AMATH 581 Homework #4

Tyler Shakibai

1. Log-log Plot of the Error Versus Δx for the (1,4) Method and the Crank-Nicolson Method.

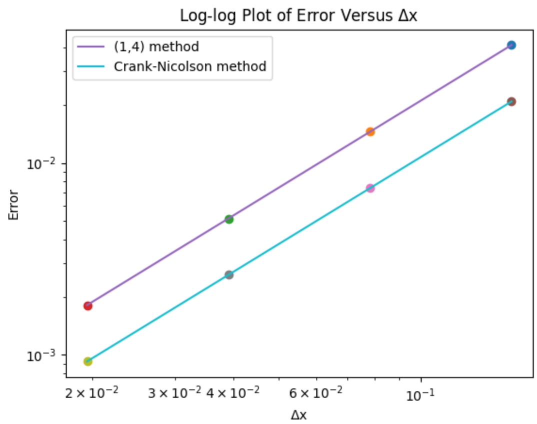

Figure 1: Log-log plot comparing the 2-norm error for the (1 temporal, 4 spacial) Method and the (2 temporal, 2 spacial) Crank-Nicolson Method at different Δx values. The norm of the difference between the numerical solution and the exact solution is taken for 128, 256, 512, and 1024 equally spaced points for Δx. A polynomial of degree 1 is fit to reveal that the order of accuracy of both numerical methods is approximately 1.5, with the slopes being $m_1 = 1.50009$ for the (1, 4) Method and $m_2 = 1.49967$ for the Crank-Nicolson Method.

Code:

```
exact1 = np.loadtxt('hw-4\exact_128.csv').reshape(-1, 1)
exact2 = np.loadtxt('hw-4\exact_256.csv').reshape(-1, 1)
exact3 = np.loadtxt('hw-4\exact_512.csv').reshape(-1, 1)
exact4 = np.loadtxt('hw-4\exact_1024.csv').reshape(-1, 1)
A11 = np.linalg.norm(exact1 - A5)
A12 = np.linalg.norm(exact1 - A9)

x = np.linspace(-10, 10, 256, endpoint=False)
dx = 20/256
t = np.linspace(0, 2, 2001)
dt = (2/500)/4
CFL = (2*dt)/(dx)**2

n = 256
e = np.ones(n)
matrix1 = scipy.sparse.spdiags([16*e, -e, -e, 16*e, -30*e, 16*e, -e, \
-e, 16*e], [1-n, 2-n, -2, -1, 0, 1, 2, n-2, n-1], n, n, format='csc')/12

sol1 = np.zeros((len(x), len(t)))
u0 = 10*np.cos(2*np.pi*x/10) + 30*np.cos(8*np.pi*x/10)
sol1[:, 0] = u0
for i in range(int(2/dt)):
    u1 = u0 + CFL*(matrix1@u0)
    u0 = u1
    sol1[:, i+1] = u1

A13 = np.linalg.norm(exact2 - sol1[:, -1].reshape(-1, 1))

matrix2 = scipy.sparse.eye(256, format='csc') - \
(CFL/2)*scipy.sparse.spdiags([e, e, -2*e, e, e], [1-n, -1, 0, 1, n-1], \
n, n, format='csc')
```

```
matrix3 = scipy.sparse.eye(256, format='csc') + \
(CFL/2)*scipy.sparse.spdiags([e, e, -2*e, e, e], [1-n, -1, 0, 1, n-1], \
n, n, format='csc')

sol2 = np.zeros((len(x), len(t)))
v0 = 10*np.cos(2*np.pi*x/10) + 30*np.cos(8*np.pi*x/10)
sol2[:, 0] = v0
PLU = scipy.sparse.linalg.splu(matrix2)
for i in range(int(2/dt)):
    v1 = PLU.solve(matrix3@v0)
    v0 = v1
    sol2[:, i+1] = v1

A14 = np.linalg.norm(exact2 - sol2[:, -1].reshape(-1, 1))

## 2D Plot

curr_norm1 = []
curr_norm2 = []

for i, exact in enumerate([exact1, exact2, exact3, exact4]):
    x = np.linspace(-10, 10, 128*(2**i), endpoint=False)
    dx = 20/(128*(2**i))
    t = np.linspace(0, 2, 500*(4**i) + 1)
    dt = (2/(500*(4**i)))

    n = 128*(2**i)
    e = np.ones(n)
    matrix1 = scipy.sparse.spdiags([16*e, -e, -e, 16*e, -30*e, 16*e, \
    -e, -e, 16*e], [1-n, 2-n, -2, -1, 0, 1, 2, n-2, n-1], n, n, \
    format='csc')/12
```

```
    sol1 = np.zeros((len(x), len(t)))
    u0 = 10*np.cos(2*np.pi*x/10) + 30*np.cos(8*np.pi*x/10)
    sol1[:, 0] = u0
    for i in range(int(2/dt)):
        u1 = u0 + CFL*(matrix1@u0)
        u0 = u1
        sol1[:, i+1] = u1

    curr_norm1.append(np.linalg.norm(exact - sol1[:, -1].reshape(-1, 1)))

    matrix2 = scipy.sparse.eye(n, format='csc') - \
    (CFL/2)*scipy.sparse.spdiags([e, e, -2*e, e, e], [1-n, -1, 0, 1, \
    n-1], n, n, format='csc')
    matrix3 = scipy.sparse.eye(n, format='csc') + \
    (CFL/2)*scipy.sparse.spdiags([e, e, -2*e, e, e], [1-n, -1, 0, 1, \
    n-1], n, n, format='csc')

    sol2 = np.zeros((len(x), len(t)))
    v0 = 10*np.cos(2*np.pi*x/10) + 30*np.cos(8*np.pi*x/10)
    sol2[:, 0] = v0
    PLU = scipy.sparse.linalg.splu(matrix2)
    for i in range(int(2/dt)):
        v1 = PLU.solve(matrix3@v0)
        v0 = v1
        sol2[:, i+1] = v1

    curr_norm2.append(np.linalg.norm(exact - sol2[:, -1].reshape(-1, 1)))

fig, ax = plt.subplots()
ax.set_title(r'Log-log Plot of Error Versus $\Delta$x')
ax.set_xlabel('$\Delta$x')
ax.set_ylabel('Error')
```

```
deltaX = [20/(128*(2**0)), 20/(128*(2**1)), 20/(128*(2**2)), \
20/(128*(2**3))]
ax.loglog(deltaX[0], curr_norm1[0], 'o')
ax.loglog(deltaX[1], curr_norm1[1], 'o')
ax.loglog(deltaX[2], curr_norm1[2], 'o')
ax.loglog(deltaX[3], curr_norm1[3], 'o')

m1, b1 = np.polyfit(np.log(deltaX), np.log(curr_norm1), 1)
ax.loglog(deltaX, np.exp(m1*np.log(deltaX) + b1), label='(1,4) method')

ax.loglog(deltaX[0], curr_norm2[0], 'o')
ax.loglog(deltaX[1], curr_norm2[1], 'o')
ax.loglog(deltaX[2], curr_norm2[2], 'o')
ax.loglog(deltaX[3], curr_norm2[3], 'o')

m2, b2 = np.polyfit(np.log(deltaX), np.log(curr_norm2), 1)
ax.loglog(deltaX, np.exp(m2*np.log(deltaX) + b2), \
label='Crank-Nicolson method')
ax.legend()
```