Tutorial 3

https://github.com/Blink29/Numerical-Techniques → Assignment 3

Name: Paurush Kumar Roll Number: NA22B002

Problem Statement

We aim to solve the 1D transient diffusion equation numerically and compare the results of two numerical methods with the analytical solution.

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

boundary conditions are: u(0,t)=0 and u(L,t)=0,for all t.

We take u(x, t) = sin(2πx)exp(−4(π^2)γt).

Numerical Methods

1. **FTCS (Forward Time-Centered Space) - Explicit Method**

Using finite difference discretization:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \gamma \frac{u_{i-1}^n - 2u_i^n + u_{i+1}^n}{\Delta x^2},$$

which gives the explicit update formula:

$$u_i^{n+1} = u_i^n + r\left(u_{i-1}^n - 2u_i^n + u_{i+1}^n\right), \quad \text{where } r = \frac{\gamma \Delta t}{\Delta x^2}.$$

This method is conditionally stable and requires r ≤ 0.5 for stability.

2. **Implicit Method**

Using backward Euler discretization for time and central differencing for space:

$$\frac{u_i^{n+1} - u_i^n}{\Delta t} = \gamma \frac{u_{i-1}^{n+1} - 2u_i^{n+1} + u_{i+1}^{n+1}}{\Delta x^2}.$$

Rearranging into a matrix system:

$$Au^{n+1} = u^n$$

where A is a tridiagonal matrix defined by:

$$A = \begin{bmatrix} 1+2r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \cdots & 0 \\ 0 & -r & 1+2r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & -r & 1+2r \end{bmatrix}.$$

Code

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from mpl_toolkits.mplot3d import Axes3D

# Parameters
gamma = 0.001  # Diffusion coefficient
L = 1.0        # Length of the domain
T = 10.0       # Total simulation time
N_x = 20       # Number of spatial intervals
dt = 1 / 20    # Time step
N_t = int(T / dt)  # Number of time intervals
dx = L / N_x
r = gamma * dt / dx**2

# Discretize the domain
x = np.linspace(0, L, N_x + 1)
t = np.linspace(0, T, N_t + 1)
```

```
# Initial condition: u(x, 0)
u_initial = np.sin(2 * np.pi * x)

# Analytical function
def analytical_solution(x, t):
    return np.sin(2 * np.pi * x) * np.exp(3 * -np.pi**2 * gam
ma * t)



# Forward Time Central Space (FTCS - Explicit)
U_ftcs = [u_initial.copy()]
u_ftcs = u_initial.copy()

for n in range(N_t):
    u_new = u_ftcs.copy()
    for i in range(1, N_x):  # Skip boundary points
        u_new[i] = u_ftcs[i] + r * (u_ftcs[i-1] - 2*u_ftcs[i]
+ u_ftcs[i+1])
    u_ftcs = u_new.copy()
    U_ftcs.append(u_ftcs)

U_ftcs = np.array(U_ftcs)

# Implicit Method
A = np.zeros((N_x-1, N_x-1))
np.fill_diagonal(A, 1 + 2*r)
np.fill_diagonal(A[:-1, 1:], -r)
np.fill_diagonal(A[1:, :-1], -r)

U_implicit = [u_initial.copy()]
u_implicit = u_initial[1:-1].copy()

for n in range(N_t):
    rhs = u_implicit.copy()
    u_implicit = np.linalg.solve(A, rhs)
    U_implicit.append(np.concatenate(([0], u_implicit, [0])))
```

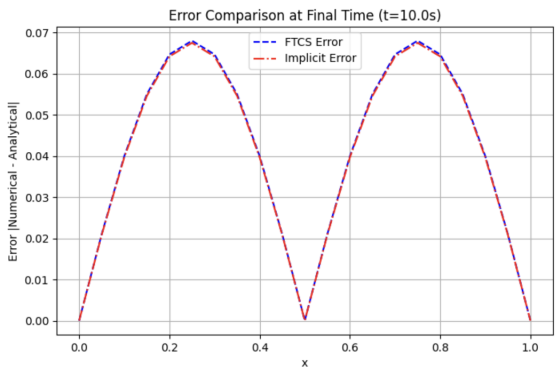
```
U_implicit = np.array(U_implicit)

# Analytical Solution
U_analytical = np.array([[analytical_solution(xi, tn) for xi
in x] for tn in t])

# Calculate Errors
error_ftcs = np.abs(U_ftcs - U_analytical)
error_implicit = np.abs(U_implicit - U_analytical)



# Plot
fig, ax = plt.subplots(figsize=(10, 6))
line_analytical, = ax.plot(x, U_analytical[0], 'g-', linewidt
h=2.5, label="Analytical Solution")
line_ftcs, = ax.plot(x, U_ftcs[0], 'b--', label="FTCS (Explic
it)")
line_implicit, = ax.plot(x, U_implicit[0], 'r-.', label="Impl
icit Method")
time_text = ax.text(0.05, 0.9, '', transform=ax.transAxes, fo
ntsize=12)
ax.set_title("Comparison of Numerical Methods for Diffusion E
quation")
ax.set_xlabel("x")
ax.set_ylabel("u(x, t)")
ax.legend()
ax.grid()

def update(frame):
    line_analytical.set_ydata(U_analytical[frame])
    line_ftcs.set_ydata(U_ftcs[frame])
    line_implicit.set_ydata(U_implicit[frame])
    time_text.set_text(f"Time: {t[frame]:.2f} s")
    return line_analytical, line_ftcs, line_implicit, time_te
xt
```

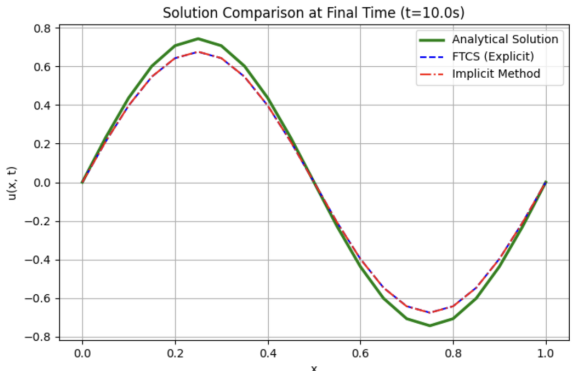
```
ani = FuncAnimation(fig, update, frames=range(0, N_t, N_t //
100), interval=50, blit=True)

ani.save("diffusion_methods_comparison.gif", writer="pillow")

# Error Plot at Final Time
plt.figure(figsize=(8, 5))
plt.plot(x, error_ftcs[-1], label="FTCS Error", linestyle='--
', color='blue')
plt.plot(x, error_implicit[-1], label="Implicit Error", lines
tyle='-.', color='red')
plt.xlabel("x")
plt.ylabel("Error |Numerical - Analytical|")
plt.title(f"Error Comparison at Final Time (t={T}s)")
plt.legend()
plt.grid()
plt.savefig("error_comparison_final_time.png")
plt.show()


# Static Plot of Results at Final Time
plt.figure(figsize=(8, 5))
plt.plot(x, U_analytical[-1], 'g-', linewidth=2.5, label="Ana
lytical Solution")
plt.plot(x, U_ftcs[-1], 'b--', label="FTCS (Explicit)")
plt.plot(x, U_implicit[-1], 'r-.', label="Implicit Method")
plt.xlabel("x")
plt.ylabel("u(x, t)")
plt.title(f"Solution Comparison at Final Time (t={T}s)")
plt.legend()
plt.grid()
plt.savefig("solution_comparison_final_time.png")
plt.show()
```

```
# 3D Plot of FTCS Solution
fig_3d_ftcs = plt.figure(figsize=(12, 8))
ax_3d_ftcs = fig_3d_ftcs.add_subplot(111, projection='3d')

X, T_grid = np.meshgrid(x, t)

ax_3d_ftcs.plot_surface(X, T_grid, U_ftcs, color='b', alpha=
0.7)

ax_3d_ftcs.set_xlabel("Space (x)")
ax_3d_ftcs.set_ylabel("Time (t)")
ax_3d_ftcs.set_zlabel("u(x, t)")
ax_3d_ftcs.set_title("FTCS (Explicit) Solution Over Time and
Space")

fig_3d_ftcs.savefig("ftcs_3d_solution.png", dpi=300)

plt.show()

# 3D Plot of Implicit Solution
fig_3d_implicit = plt.figure(figsize=(12, 8))
ax_3d_implicit = fig_3d_implicit.add_subplot(111, projection
='3d')

ax_3d_implicit.plot_surface(X, T_grid, U_implicit, color='r',
alpha=0.7)

ax_3d_implicit.set_xlabel("Space (x)")
ax_3d_implicit.set_ylabel("Time (t)")
ax_3d_implicit.set_zlabel("u(x, t)")
ax_3d_implicit.set_title("Implicit Method Solution Over Time
and Space")

fig_3d_implicit.savefig("implicit_3d_solution.png", dpi=300)

plt.show()
```

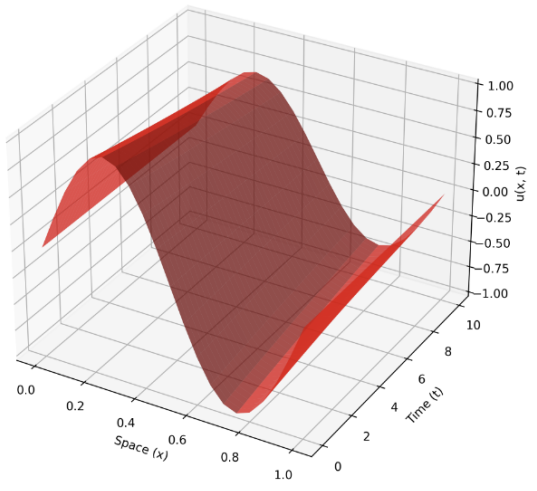
```
# 3D Plot of Analytical Solution
fig_3d_analytical = plt.figure(figsize=(12, 8))
ax_3d_analytical = fig_3d_analytical.add_subplot(111, project
ion='3d')

ax_3d_analytical.plot_surface(X, T_grid, U_analytical, color
='g', alpha=0.7)

ax_3d_analytical.set_xlabel("Space (x)")
ax_3d_analytical.set_ylabel("Time (t)")
ax_3d_analytical.set_zlabel("u(x, t)")
ax_3d_analytical.set_title("Analytical Solution Over Time and
Space")

fig_3d_analytical.savefig("analytical_3d_solution.png", dpi=3
00)

plt.show()
```

Results

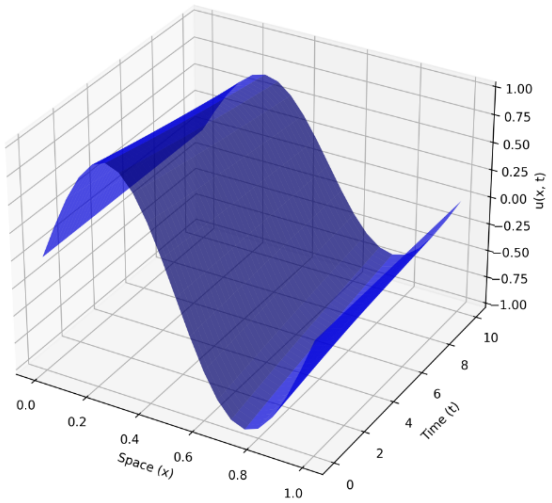




Implicit Method Solution Over Time and Space



FTCS (Explicit) Solution Over Time and Space



Analytical Solution Over Time and Space