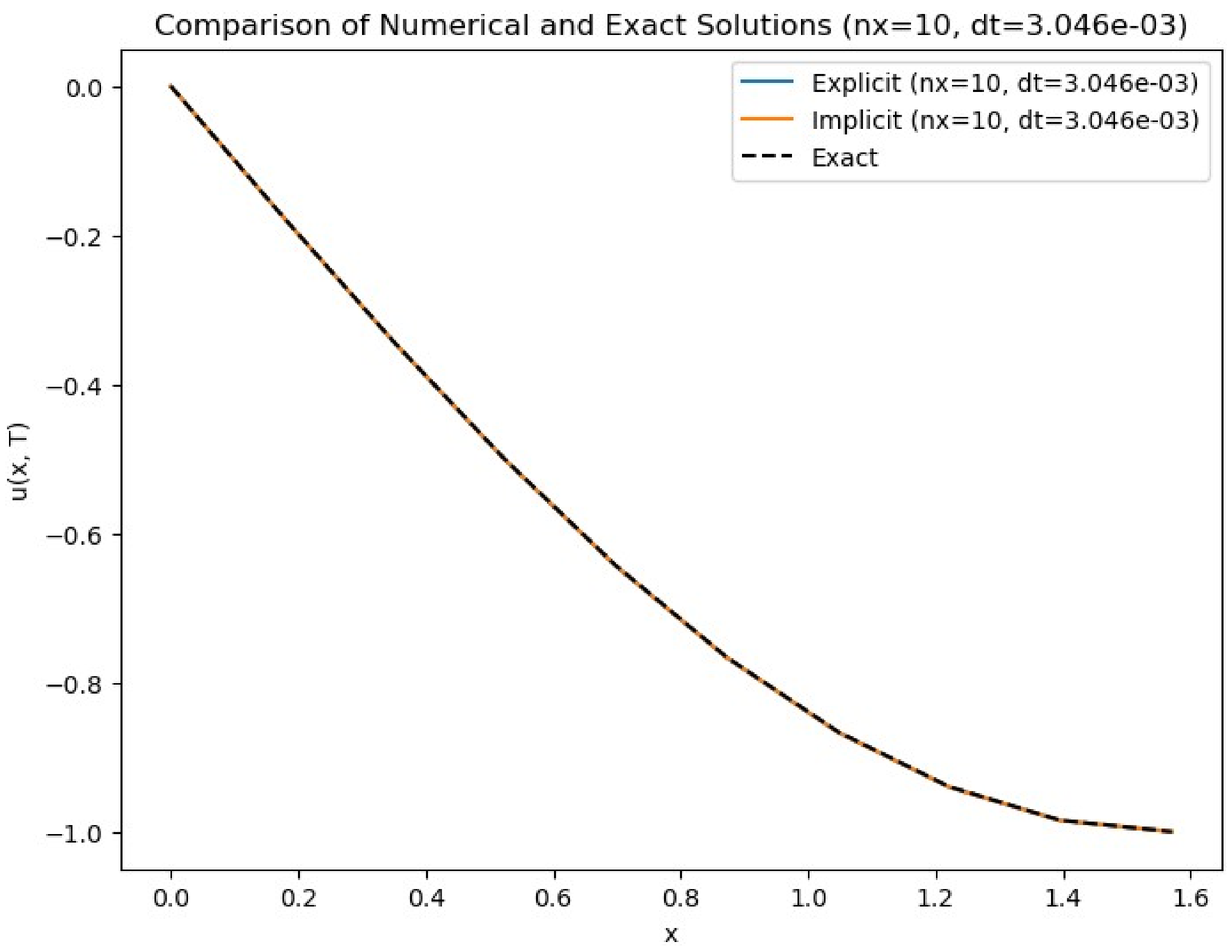
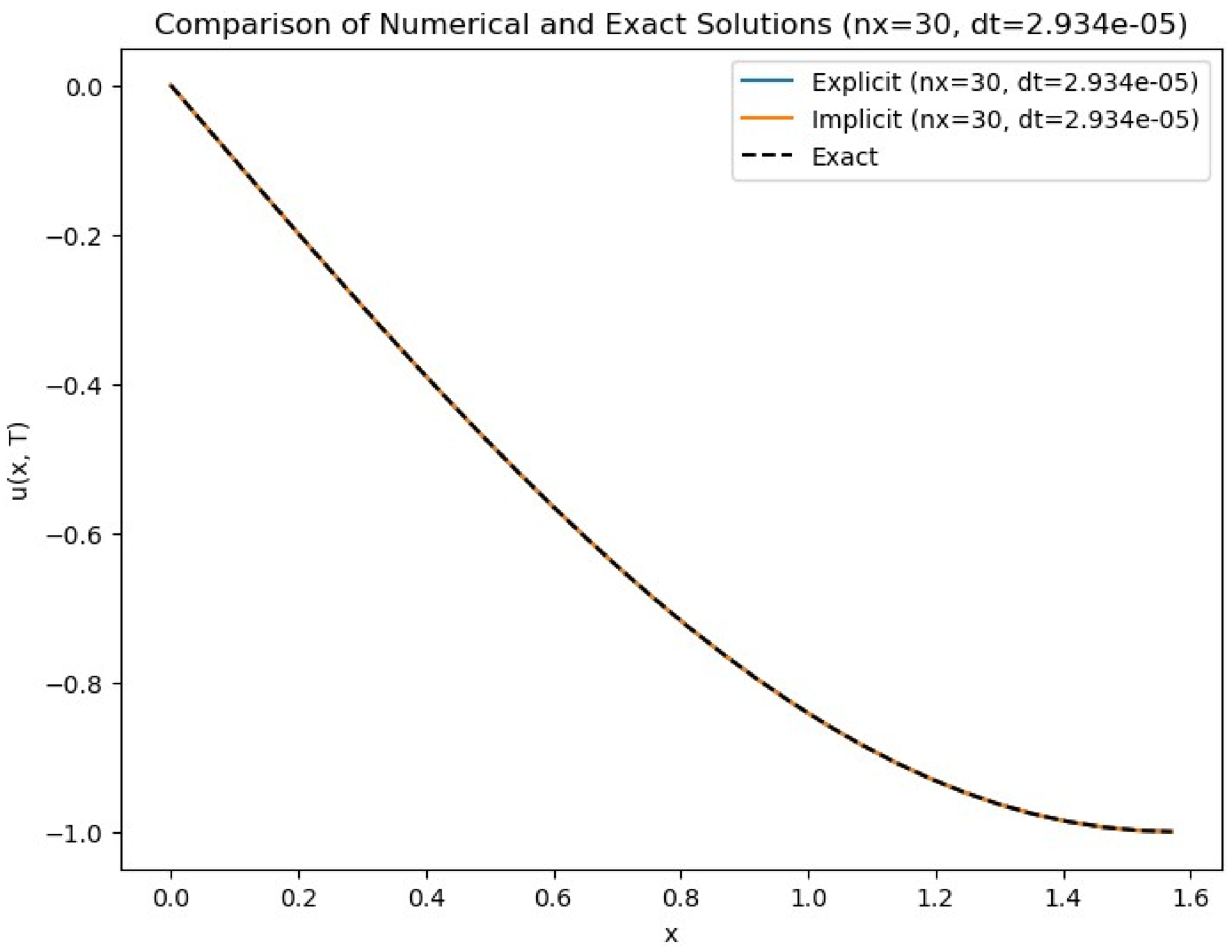
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.sparse import diags
from scipy.sparse.linalg import spsolve

# Parameters
L = np.pi / 2  # Spatial domain length
T = np.pi  # End time of the simulation

# Exact solution function for comparison
def exact_solution(x, t_val):
    return np.sin(x) * np.cos(t_val)

# Boundary conditions function
def boundary_conditions(u, t_val):
    u[0] = 0  # u(0, t) = 0
    u[-1] = np.cos(t_val)  # u(L, t) = cos(t)

# Source term function
def source_term(x, t_val):
    return -np.sin(x) * np.sin(t_val) + np.sin(x) * np.cos(t_val)

# Numerical solver
def solve_1d_heat(nx, alpha, method="explicit"):
    dx = L / (nx - 1)
    dt = alpha * dx**2  # Time step size for explicit method
    nt = int(T / dt) + 1
    dt = T / (nt - 1)  # Adjust dt to ensure exact time division

    x = np.linspace(0, L, nx)
    t = np.linspace(0, T, nt)

    u = np.zeros((nt, nx))
    u[0, :] = np.sin(x)  # Initial condition

    if method == "explicit":
        for n in range(0, nt - 1):
            boundary_conditions(u[n, :], t[n])
            for i in range(1, nx - 1):
                u[n + 1, i] = (u[n, i]
                               + alpha * (u[n, i + 1] - 2 * u[n, i] +
u[n, i - 1])
                               + dt * source_term(x[i], t[n]))
            boundary_conditions(u[n + 1, :], t[n + 1])

    elif method == "implicit":
        A = diags([-alpha, 1 + 2 * alpha, -alpha], [-1, 0, 1],
shape=(nx - 2, nx - 2)).toarray()
        for n in range(0, nt - 1):
```

```
            boundary_conditions(u[n, :], t[n])
            b = u[n, 1:-1] + dt * source_term(x[1:-1], t[n])
            b[0] += alpha * 0  # Adjust for boundary condition at left
end
            b[-1] += alpha * np.cos(t[n + 1])  # Adjust for boundary
condition at right end
            u[n + 1, 1:-1] = spsolve(A, b)
            boundary_conditions(u[n + 1, :], t[n + 1])

    return x, t, u, dx

# Initialize a table to store results
results_table = []

# Error and plotting
grid_settings = [
    {'nx': 10, 'alpha': 0.1},
    {'nx': 30, 'alpha': 0.01},
    {'nx': 50, 'alpha': 0.01},
    {'nx': 70, 'alpha': 0.01}
]

for setting in grid_settings:
    nx, alpha = setting['nx'], setting['alpha']
    x, t, u_explicit, dx = solve_1d_heat(nx, alpha, method="explicit")
    _, _, u_implicit, _ = solve_1d_heat(nx, alpha, method="implicit")

    u_exact = exact_solution(x, T)
    norm_u_exact = np.linalg.norm(u_exact)

    error_explicit = np.linalg.norm(u_explicit[-1, :] - u_exact) /
norm_u_exact
    error_implicit = np.linalg.norm(u_implicit[-1, :] - u_exact) /
norm_u_exact

    plt.figure(figsize=(8, 6))
    plt.plot(x, u_explicit[-1, :], label=f'Explicit (nx={nx},
dt={alpha*dx**2:.3e})')
    plt.plot(x, u_implicit[-1, :], label=f'Implicit (nx={nx},
dt={alpha*dx**2:.3e})')
    plt.plot(x, u_exact, 'k--', label='Exact')
    plt.xlabel('x')
    plt.ylabel('u(x, T)')
    plt.title(f'Comparison of Numerical and Exact Solutions (nx={nx},
dt={alpha*dx**2:.3e})')
    plt.legend()
    plt.show()

    # Save results to the table
    results_table.append({
```

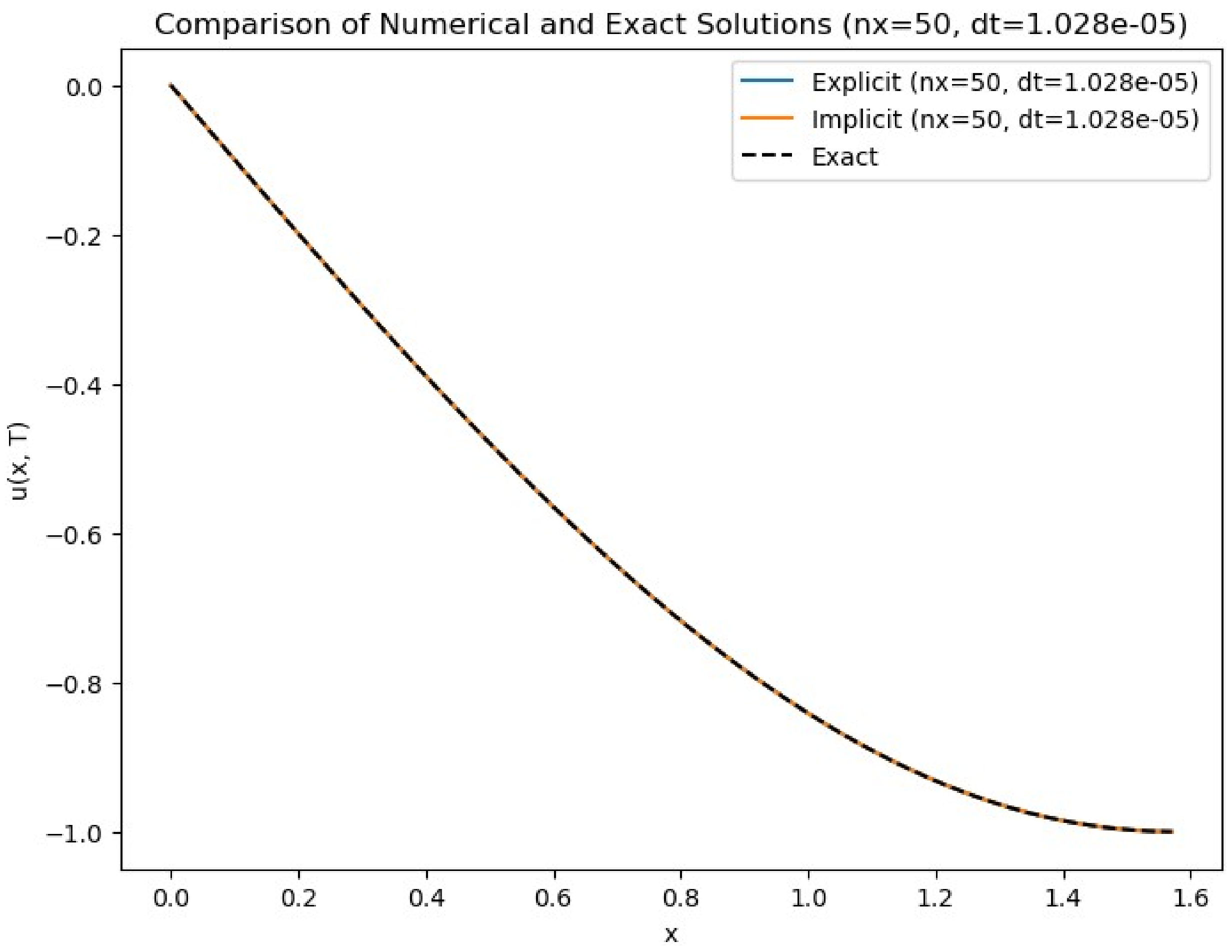
```
        "nx": nx,
        "alpha": alpha,
        "dx": dx,
        "dt": alpha * dx**2,
        "Relative Error (Explicit)": error_explicit,
        "Relative Error (Implicit)": error_implicit
    })

# Convert results to DataFrame and display
results_df = pd.DataFrame(results_table)
print(results_df)
```

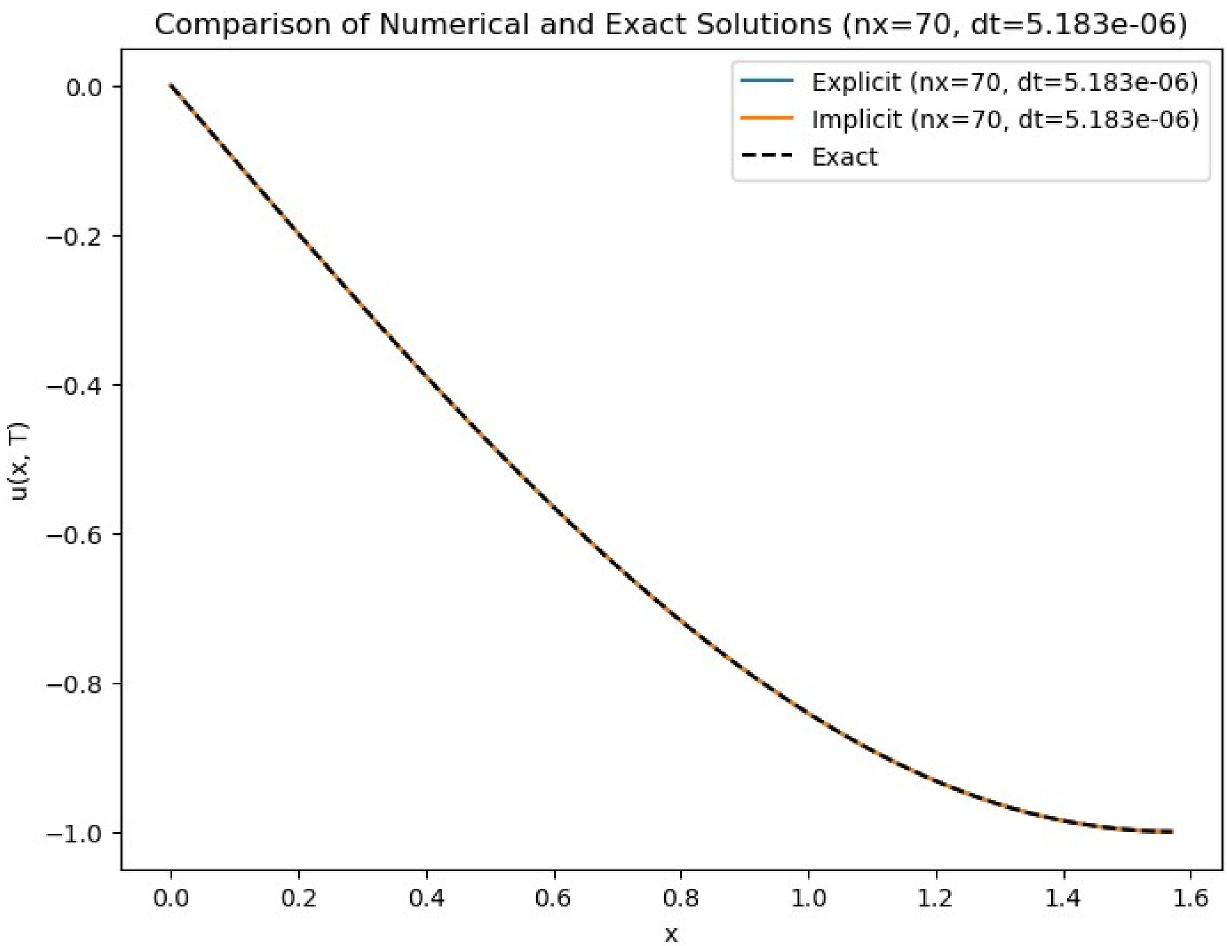
```
C:\Users\jhyang\AppData\Local\Temp\ipykernel_9248\3979193567.py:47:
SparseEfficiencyWarning: spsolve requires A be CSC or CSR matrix
format
  u[n + 1, 1:-1] = spsolve(A, b)
```





Comparison of Numerical and Exact Solutions (nx=30, dt=2.934e-05)
Explicit (nx=30, dt=2.934e-05)
Implicit (nx=30, dt=2.934e-05)
Exact
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
u(x, T)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
x


Comparison of Numerical and Exact Solutions (nx=50, dt=1.028e-05)
Explicit (nx=50, dt=1.028e-05)
Implicit (nx=50, dt=1.028e-05)
Exact
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
u(x, T)
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
x




```
   nx  alpha        dx        dt  Relative Error (Explicit)  \
0  10   0.10  0.174533  0.003046                   0.000835   
1  30   0.01  0.054165  0.000029                   0.000052   
2  50   0.01  0.032057  0.000010                   0.000019   
3  70   0.01  0.022765  0.000005                   0.000009   

   Relative Error (Implicit)  
0                   0.000688  
1                   0.000051  
2                   0.000018  
3                   0.000009  
```