with periodic boundary conditions at .

We know from chapter that the Gaussian

is one solution of the diffusion equation

This suggests that one solution of the transport equation is a Gaussian that is moving in the positive direction with velocity :

We can verify that this is a solution of the transport equation by evaluating the partials:

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

And so

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Therefore the moving Gaussian is a solution of the transport equation.

The general solution for the transport equation with periodic boundary conditions can be obtained by applying the method of images to this moving Gaussian function. Suppose the advection equation is defined on a one dimensional region of length with an initial delta spike at . Then we know from using the method of images (previously derived in problem 6.7) that the solution which satisfies periodic boundary conditions is:

7.4: Write a program using the FTCS scheme to solve the one-dimensional transport equation from problem 7.3.

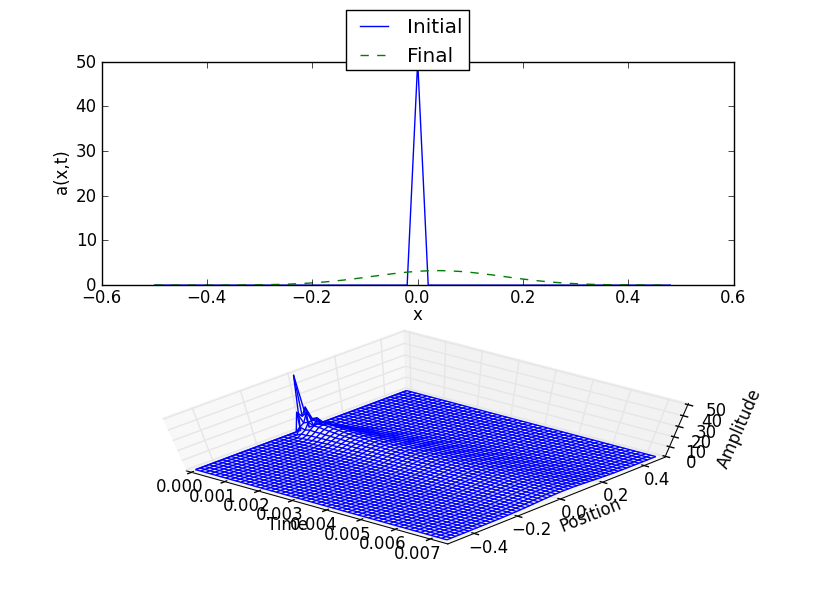
1. Show that the numerical solution is stable if

Replacing the partial derivatives in the transport equation with the FTCS discretization gives the formula:

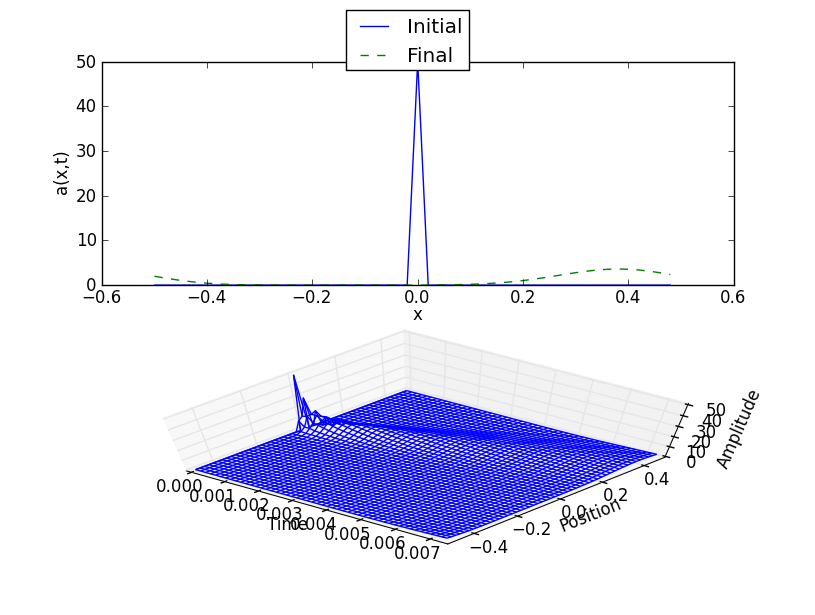
and solving for the future step gives:

Let . Then

So the solution should be stable. Plotting the first iterations of the system gives:



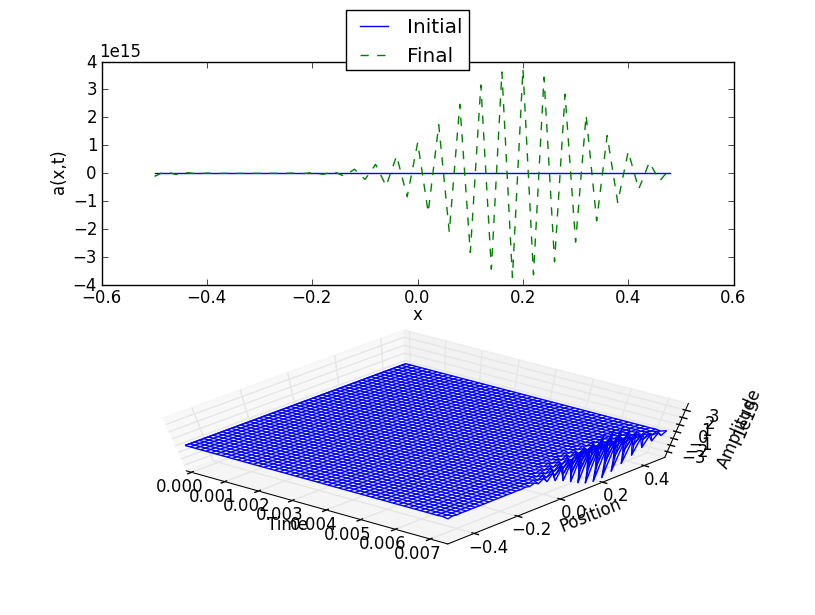
Increasing the speed to and keeping all other parameters the same gives:



In both cases the solution is stable.

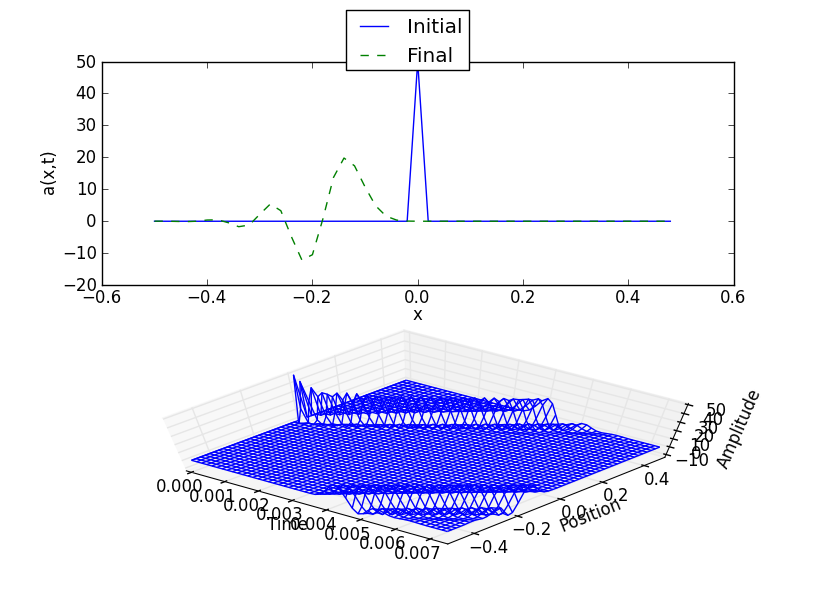
Let . Then

So the solution should be unstable. The first iterations are:



Another unstable solution comes from the parameters and . In this case

The first iterations of this system are:



It is unstable because the shape of the initial spike is not preserved. These examples show that the FTCS scheme gives a stable solution to the transport equation only if both parts of the given criterion are satisfied.

1. Compare the results to the analytical solution from problem 6.4.

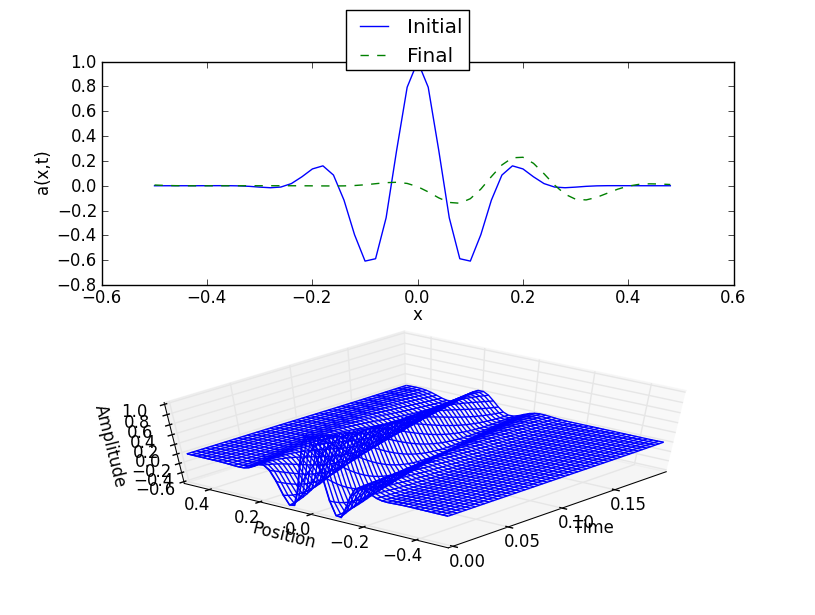
The stable numerical solutions agree with the analytical solution. Both cases show an initial delta spike that flattens out over time while the peak of the spike moves through space with speed .

7.5: The upwind scheme for the advection equation uses the discretization

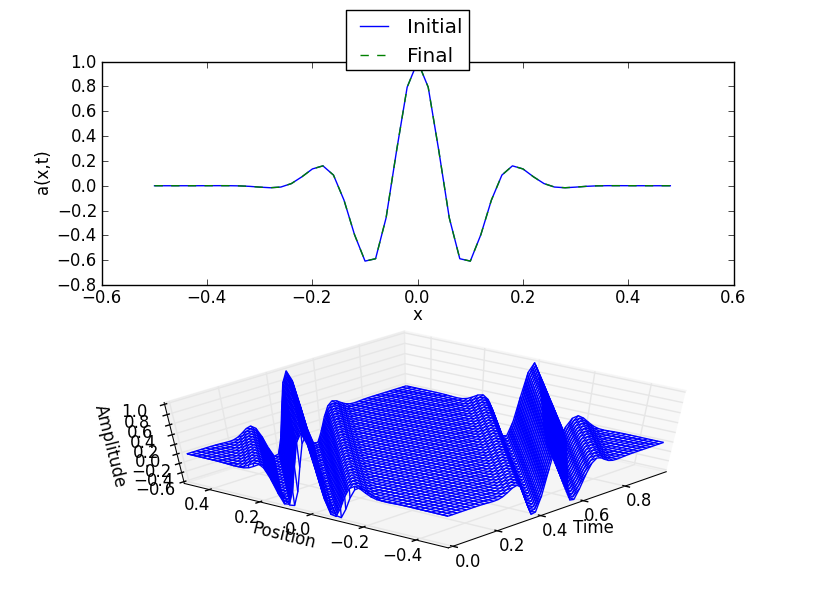
Modify the **advect** program to use this scheme, and find the values of for which it is stable.

Solving the discretization above for the future step gives:

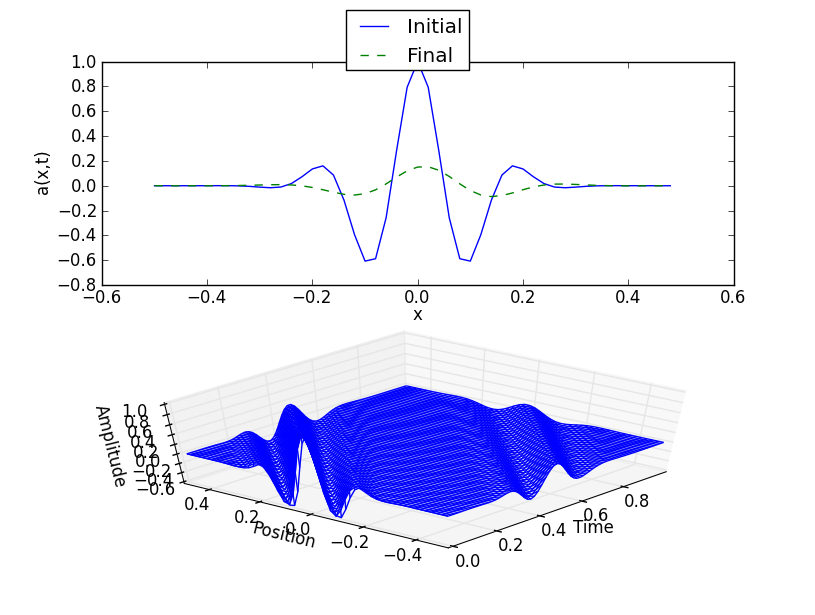
Using this scheme with , , and a Gaussian modified cosine function as the initial condition gives the result:



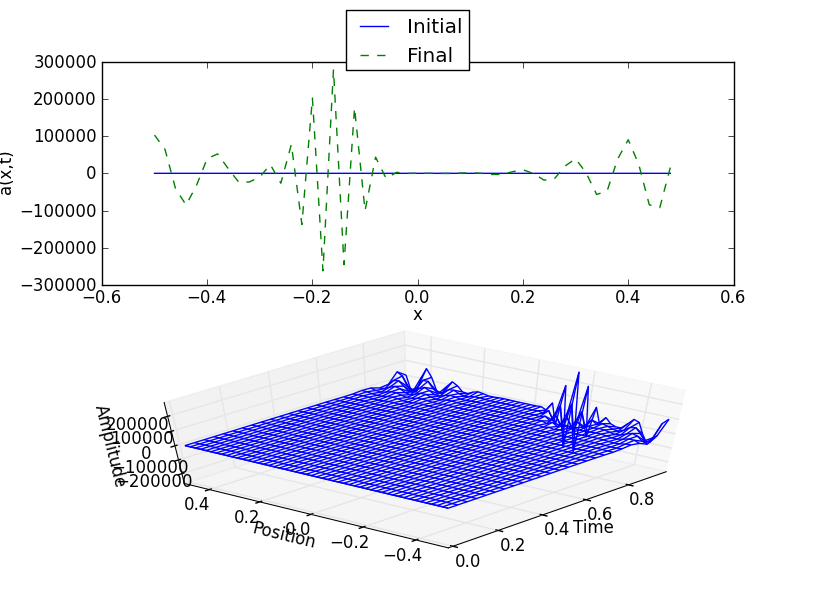
For and :



For and :



From these examples, it appears that the upwind scheme is stable whenever . If is strictly less than , then the initial pulse dies out over time. To analyze the stability of the case , consider the result for and :



The upwind scheme is not stable for .

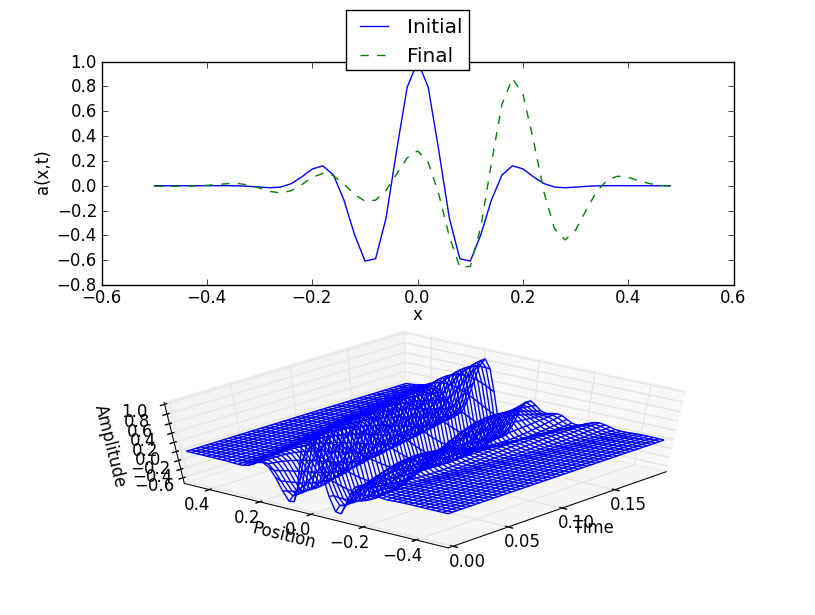
7.6: The leap-frog scheme uses a centered discretization for both sides of the equation:

Modify the **advect** program to use this scheme, and find the values of for which it is stable.

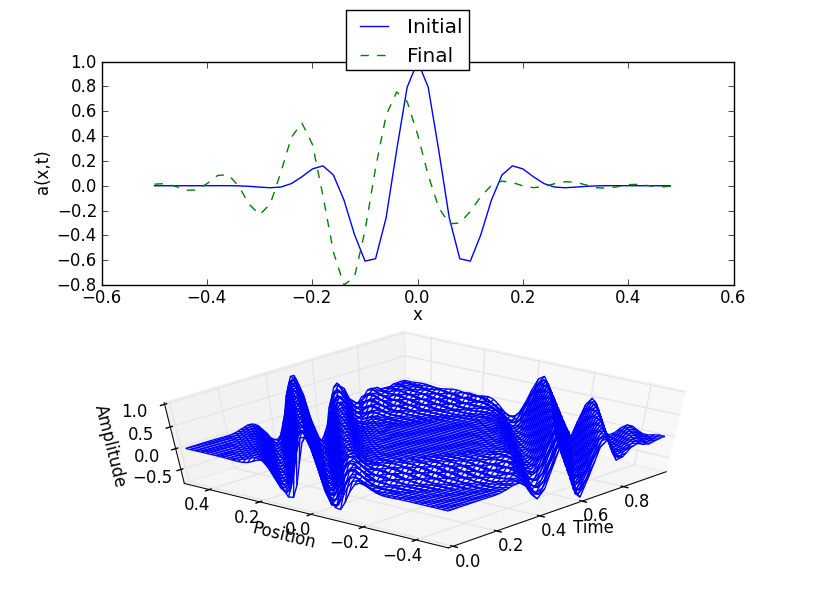
Solving the equation above for the future step yields:

This scheme uses two earlier time steps, so I have used the Lax scheme to calculate the first iteration.

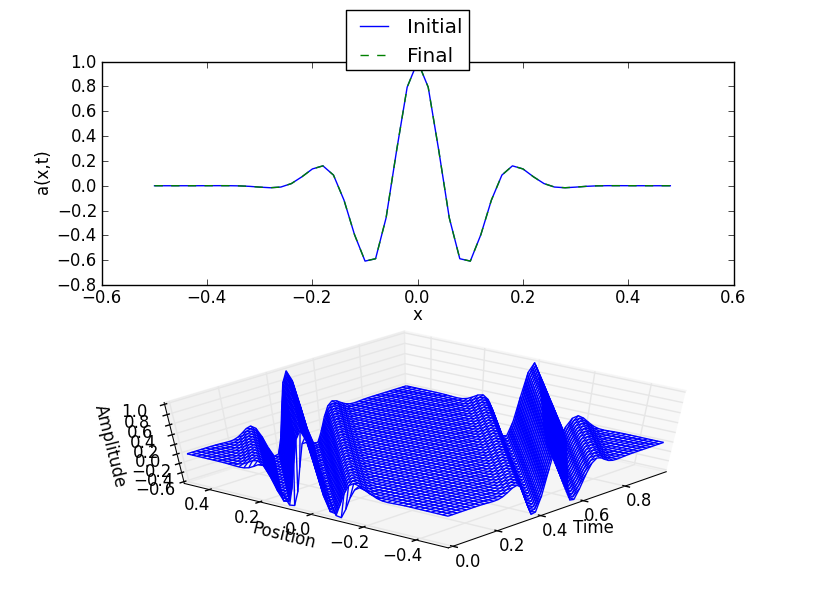
Using the leap-frog scheme with , and a Gaussian modified cosine function for the initial condition gives:



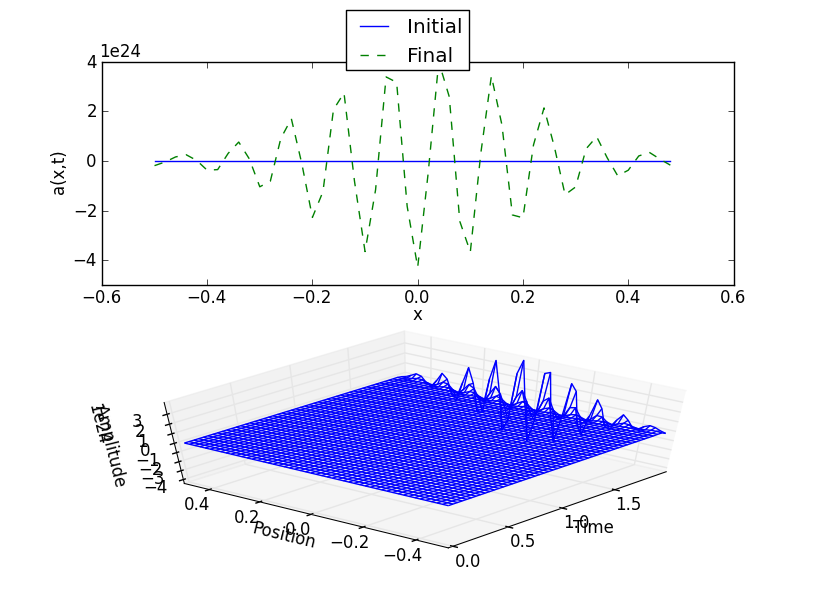
For and :



In both of these graphs the initial shape of the pulse is distorted, so the solution is not stable. When and :



Finally, for the case consider and :



From the examples it appears that the leap-frog method is only stable when . However, for , the leap-frog scheme may provide a better approximation that the Lax method. The leap-frog scheme does not die out as fast and its instabilities accumulate slowly. The method does not give a good approximation for .

**Conclusions**

The advection equation describes the movement of a conserved quantity through a medium. It is described by the PDE

There are several schemes to solve this equation. Although it is easiest to describe, the FTCS scheme is unstable for almost all choices of time step. The Lax method and the upwind method are stable when . However, these methods die off relatively quickly when is strictly less than . The

Lax-Wendroff and leap-frog methods are best for this case.

**APPENDIX:**

**Source Code**

# Python 2.7.2

# Steven Wray

# Physics 4610/HW5

# Exercise 7.2

# Program to solve the advection equation with different

# hyperbolic schemes, using Dirichlet boundary conditions.

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import axes3d

# Constants that represent the different numerical schemes

FTCS = 1

LAX = 2

LAX\_WENDROFF = 3

# Constants that represent different initial conditions

MODIFIED\_GAUSSIAN = 1

CONSTANT = 2

def main():

# Initialize parameters

print("Choose a numerical method.")

print("\t1. FTCS")

print("\t2. Lax")

print("\t3. Lax-Wendroff")

s = input("Enter a method: ")

method = int(s)

s = input("Enter number of grid points (N): ")

N = int(s)

L = 1.0 # System size

h = L/N # Grid spacing

c = 1.0 # Wave speed

print("Time for wave to move one grid spacing is " + \

str(h/c) + ".")

s = input("Enter time step: ")

tau = float(s)

coeff = -1 \* c \* tau / (2.0 \* h) # Coefficient used by all schemes

coefflw = 2 \* coeff\*\*2 # Coefficient used by LW scheme

print("Wave moves across system in " + \

str(L / (c\*tau)) + " steps.")

s = input("Enter number of steps: ")

nStep = int(s)

s = input("Enter angular freqeuncy of oscillation at left boundary: ")

freq = float(s)

print("Choose an initial condition.")

print("\t1. Gaussian modified cosine")

print("\t2. Constant (0 amplitude)")

s = input("Enter a number: ")

initial\_condition = int(s)

# Set ICs and BCs

sigma = 0.1 # Width of the Gaussian pulse

k\_wave = math.pi / sigma # Wave number of the cosine

x = np.array(range(N))

x = x \* h - L/2 # Coordinates of grid points

if initial\_condition == MODIFIED\_GAUSSIAN:

# IC is a Gaussian cosine pulse

sigma = 0.1 # Width of the Gaussian pulse

k\_wave = math.pi / sigma # Wave number of the cosine

a = np.copy(x)

a = np.cos(k\_wave \* a) \* np.exp(-1 \* a\*\*2 / (2 \* sigma\*\*2))

else:

# IC is constant

a = np.zeros(N)

# Set Dirichlet boundary conditions at t=0

a[0] = math.sin(freq \* 0)

a[-1] = 0

ip = np.array(range(1, N + 1)) # ip = i+1 (reflects periodic BCs)

ip[-1] = 0

im = np.array(range(-1, N - 1)) # im = i-1 (reflects periodic BCs)

im[0] = N - 1

# Initialize plotting variables

iplot = 1 # Plot counter

aplot = []

aplot.append(np.copy(a)) # Record the initial state

tplot = [0] # Record the initial time

nplots = 50 # Desired number of plots

plotStep = nStep / nplots # Steps between plots

if plotStep == 0:

plotStep = 1 # Avoid possible division by zero

# Loop over desired number of steps

for iStep in range(nStep):

# Compute new values of wave amplitude

if method == FTCS:

a = a + coeff \* (a[ip] - a[im])

# As written, the scheme reflects periodic boundary conditions.

# Set endpoints to reflect Dirichlet BCs.

a[0] = math.sin(freq \* tau \* iStep)

a[-1] = 0

elif method == LAX:

a = 0.5 \* (a[ip] + a[im]) + coeff \* (a[ip] - a[im])

# As written, the scheme reflects periodic boundary conditions.

# Set endpoints to reflect Dirichlet BCs

a[0] = math.sin(freq \* tau \* iStep)

a[-1] = 0

else: # Using Lax-Wendroff

a = a + coeff \* (a[ip] - a[im]) + \

coefflw \* (a[ip] + a[im] - 2 \* a)

# As written, the scheme reflects periodic boundary conditions.

# Set endpoints to reflect Dirichlet BCs

a[0] = math.sin(freq \* tau \* iStep)

a[-1] = 0

# Periodically record state for plotting

if iStep % plotStep == 0:

iplot = iplot + 1

aplot.append(np.copy(a))

tplot.append(tau \* iStep)

# Create lists for wire mesh plot

X = np.zeros( (len(tplot),len(x)) )

Y = np.zeros( (len(tplot),len(x)) )

Z = np.zeros( (len(tplot),len(x)) )

for i in range(len(tplot)):

for j in range(len(x)):

X[i,j] = tplot[i]

Y[i,j] = x[j]

Z[i,j] = aplot[i][j]

# Wire mesh plot

fig = plt.figure()

ax = fig.add\_subplot(212, projection='3d')

ax.plot\_wireframe(X, Y, Z)

ax.set\_xlabel('Time')

ax.set\_ylabel('Position')

ax.set\_zlabel('Amplitude')

ax.view\_init(elev = 65, azim = -50)

# Plot initial and final states

plt.subplot(211)

line1 = plt.plot(x, aplot[0], 'b')

line2 = plt.plot(x, a, 'g--')

plt.ylabel('a(x,t)')

plt.xlabel('x')

plt.figlegend((line1, line2), ('Initial', 'Final'), 'upper center')

plt.show()

main()

# Python 2.7.4

# Steven Wray

# Physics 4610/HW5

# Exercise 7.4

# Program to solve the transport equation with the

# FTCS scheme, using periodic boundary conditions.

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import axes3d

def main():

# Initialize parameters

s = input("Enter number of grid points: ")

N = int(s)

L = 1.0 # System size

h = L/N # Grid spacing

s = input("Enter diffusion coefficient (kappa): ")

kappa = float(s)

s = input("Enter wave speed (c): ")

c = float(s)

print("Time for wave to move one grid spacing is " + \

str(h/c) + ".")

s = input("Enter time step: ")

tau = float(s)

print("Wave circles system in " + \

str(L / (c\*tau)) + " steps.")

s = input("Enter number of steps: ")

nStep = int(s)

if (c \* tau / h)\*\*2 <= 2 \* kappa \* tau / h\*\*2 and \

2 \* kappa \* tau / h\*\*2 <= 1:

print("Solution expected to be stable.")

else:

print("WARNING: Solution expected to be unstable.")

x = np.array(range(N))

x = x \* h - L/2 # Coordinates of grid points

ip = np.array(range(1, N + 1)) # ip = i+1 (reflects periodic BCs)

ip[-1] = 0

im = np.array(range(-1, N - 1)) # im = i-1 (reflects periodic BCs)

im[0] = N - 1

# Set delta spike as IC

a = np.zeros(N)

a[int(round(N/2))] = 1/h

# Coefficients used by FTCS scheme

coeff1 = tau \* (-1 \* c /(2 \* h) + kappa / h\*\*2)

coeff2 = tau \* (c /(2 \* h) + kappa / h\*\*2)

coeff3 = 1 - 2 \* tau \* kappa / h\*\*2

# Initialize plotting variables

iplot = 1 # Plot counter

aplot = []

aplot.append(np.copy(a)) # Record the initial state

tplot = [0] # Record the initial time

nplots = 50 # Desired number of plots

plotStep = nStep / nplots # Steps between plots

if plotStep == 0:

plotStep = 1 # Avoid possible division by zero

# Loop over desired number of steps

for iStep in range(nStep):

# Compute new values of wave amplitude

a = coeff1 \* a[ip] + coeff2 \* a[im] + coeff3 \* a

# Periodically record state for plotting

if iStep % plotStep == 0:

iplot = iplot + 1

aplot.append(np.copy(a))

tplot.append(tau \* iStep)

# Create lists for wire mesh plot

X = np.zeros( (len(tplot),len(x)) )

Y = np.zeros( (len(tplot),len(x)) )

Z = np.zeros( (len(tplot),len(x)) )

for i in range(len(tplot)):

for j in range(len(x)):

X[i,j] = tplot[i]

Y[i,j] = x[j]

Z[i,j] = aplot[i][j]

# Wire mesh plot

fig = plt.figure()

ax = fig.add\_subplot(212, projection='3d')

ax.plot\_wireframe(X, Y, Z)

ax.set\_xlabel('Time')

ax.set\_ylabel('Position')

ax.set\_zlabel('Amplitude')

ax.view\_init(elev = 65, azim = -50)

# Plot initial and final states

plt.subplot(211)

line1 = plt.plot(x, aplot[0], 'b')

line2 = plt.plot(x, a, 'g--')

plt.ylabel('a(x,t)')

plt.xlabel('x')

plt.figlegend((line1, line2), ('Initial', 'Final'), 'upper center')

plt.show()

main()

# Python 2.7.2

# Steven Wray

# Physics 4610/HW5

# Exercise 7.5

# Program to solve the advection equation with the upwind

# hyperbolic schemes.

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import axes3d

def main():

# Initialize parameters

print("Using the upwind scheme.")

s = input("Enter number of grid points: ")

N = int(s)

L = 1.0 # System size

h = L/N # Grid spacing

c = 1.0 # Wave speed

print("Time for wave to move one grid spacing is " + \

str(h/c) + ".")

s = input("Enter time step: ")

tau = float(s)

coeff = tau \* c / h # Coefficient used by upwind scheme

print("Wave circles system in " + \

str(L / (c\*tau)) + " steps.")

s = input("Enter number of steps: ")

nStep = int(s)

# Set ICs and BCs

sigma = 0.1 # Width of the Gaussian pulse

k\_wave = math.pi / sigma # Wave number of the cosine

x = np.array(range(N))

x = x \* h - L/2 # Coordinates of grid points

# IC is a Gaussian cosine pulse

a = np.copy(x)

a = np.cos(k\_wave \* a) \* np.exp(-1 \* a\*\*2 / (2 \* sigma\*\*2))

# Use periodic boundary conditions

ip = np.array(range(1, N + 1)) # ip = i+1 with periodic BCs

ip[-1] = 0

im = np.array(range(-1, N - 1)) # im = i-1 with periodic BCs

im[0] = N - 1

# Initialize plotting variables

iplot = 1 # Plot counter

aplot = []

aplot.append(np.copy(a)) # Record the initial state

tplot = [0] # Record the initial time

nplots = 50 # Desired number of plots

plotStep = nStep / nplots # Steps between plots

if plotStep == 0:

plotStep = 1 # Avoid possible division by zero

# Loop over desired number of steps

for iStep in range(nStep):

# Compute new values of wave amplitude

# using upwind method

a = (1 - coeff) \* a + coeff \* a[im]

# Periodically record state for plotting

if iStep % plotStep == 0:

iplot = iplot + 1

aplot.append(np.copy(a))

tplot.append(tau \* iStep)

# Create lists for wire mesh plot

X = np.zeros( (len(tplot),len(x)) )

Y = np.zeros( (len(tplot),len(x)) )

Z = np.zeros( (len(tplot),len(x)) )

for i in range(len(tplot)):

for j in range(len(x)):

X[i,j] = tplot[i]

Y[i,j] = x[j]

Z[i,j] = aplot[i][j]

# Wire mesh plot

fig = plt.figure()

ax = fig.add\_subplot(212, projection='3d')

ax.plot\_wireframe(X, Y, Z)

ax.set\_xlabel('Time')

ax.set\_ylabel('Position')

ax.set\_zlabel('Amplitude')

ax.view\_init(elev = 50, azim = -140)

# Plot initial and final states

plt.subplot(211)

line1 = plt.plot(x, aplot[0], 'b')

line2 = plt.plot(x, a, 'g--')

plt.ylabel('a(x,t)')

plt.xlabel('x')

plt.figlegend((line1, line2), ('Initial', 'Final'), 'upper center')

plt.show()

main()

# Python 2.7.2

# Steven Wray

# Physics 4610/HW5

# Exercise 7.6

# Program to solve the advection equation with the

# leap-frog scheme.

import math

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import axes3d

def main():

# Initialize parameters

print("Using leap-frog scheme to model system.")

s = input("Enter number of grid points: ")

N = int(s)

L = 1.0 # System size

h = L/N # Grid spacing

c = 1.0 # Wave speed

print("Time for wave to move one grid spacing is " + \

str(h/c) + ".")

s = input("Enter time step: ")

tau = float(s)

coeff = -1 \* c \* tau / h # Coefficient used by both schemes

print("Wave circles system in " + \

str(L / (c\*tau)) + " steps.")

s = input("Enter number of steps: ")

nStep = int(s)

# Set ICs and BCs

sigma = 0.1 # Width of the Gaussian pulse

k\_wave = math.pi / sigma # Wave number of the cosine

x = np.array(range(N))

x = x \* h - L/2 # Coordinates of grid points

# IC is a Gaussian cosine pulse

a = np.copy(x)

a = np.cos(k\_wave \* a) \* np.exp(-1 \* a\*\*2 / (2 \* sigma\*\*2))

# Use periodic boundary conditions

ip = np.array(range(1, N + 1)) # ip = i+1 with periodic BCs

ip[-1] = 0

im = np.array(range(-1, N - 1)) # im = i-1 with periodic BCs

im[0] = N - 1

# Initialize plotting variables

iplot = 1 # Plot counter

aplot = []

aplot.append(np.copy(a)) # Record the initial state

tplot = [0] # Record the initial time

nplots = 50 # Desired number of plots

plotStep = nStep / nplots # Steps between plots

if plotStep == 0:

plotStep = 1 # Avoid possible division by zero

# Use Lax Scheme to calculate first iteration

a\_previous = np.copy(a)

a = 0.5 \* (a[ip] + a[im]) + 0.5 \* coeff \* (a[ip] - a[im])

# Loop over desired number of steps

for iStep in range(1, nStep):

# Compute new values of wave amplitude

a\_temp = np.copy(a)

a = coeff \* (a[ip] - a[im]) + a\_previous

a\_previous = np.copy(a\_temp)

# Periodically record state for plotting

if iStep % plotStep == 0:

iplot = iplot + 1

aplot.append(np.copy(a))

tplot.append(tau \* iStep)

# Create lists for wire mesh plot

X = np.zeros( (len(tplot),len(x)) )

Y = np.zeros( (len(tplot),len(x)) )

Z = np.zeros( (len(tplot),len(x)) )

for i in range(len(tplot)):

for j in range(len(x)):

X[i,j] = tplot[i]

Y[i,j] = x[j]

Z[i,j] = aplot[i][j]

# Wire mesh plot

fig = plt.figure()

ax = fig.add\_subplot(212, projection='3d')

ax.plot\_wireframe(X, Y, Z)

ax.set\_xlabel('Time')

ax.set\_ylabel('Position')

ax.set\_zlabel('Amplitude')

ax.view\_init(elev = 50, azim = -140)

# Plot initial and final states

plt.subplot(211)

line1 = plt.plot(x, aplot[0], 'b')

line2 = plt.plot(x, a, 'g--')

plt.ylabel('a(x,t)')

plt.xlabel('x')

plt.figlegend((line1, line2), ('Initial', 'Final'), 'upper center')

plt.show()

main()