Math 104C HW5

May 12, 2022

Math 104C Homework #5

Aidan Baker (perm:8141905)

[84]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

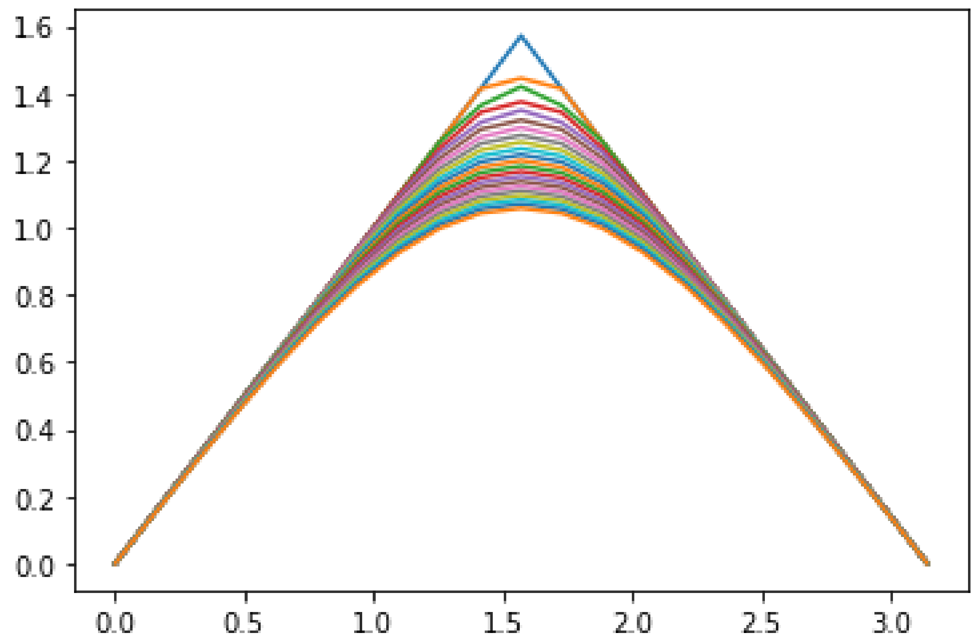
[85]:
```
##INITIAL BOUNDARY VALUE PROBLEM
def initial_value(M,delta_x):
    result = []
    for i in range(M+1):
        x = i * delta_x
        u = x if x < math.pi/2 else math.pi - x
        result.append(u)
    return np.array(result)


##EXPLICIT FINITE DIFFERENCE SCHEME
def explicit_forward(u_init,alpha,delta_t,N):
    current = u_init
    for _ in range(1,N):
        next = np.zeros_like(current)
    for j in range(1,M):
        next[j] = current[j] + alpha *(current[j-1] - 2*current[j]+current[j+1])
    current = next
    return current

##OUR INITIAL CONDITIONS
D=1
M=20
delta_x = math.pi/M
alpha = 0.4
delta_t = alpha * (delta_x **2) / D
N=150
x_values = [i*delta_x for i in range(M+1)]
initial = initial_value(M,delta_x)
plt.plot(x_values,initial)
```

```
for i in range(M+1):
    u = explicit_forward(initial,alpha,delta_t,N)
    plt.plot(x_values,u)
    initial = u

## WE SEE THAT WE HAVE A STABLE RESULT IF ALPHA <= .5. If we take a value over .
↪5 we lose stability
```

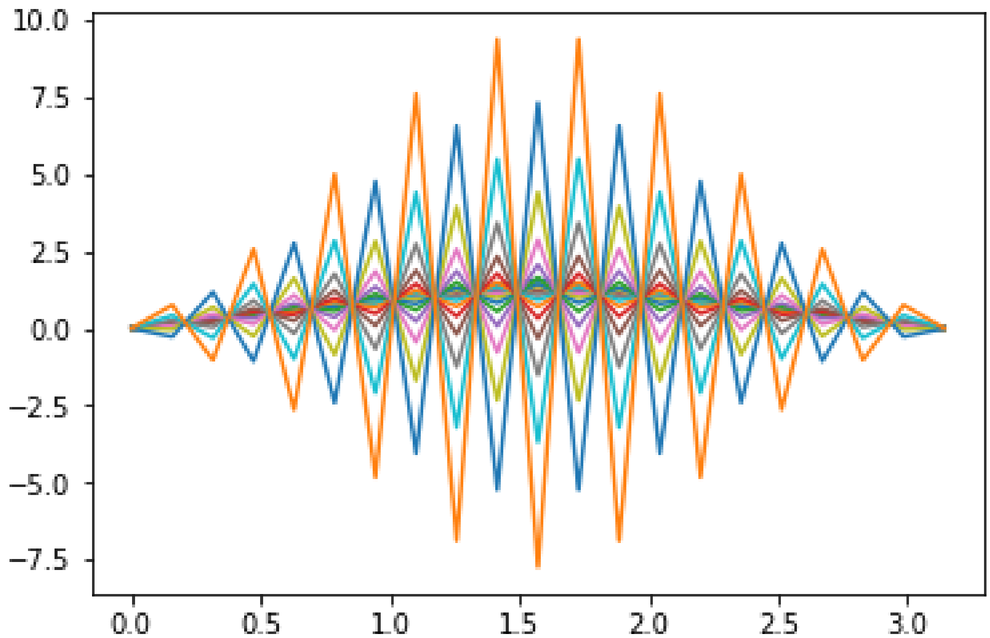


[86]:

```
## REPEATING PROCESS WITH ALPHA OVER .5 Shows how we lose stability

D=1
M=20
delta_x = math.pi/M
alpha = 0.6
delta_t = alpha * (delta_x **2) / D
N=150
x_values = [i*delta_x for i in range(M+1)]
initial = initial_value(M,delta_x)
plt.plot(x_values,initial)

for i in range(M+1):
    u = explicit_forward(initial,alpha,delta_t,N)
    plt.plot(x_values,u)
    initial = u
```



[87]:

```
def tridiag_mat(M,alpha):
    A = -2 * alpha * np.eye(M)
    for i in range(M-1):
        A[i][i+1] = 1.0
        A[i+1][i] = 1.0
    return np.eye(M) - A

def system_solver(A,b):
    c = np.zeros(len(b)-1)
    d = np.zeros(len(b))
    for i in range(len(b)):
        if i == 0:
            c[i] = A[i][i+1]/A[i][i]
            d[i] = b[i]/A[i][i]
        elif i == len(b)-1:
            d[i] = (b[i]-A[i][i-1]*d[i-1])/(A[i][i]-A[i][i-1]*c[i-1])
        else:
            c[i] = A[i][i+1]/(A[i][i] - A[i][i-1]*c[i-1])
            d[i] = (b[i]-A[i][i-1]*d[i-1])/(A[i][i] - A[i][i-1]*c[i-1])
    x = np.zeros(len(b))
    for i in reversed(range(len(b))):
        if i == len(b)-1:
            x[i] = d[i]
        else:
            x[i] = d[i] - c[i]*x[i+1]
```

```
    return x

def backward_difference(u_init,alpha,delta_t,N):
    M = len(u_init)
    current = u_init
    tridiag = tridiag_mat(M,alpha)
    for i in range(N):
        current = system_solver(tridiag,current)
    return current

delta_x = 0.5
delta_t = delta_x
D = 1
alpha = D * (delta_t)/(delta_x **2)
N = 150
x_values = np.array([i*delta_x for i in range(M+1)])
approx = backward_difference(initial,alpha,delta_t,N)
plt.plot(x_values,approx)

## WE SEE THAT THE BACKWARDS DIFFERENCE METHOD IS ALSO STABLE for delta_t =␣
↪delta_x and D=1
```

[87]: [<matplotlib.lines.Line2D at 0x7fcef04c89d0>]