Assignment 1 - Advanced Computational Science

Liam O'Sullivan – 10309537

March 10, 2014

Question 1

(1)

We are interested in solving the 1-d wave equation with $c = 1$,

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} \tag{1}$$

using central differences,

$$\frac{\partial^2 u}{\partial t^2} \approx \frac{\delta_t^2 u_j^n}{\tau^2}, \qquad \frac{\partial^2 u}{\partial x^2} \approx \frac{\delta_x^2 u_j^n}{h^2}$$

so our equation becomes

$$\frac{\delta_t^2 u_j^n}{\tau^2} = \frac{\delta_x^2 u_j^n}{h^2} \tag{2}$$

where τ is the time step size, and h is the x step size.

The operator δ_x^2, when applied to u_j^n, gives

$$\delta_x^2 = u_j^{n-1} - 2u_j^n + u_j^{n+1}$$

allowing us to expand out for both spatial and temporal derivatives and rearrange for u_j^{n+1}, giving

$$u_j^{n+1} = \nu^2 \left[u_{j-1}^n + u_{j+1}^n\right] - u_j^{n-1} + 2(1-\nu^2)u_j^n \tag{3}$$

with $\nu = \frac{\tau}{h}$.

A program was written in Python 3.3 to solve this problem with Dirichlet boundary conditions, $u(-7,t) = u(7,t) = 0$, on the intervals $x \in [-7,7]$, $t \in [0,14]$. The initial condition is given by $u(x,0) = e^{-x^2}$. The program is included in appendix A.

The solutions at various times are shown in figure 1.

(2)

Our expression for u_j^n using Central Differences is given in equation 3. To perform Fourier/von Neumann stability analysis, we take $u_j^n = \xi^n e^{ikhj}$ as a single Fourier mode solution, and see how our *amplification factor*, ξ, behaves as a function of τ, h, k. Substituting this into equation 3 gives

$$\xi^{n+1} e^{ikhj} = \nu^2 \left[\xi^n e^{ikh(j-1)} + \xi^n e^{ikh(j+1)}\right] - \xi^{n-1} e^{ikhj} + 2(1-\nu^2)\xi^n e^{ikhj}.$$

Dividing across by $\xi^{n-1} e^{ikhj}$ and cleaning up gives

$$\xi^2 + \alpha\xi + 1 = 0$$

with

$$\alpha = -\nu^2 \left[e^{-ikh} + e^{ikh} - 2\left(1 - \frac{1}{\nu^2}\right)\right] = -2\nu^2 \left[\cos(kh) - \left(1 - \frac{1}{\nu^2}\right)\right].$$

This is a quadratic equation with roots

$$\xi_\pm = \frac{1}{2}\left(-\alpha \pm \sqrt{\alpha^2 - 4}\right) = \frac{1}{2}\left(-\alpha \pm i\sqrt{(2-\alpha)(2+\alpha)}\right)$$

Our stability condition is $|\xi| \leq 1$, with this implying

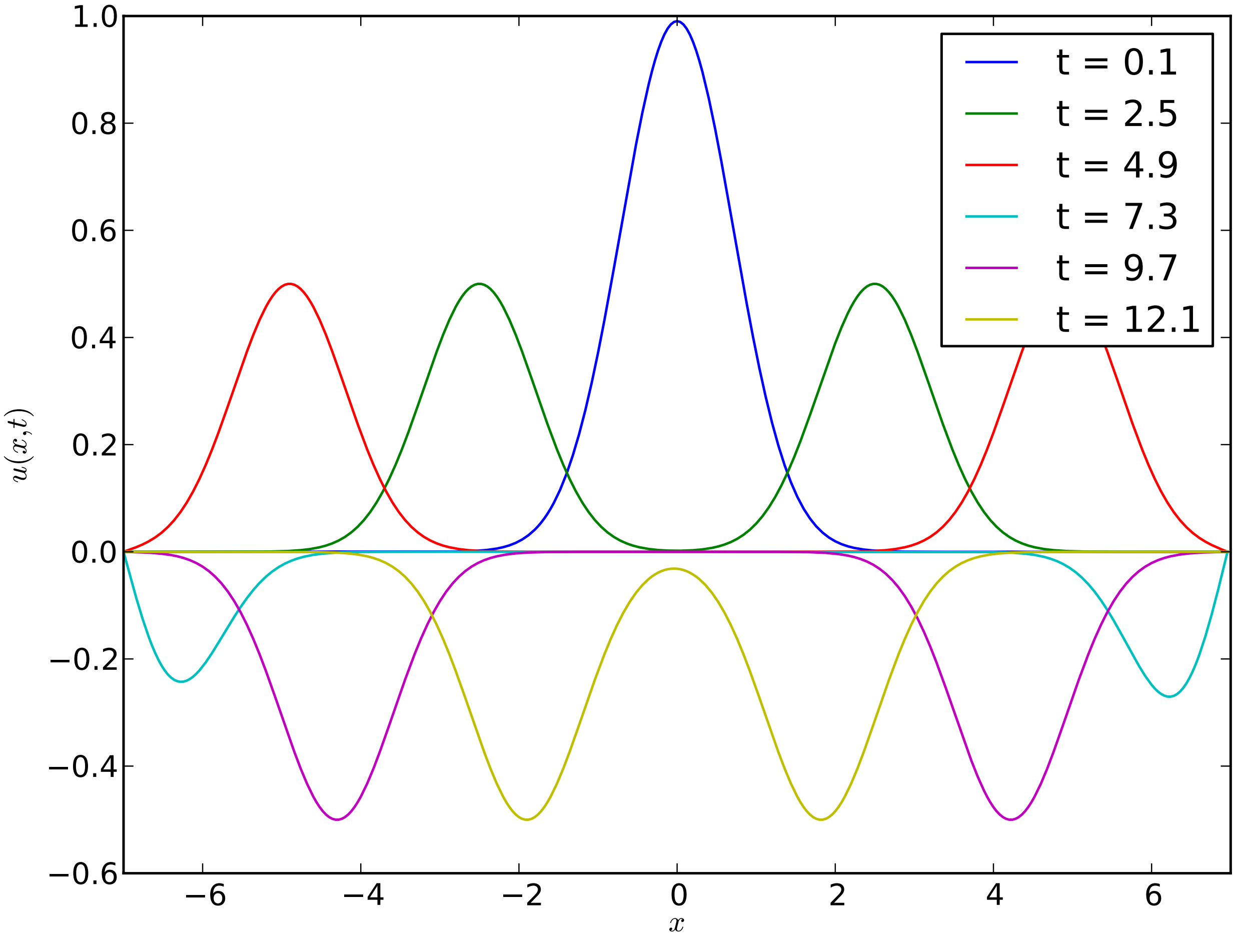

Figure 1: Solution to the wave equation with Gaussian initial condition.

(3)

The program used is included in appendix B.

A Program for 1.1

```
#!/usr/bin/env python3
# Program to solve the wave equation with c=1 using central differences.
# Dirichlet boundaries are used, and imposed implicitly, by allocating an array
# of zeros and not writing to the edge points.
# Liam O'Sullivan

from matplotlib.pyplot import *
from numpy import *

# Define our space and time step sizes.
dx = 0.04
dt = dx

# Define our Temporal and Spatial lengths.
Time = [0,14]
Dist = [-7,7]

# Define variables with the number of space/time steps for ease later.
# They must be cast as ints for use with range(), but should be integers
# anyway, since I'll only be setting dx as a nice value, I swear...
nt = int((Time[1] - Time[0])/dt)
nx = int((Dist[1] - Dist[0])/dx)

# Define our initial arrays.
U = zeros((nt, nx), dtype=float64)
t = zeros(nt, dtype=float64)
x = zeros(nx, dtype=float64)
# Initialise the tand x arrays.
for n in range(0,nt):
    t[n] = Time[0] + n*dt
for j in range(0,nx):
    x[j] = Dist[0] + j*dx

# Initialise the t_0 part of the array.
for j in range(1,nx-1):
    U[0][j] = e**(-x[j]*x[j])

# c is not c from the wave equation:
c = (dt/dx)**2

# Since U_j^n-1 is unknown, I take is equivalent to U_j^n for the first run.
for j in range(1,nx-1):
    U[1][j] = c*(U[0][j-1] - 2*U[0][j] + U[0][j+1]) + U[0][j]

# Now perform our integration.
for n in range(2,nt):
    for j in range(1,nx-1):
        U[n][j] = c*(U[n-1][j-1] - 2*U[n-1][j] + U[n-1][j+1]) - U[n-2][j] + 2*U[n-1][j]
    if ((n-2)%60) is 0:
        plot(x,U[n],label="t = "+str(round(14*(n/nt),1)))
xlabel("$x$")
ylabel("$u(x,t)$")
xlim([-7,7])
legend()
savefig("dirichlet.pdf",format="pdf")
show()
```

B Program for 1.3

```
#!/usr/bin/env python3
# Program to solve the wave equation with c=1 using central differences.
# Dirichlet boundaries are used, and imposed implicitly, by allocating an array
# of zeros and not writing to the edge points.
# A variety of h values are used.
# Liam O'Sullivan

from matplotlib.pyplot import *
from numpy import *

def run(dt,dx,T,D):
    # Define variables with the number of space/time steps for ease later.
    # They must be cast as ints for use with range(), but should be integers
    # anyway, since I'll only be setting dx as a nice value, I swear...
    nt = int((T[1] - T[0])/dt)
    nx = int((D[1] - D[0])/dx)

    # Define our initial arrays.
    U = zeros((nt, nx), dtype=float64)
    t = zeros(nt, dtype=float64)
    x = zeros(nx, dtype=float64)
    # Initialise the tand x arrays.
    for n in range(0,nt):
        t[n] = Time[0] + n*dt
    for j in range(0,nx):
        x[j] = Dist[0] + j*dx

    # Initialise the t_0 part of the array.
    for j in range(1,nx-1):
        U[0][j] = e**(-x[j]*x[j])

    # c is not c from the wave equation, just a useful constant.
    c = (dt/dx)**2

    # Since U_j^n-1 is unknown, I take is equivalent to U_j^n for the first run.
    for j in range(1,nx-1):
        U[1][j] = c*(U[0][j-1] - 2*U[0][j] + U[0][j+1]) + U[0][j]

    # Now perform our norm computation.
    for n in range(2,nt):
        for j in range(1,nx-1):
            U[n][j] = c*(U[n-1][j-1] - 2*U[n-1][j] + U[n-1][j+1]) - U[n-2][j] + 2*U[n-1][j]

    Ltime = nt*10.5/(T[1] - T[0])
    norm = 0
    for k in U[Ltime]:
        norm += k**2
    norm = sqrt(dx*norm)
    return norm


# Define our Temporal and Spatial lengths.
Time = [0,14]
Dist = [-7,7]

# Define our space and time step sizes.
dxs = [(0.2 + o/2500) for o in range(0,2000)]
dts = [(o**2)/4 for o in dxs] # The limit for stability.
err = []

for i in range(0,len(dxs)):
    Lfull = run(dts[i],dxs[i],Time,Dist)
    Lhalf = run(dts[i]/4,dxs[i]/2,Time,Dist)
    err.append(1 - (Lhalf/Lfull))
    print("h = "+str(round(dxs[i],4))+", err = "+str(err[-1]))

xlabel("h")
ylabel("$\epsilon(h)$")
#semilogy(dxs,err,'.')
```

```
plot(dxs,err,'.')
savefig("Error-linear.pdf",format='PDF')
show()
```