AMath 586 - HW 2

Vivek Gandhi

April 30, 2025

T1

(I'm using j for indexing to avoid confusion with the imaginary i)

$$\frac{U_j^{n+1} - U_j^{n-1}}{2\Delta t} = \frac{U_{j+1}^n - 2U_j^n + U_{j-1}^n}{(\Delta x)^2} \tag{1}$$

As the heat equation in question has constant coefficients ($\partial_t u = \partial_{xx} u$), we can try using Von-Neumann analysis to study the L^2 stability of the leapfrog method.

Assume $U_j^n = g(\xi)^n e^{i\xi jh}$ with $h = \Delta x$. Then, 1 becomes:

$$\begin{aligned}
\frac{g(\xi)^{n+1}e^{i\xi jh} - g(\xi)^{n-1}e^{i\xi jh}}{2\Delta t} &= \frac{g(\xi)^n \left(e^{i\xi(j+1)h} - 2e^{i\xi jh} + e^{i\xi(j-1)h}\right)}{(\Delta x)^2} \\
g(\xi)^{n-1}e^{i\xi jh}\left(\frac{g(\xi)^2 - 1}{2\Delta t}\right) &= g(\xi)^n e^{i\xi jh}\left(\frac{e^{i\xi h} - 2 + e^{-i\xi h}}{(\Delta x)^2}\right) \\
\frac{g(\xi)^2 - 1}{2\Delta t} &= \frac{g(\xi)}{(\Delta x)^2}\left(e^{i\xi h} - 2 + e^{-i\xi h}\right) \\
\frac{g(\xi)^2 - 1}{2\Delta t} &= \frac{g(\xi)}{(\Delta x)^2}\left(2\cos(\xi h) - 2\right) \\
g(\xi)^2 - 1 &= \left(\frac{4\Delta t}{(\Delta x)^2}\right) g(\xi)\left(\cos(\xi h) - 1\right)
\end{aligned} \tag{2}$$

As $U_j^n = g(\xi)^n e^{i\xi jh}$, $|U_j^n| = |g(\xi)^n e^{i\xi jh}| \leq |g(\xi)|^n$.

For stability, we want U_j^n to remain bounded as $n \to \infty$. So, we want conditions on Δt and Δx that might ensure that $|g(\xi)| \leq 1$.

Condition 2 is quadratic in $g(\xi)$. We can assume the worst case $\cos(\xi h) = -1$ to see if we can find a valid condition.

For simplicity, let $A := \dfrac{4\Delta t}{(\Delta x)^2}$. As $\Delta t > 0$ and $\Delta x > 0$, $A > 0$.

$$g(\xi)^2 - 1 = -2A\ g(\xi)$$

$$g(\xi)^2 + 2A\ g(\xi) - 1 = 0$$

$$g(\xi) = \frac{-2A \pm \sqrt{4A^2 + 4}}{2} = -A \pm \sqrt{A^2 + 1}$$

As $A^2 > 0$, $\sqrt{A^2+1} > 1$. As the difference between the two roots $2\sqrt{A^2+1} > 2$, at least one of them must satisfy $|g(\xi)| > 1$, violating our requirement of $|g(\xi)| \leq 1$.

Therefore, Von-Neumann analysis can't give us a condition to ensure L^2 stability of this method, suggesting that it is always L^2 unstable.

T2

$$\frac{U_j^{n+1} - U_j^n}{\Delta t} + b\frac{U_{j+1}^{n+1} - U_{j-1}^{n+1}}{2\Delta x} = a\frac{U_{j+1}^{n+1} - 2U_j^{n+1} + U_{j-1}^{n+1}}{(\Delta x)^2} \tag{2}$$

As the PDE in question has constant coefficients ($\partial_t u + b\partial_x u = a\partial_{xx}u$), we can try using Von-Neumann analysis to study the L^2 stability of the finite difference scheme.

Assume $U_j^n = g(\xi)^n e^{i\xi jh}$ with $h = \Delta x$.
For simplicity, split equation 2 into 3 terms such that $T_1 + T_2 = T_3$. Then, the terms becomes:

$$\begin{aligned}
T_1 &= \frac{g(\xi)^{n+1}e^{i\xi jh} - g(\xi)^n e^{i\xi jh}}{\Delta t} \\
&= g(\xi)^n e^{i\xi jh}\left(\frac{g(\xi) - 1}{\Delta t}\right) \\
T_2 &= b\ \frac{g(\xi)^{n+1}e^{i\xi(j+1)h} - g(\xi)^{n+1}e^{i\xi(j-1)h}}{2\Delta x} \\
&= b\ g(\xi)^{n+1}e^{i\xi jh}\left(\frac{e^{i\xi h} - e^{-i\xi h}}{2\Delta x}\right) = bi\ g(\xi)^{n+1}e^{i\xi jh}\left(\frac{\sin(\xi h)}{\Delta x}\right) \\
T_3 &= a\ \frac{g(\xi)^{n+1}e^{i\xi(j+1)h} - 2g(\xi)^{n+1}e^{i\xi jh} + g(\xi)^{n+1}e^{i\xi(j-1)h}}{(\Delta x)^2} \\
&= a\ g(\xi)^{n+1}e^{i\xi jh}\left(\frac{e^{i\xi h} - 2 + e^{-i\xi h}}{(\Delta x)^2}\right) = 2a\ g(\xi)^{n+1}e^{i\xi jh}\left(\frac{\cos(\xi h) - 1}{(\Delta x)^2}\right)
\end{aligned}$$

Canceling $g(\xi)^{n+1}e^{i\xi jh}$ from each term leaves us with:

$$\begin{aligned}
\left(\frac{1 - g(\xi)^{-1}}{\Delta t}\right) + \frac{ib}{\Delta x}\sin(\xi h) &= \frac{2a}{(\Delta x)^2}(\cos(\xi h) - 1) \\
\frac{ib\Delta t}{\Delta x}\sin(\xi h) + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h)) &= g(\xi)^{-1} - 1 \\
g(\xi) &= \frac{1}{1 + \frac{ib\Delta t}{\Delta x}\sin(\xi h) + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h))}
\end{aligned} \tag{3}$$

As $U_j^n = g(\xi)^n e^{i\xi jh}$, $|U_j^n| = |g(\xi)^n e^{i\xi jh}| \leq |g(\xi)|^n$.

For stability, we want U_j^n to remain bounded as $n \to \infty$. So, we want conditions on Δt and Δx that might ensure that $|g(\xi)| \leq 1$.

So, we need the magnitude of the denominator in 3 to be greater than or equal to 1.

$$\left|1 + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h)) + \frac{ib\Delta t}{\Delta x}\sin(\xi h)\right| \geq 1$$

As this is a complex number of the form $c + id$:

$$\begin{aligned}
|c + id| &\geq 1 \\
\sqrt{c^2 + d^2} &\geq 1 \\
c^2 + d^2 &\geq 1 \\
c^2 &\geq 1 - d^2 \\
\left(\frac{b\Delta t}{\Delta x}\sin(\xi h)\right)^2 &\geq 1 - \left(1 + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h))\right)^2
\end{aligned}$$

Notice that all the constants on the right hand size are greater than 0 $(a, \Delta t, \Delta x)$.

$$
\begin{aligned}
\text{As} \qquad 1 - \cos(\xi h) &\geq 0, \\
\frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h)) &\geq 0 \\
1 + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h)) &\geq 1 \\
\left(1 + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h))\right)^2 &\geq 1 \\
1 - \left(1 + \frac{2a\Delta t}{(\Delta x)^2}(1 - \cos(\xi h))\right)^2 &\leq 0
\end{aligned}
$$

As the left hand side is squared, it must be greater than or equal to 0. As the right hand size is always less than or equal to 0, this inequality is satisfied unconditionally ($|g(\xi)| \leq 1$ always). Therefore, Von-Neumann analysis suggests that this scheme is unconditionally L^2 stable.

C1

	Explicit Euler	**Crank–Nicolson**	**Implicit Euler**
θ	0	0.5	1
L^1 **slope**	2.00004	1.98972	0.98170
L^2 **slope**	2.00004	1.99554	0.98569
L^∞ **slope**	1.99616	2.01947	0.99056
H^1 **slope**	2.03623	2.02947	1.02225

Table 1: I used polyfit to find the slopes of each line on the log-log plots. This shows that the Explicit-Euler and Crank-Nicolson solutions do indeed converge quadratically $\mathcal{O}(\Delta x^2)$, while the implicit Euler scheme only converges linearly $\mathcal{O}(\Delta x)$.

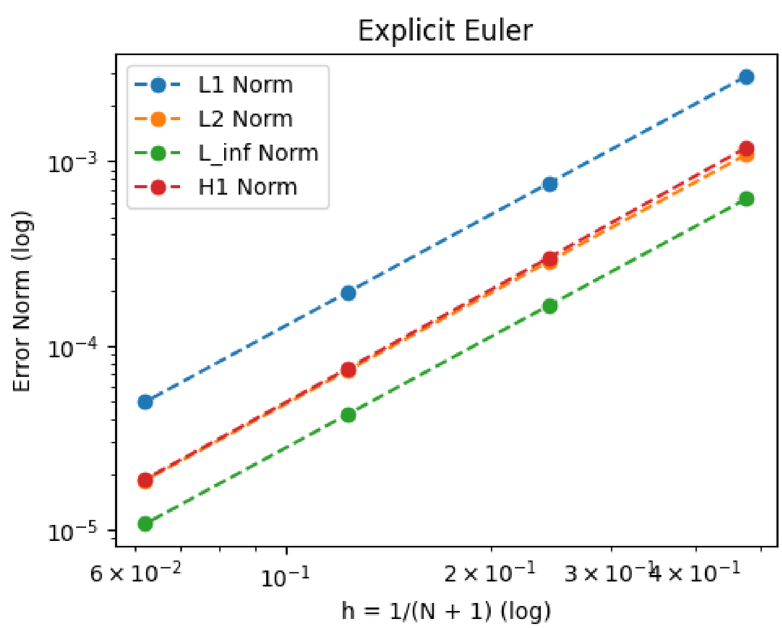

Figure 1: Error for the Explicit Euler scheme in different norms plotted on a log-log plot.

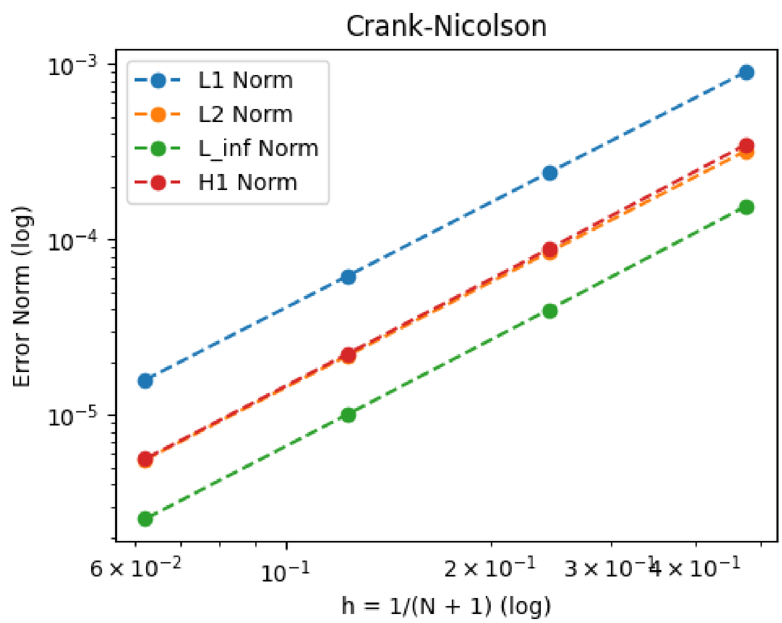

Figure 2: Error for the Crank-Nicolson scheme in different norms plotted on a log-log plot.

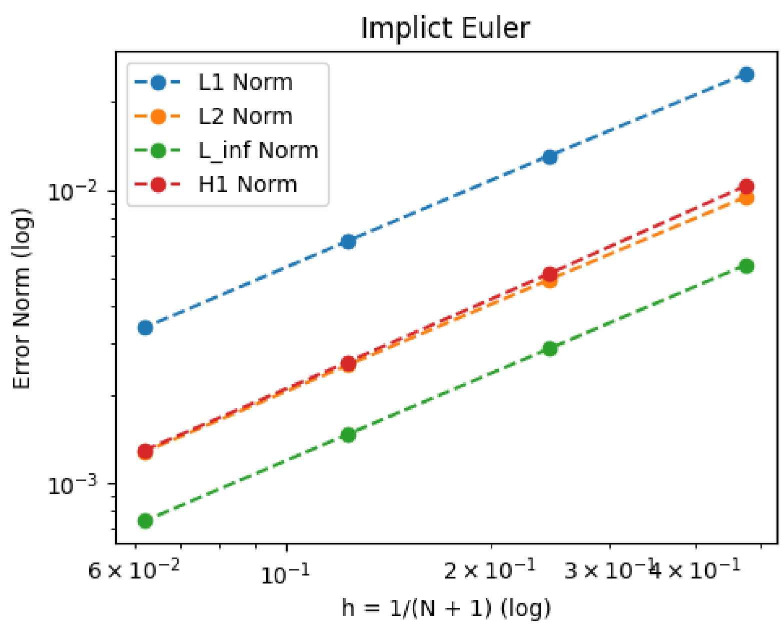

Figure 3: Error for the Implicit Euler scheme in different norms plotted on a log-log plot.

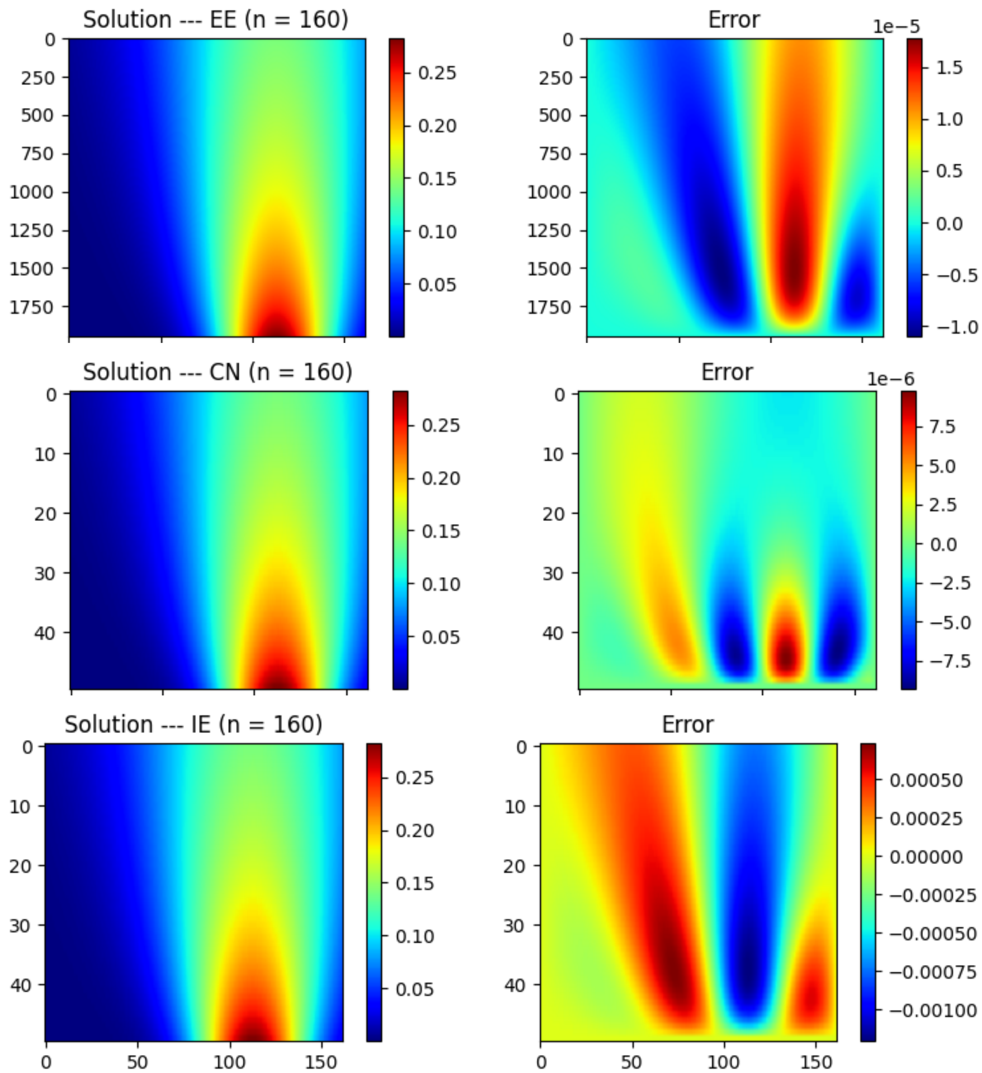

Figure 4: Solution and error plots of each method at $T = 3$. ($n = 160$)

```python
import numpy as np

def NormL1(v, dx):
  return dx*np.sum(np.abs(v[1:-1]))

def NormLInf(v, dx):
  return np.max(np.abs(v[1:-1]))

def NormL2(v, dx):
  v = v[1:-1]
  return np.sqrt(np.sum(dx * v**2))

def NormH1(v, dx):
  return np.sqrt(NormL2(v, dx)**2 + DxV_xhNorm2(v, dx))

def DxV_xhNorm2(v, dx):
  base = np.copy(v[1:])
  base = base - v[:-1]
  return np.sum(base**2)
```

Listing 1: Norms.py

```python
import numpy as np
import matplotlib.pyplot as plt

# For A
from scipy.sparse import spdiags
# To solve AL = U
from scipy.sparse.linalg import gmres, spsolve
# from scipy.linalg import solve
from Norms import *

# ---------------------------------------------

do_viz = False

# ---------------------------------------------
# Explicit solution

def v(t, x):
  coef = 1/np.sqrt(4*np.pi*t)
  exp = np.exp(-((x-2)**2)/(4*t))
  return coef*exp

# =============================================

def main():
  Ns = [20, 40, 80, 160]
  dxs = [10/(n+1) for n in Ns]
  modes = ['EE', 'CN', 'IE']

  for mode in modes:
    L1s = []
    L2s = []
    Linfs = []
    H1s = []

    for n in Ns:
      sol, error = theta_scheme(mode, n)
      viz(sol, error, f'{mode} (n = {n})')

      eT = error[0] # Top slice - error at time T
      dx = 10/(n+1)

      L1s.append(NormL1(eT, dx))
      L2s.append(NormL2(eT, dx))
      Linfs.append(NormLInf(eT, dx))
      H1s.append(NormH1(eT, dx))

    get_slope = lambda errors: np.polyfit(np.log10(dxs), np.log10(errors), 1)[0]

    plt.loglog(dxs, L1s, label='L1 Norm', linestyle='--', marker='o')
    plt.loglog(dxs, L2s, label='L2 Norm', linestyle='--', marker='o')
    plt.loglog(dxs, Linfs, label='L_inf Norm', linestyle='--', marker='o')
    plt.loglog(dxs, H1s, label='H1 Norm', linestyle='--', marker='o')
    plt.legend()
    plt.title(f'{mode}')
    plt.xlabel('h = 1/(N + 1) (log)')
    plt.ylabel('Error Norm (log)')
    plt.tight_layout()
    plt.show()

    print(f'\n{mode}')
    print(f'L1 slope - {get_slope(L1s)}')
    print(f'L2 slope - {get_slope(L2s)}')
    print(f'L_inf slope - {get_slope(Linfs)}')
    print(f'H1 slope - {get_slope(H1s)}')

# =============================================

```

```python
def theta_scheme(mode, n):
  T = 3

  # ------ Setup -----------------------

  xRange = np.linspace(-5, 5, n+2)
  dx = np.diff(xRange)[0]
  dt = 0.4*(dx**2) if mode == 'EE' else dx
  dt_dx2 = dt/(dx*dx)

  tRange = np.arange(0, T+dt, dt)

  theta = 1   # IE
  if mode == 'EE':
    theta = 0
  elif mode == 'CN':
    theta = 0.5

  # ------ Produce u0 ------------------

  un = generate_u0(xRange, lambda x: v(1, x))
  u = np.array([un])

  # ------ Produce ImpMat and ExpMat ---

  A = 1 + (2*theta*dt_dx2)    # Implicit diagonal
  B = -theta*dt_dx2       # Implicit off-diagonal
  C = 1 - (1 - theta)*2*dt_dx2  # Explicit diagonal
  D = (1 - theta)*dt_dx2      # Explicit off-diagonal

  ImpMat = buildMatrix(n, A, B)
  ExpMat = buildMatrix(n+2, C, D)   # n+2 to include U_0 and U_N+1

  # ------ Step through time -----------

  for t in tRange[1:]:
    F = (ExpMat @ un)[1:-1]     # Calculate explicit part

    U_0 = v(t+1, -5)
    U_N1 = v(t+1, 5)

    if mode == 'EE':
      un = F            # Return early if fully explicit
    else:
      F[0] -= B*U_0
      F[-1] -= B*U_N1
      un = spsolve(ImpMat, F)   # Solve implicit part

    un = np.concatenate(([U_0], un, [U_N1]))
    u = np.insert(u, 0, [un], axis=0)

  # ------ Calculate error -------------

  real = generate_full(xRange, tRange, lambda t, x: v(t+1, x))
  error = real-u

  return [u, error]

# ============================================
# Helpers
# ============================================

# Generate an nxn matrix across ranges with an initializer func
def generate_u0(xRange, func):
  w0 = np.zeros(xRange.size)
  for x in range(xRange.size):
    w0[x] = func(xRange[x])
  return w0
```

```python

# ---------------------------------------------
# Generate an nxn matrix across ranges with an initializer func
def generate_full(xRange, tRange, func):
  w0 = np.zeros((tRange.size, xRange.size))
  for x in range(xRange.size):
    for t in range(tRange.size):
      w0[t,x] = func(tRange[t], xRange[x])
  return w0[::-1]

# ---------------------------------------------
# Vizualize U and error=U-u
def viz(A, B, title=''):
  if not do_viz:
    return
  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
  im1 = ax1.imshow(A, cmap='jet')
  im2 = ax2.imshow(B, cmap='jet')
  ax1.set_aspect(1.0/ax1.get_data_ratio(), adjustable='box')
  ax2.set_aspect(1.0/ax2.get_data_ratio(), adjustable='box')
  ax1.set_box_aspect(1)
  ax2.set_box_aspect(1)
  ax1.set_title(f'Solution --- {title}')
  ax2.set_title(f'Error')

  fig.colorbar(im1, ax=ax1)
  fig.colorbar(im2, ax=ax2)

  plt.tight_layout()
  plt.show()

# =============================================
# Builds an nxn matrix with A on the diagonal and
#   B on the off-diagonals
# =============================================

def buildMatrix(n, A, B):
  e1 = np.ones(n)     # vector of 1s

  diagonals = [B*e1, A*e1, B*e1]
  offsets = [-1, 0, 1]

  mat = spdiags(diagonals, offsets, n, n, format = 'csr')
  return mat

# =============================================

main()
```

Listing 2: HW2-C1.py