AMATH 581: Report 4

Minho Choi

November 29th, 2023

Problem 3(a)

Using Forward Euler method with $\Delta t = 1/3$, we get:

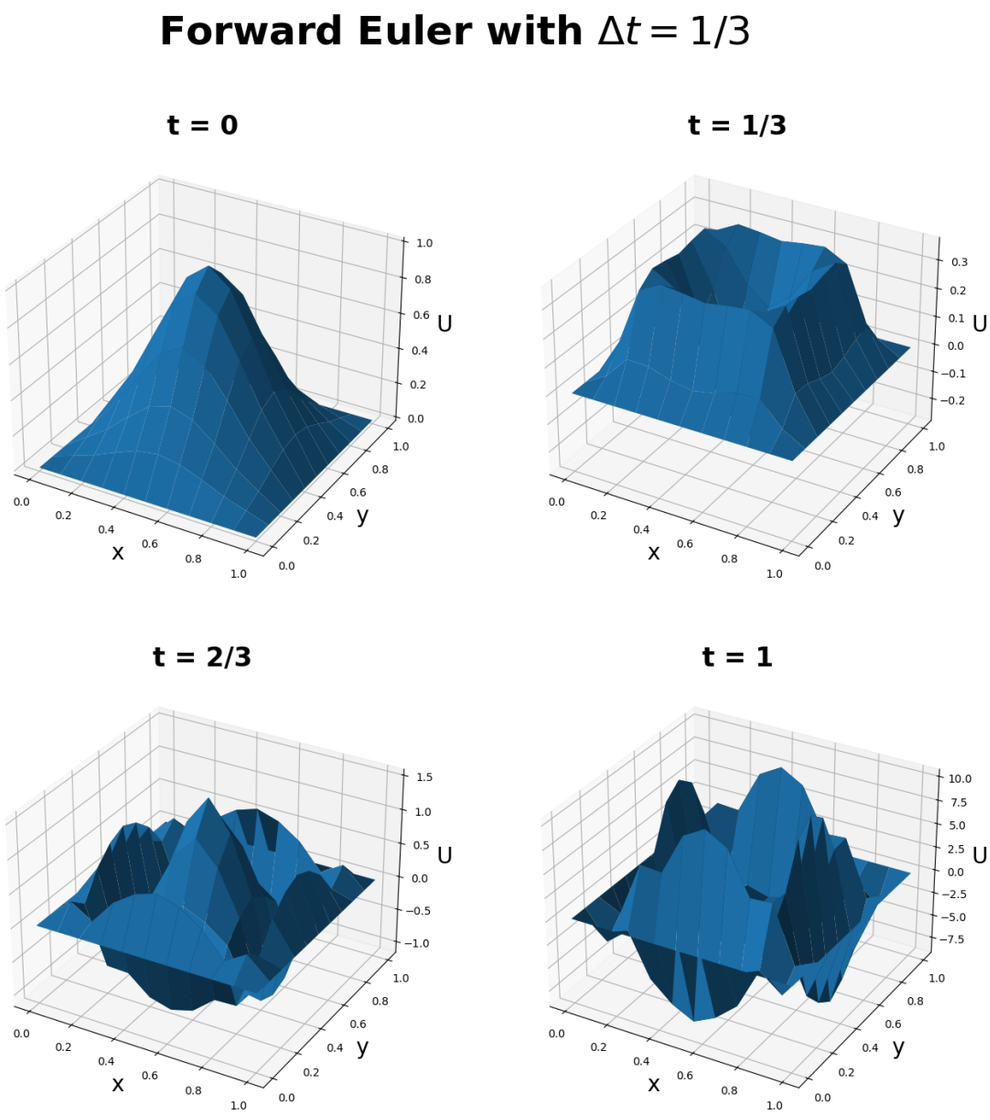

Figure 1: Forward Euler with $\Delta t = 1/3$

Problem 3(b)

Using Forward Euler method with $\Delta t = 0.01$, we get:

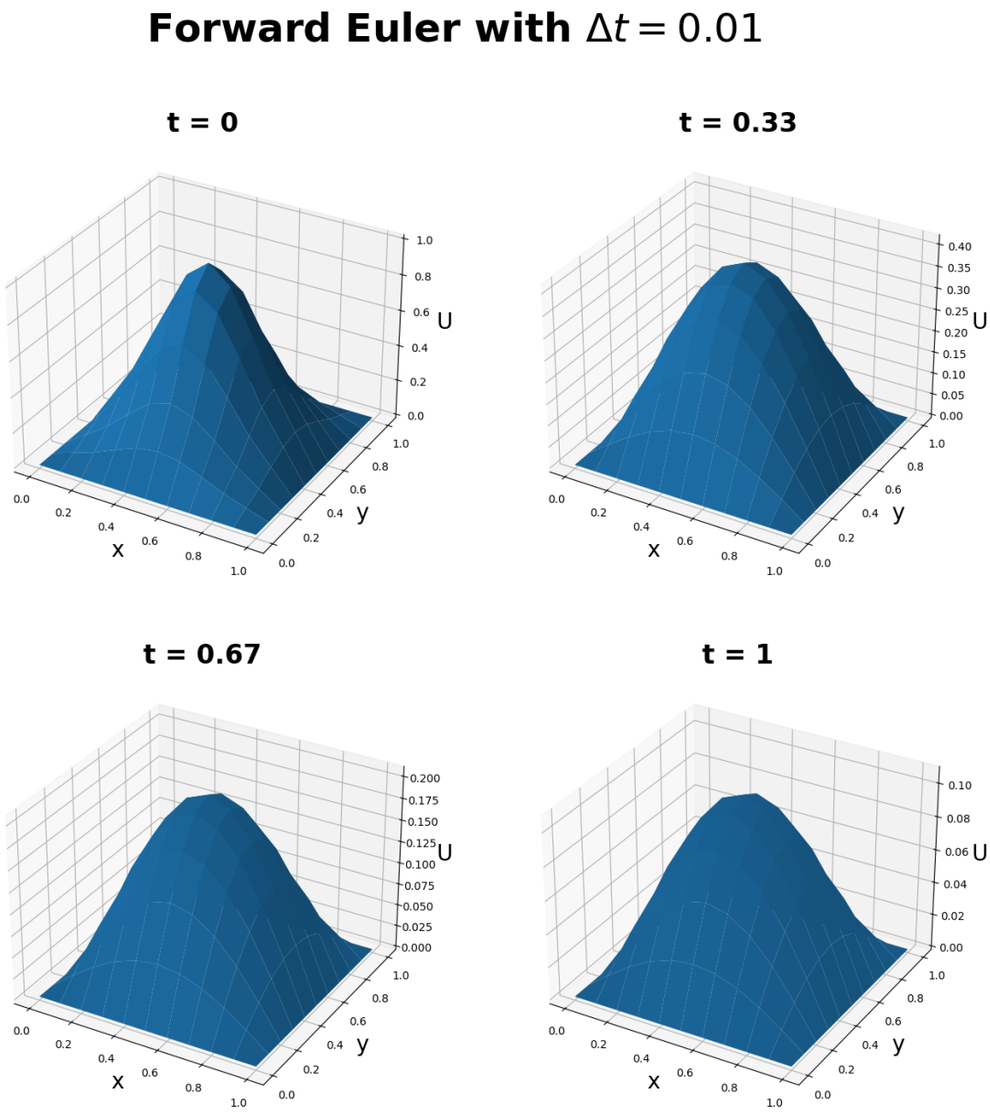

Figure 2: Forward Euler with $\Delta t = 0.01$

Problem 3(c)

Using Trapezoidal method with $\Delta t = 1/3$, we get:

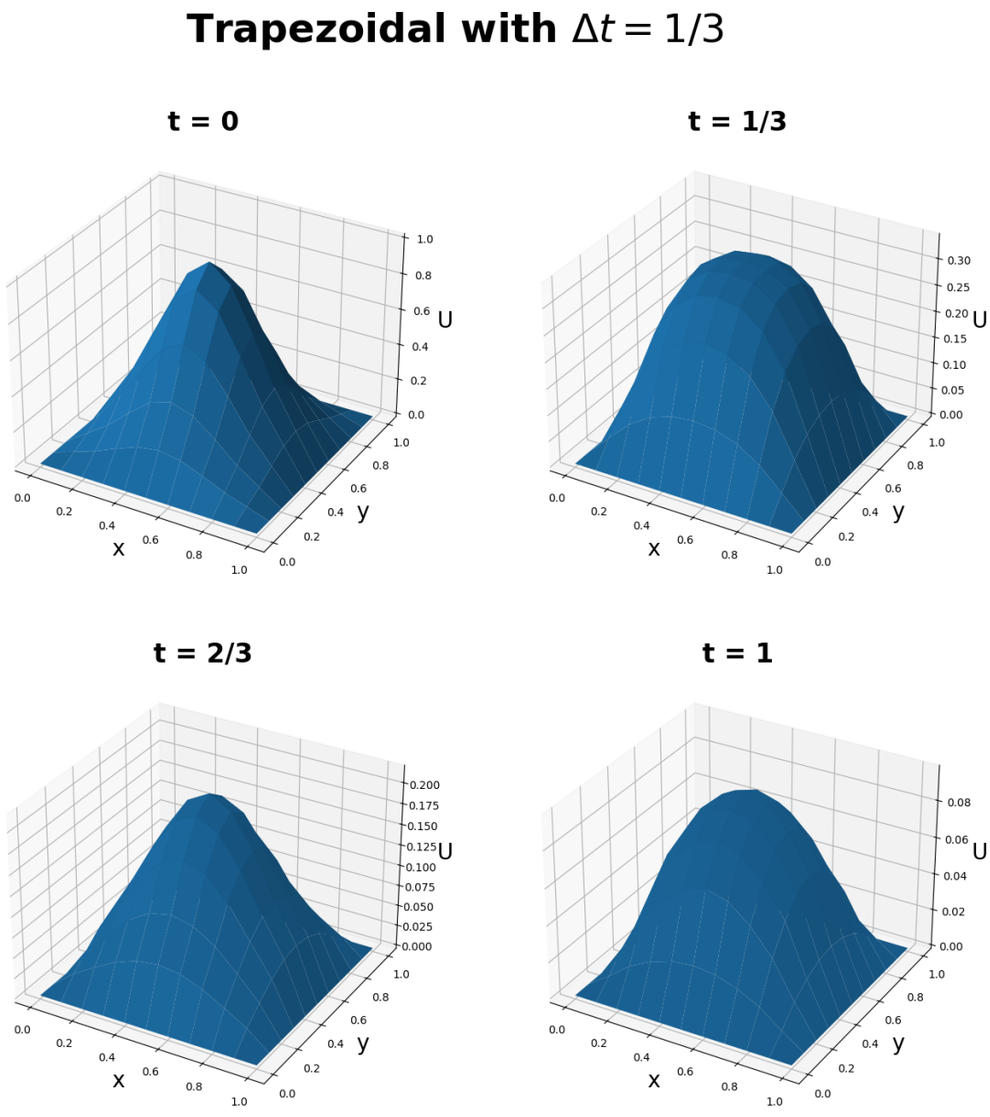

Figure 3: Trapezoidal method with $\Delta t = 1/3$

Problem 3(d)

Using Trapezoidal method with $\Delta t = 0.01$, we get:

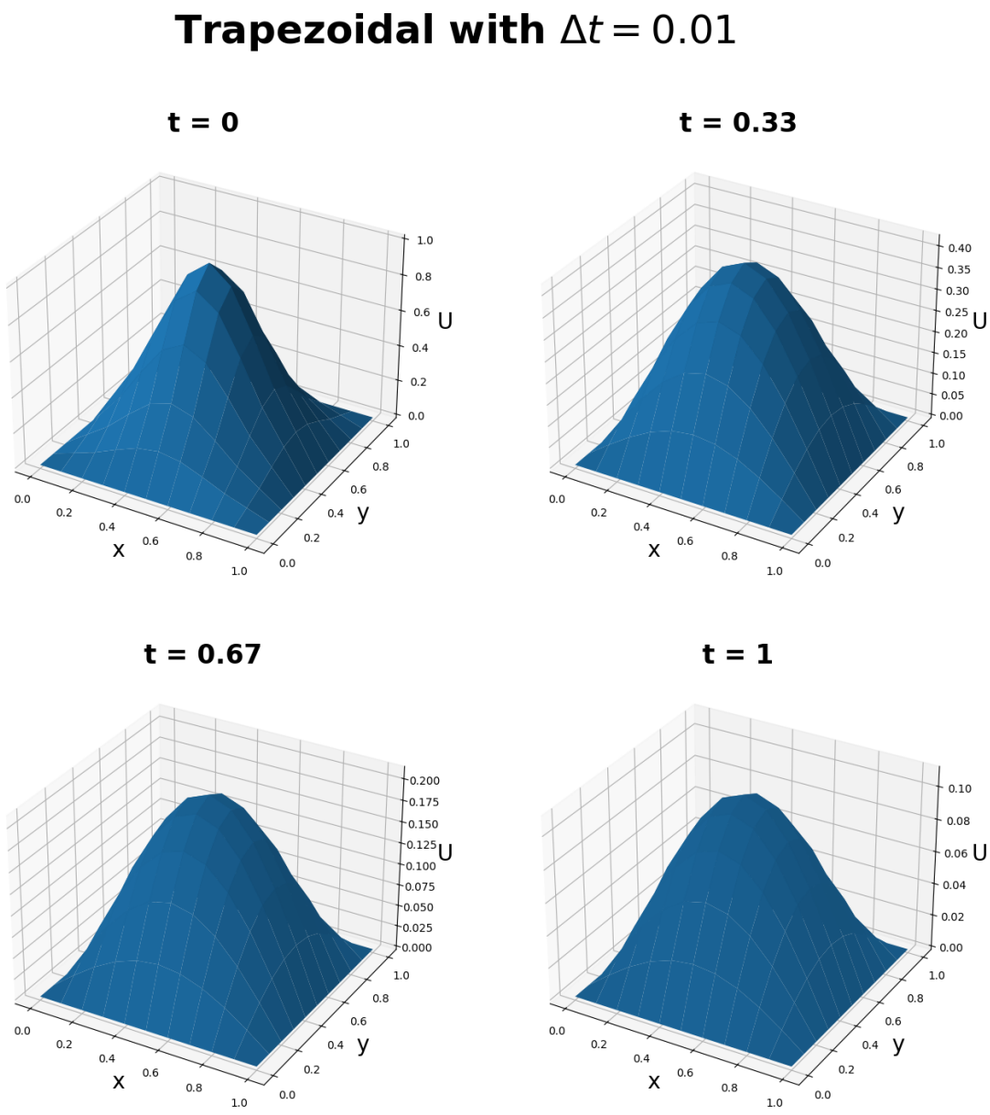

Figure 4: Trapezoidal with $\Delta t = 0.01$

Conclusion

Based on the figures, we can clearly see that the solution using Forward Euler method with $\Delta t = 1/3$ (Figure 1) exhibits instability. Note that in Figure 1, the range of U-axis increases rapidly as t value increases, showing instability.

To get a rough picture of the solutions at $t = 0$, $t \approx 1/3$, $t \approx 2/3$ and $t = 1$, Trapezoidal method is better than Forward Euler method. Trapezoidal method provides stable approximation with large Δt values, while Forward Euler method needs small Δt values to get stable approximation. Hence, to obtain stable approximation (a rough idea of how solution changes over time), Trapezoidal method is better than Forward Euler method. However, Forward Euler method is computed simply using matrix multiplications and vector additions, while Trapezoidal method contains solving a linear system for each step. As a consequence, for smaller Δx and Δy values, Forward Euler method can perform faster than Trapezoidal method. Therefore, in general, Trapezoidal method will give better picture of the solutions at different times, but with very small Δx and Δy values, Trapezoidal method may take significantly longer time to obtain a stable approximation than Forward Euler method.

Appendix

Appendix 1: Code for boundary conditions and positioning

```
kappa = 0.1
x0 = 0
xf = 1
y0 = 0
yf = 1
t0 = 0
tf = 1
N = 11

def bc_t0(x, y):
    if x == 0 or y == 0 or x == 1 or y == 1:
        return 0
    return np.exp(-10 * ((x - 0.5) ** 2 + (y - 0.5) ** 2))

def point2ind(m, n):
    return (n - 1) * (N - 2) + m - 1
```

Appendix 2: Code for Forward Euler method

```
def forward_euler3(Nt):
    x = np.linspace(x0, xf, N)
    y = np.linspace(y0, yf, N)
    dx = x[1] - x[0]
    t = np.linspace(t0, tf, Nt)
    dt = t[1] - t[0]

    N_total = (N - 2) * (N - 2)
    A = np.zeros((N_total, N_total))
    U_int = np.zeros((N_total, Nt))

    for n in range(1, N-1):
        for m in range(1, N-1):
            k = point2ind(m, n)
            A[k, k] = -4 * kappa / dx ** 2
            U_int[k, 0] = bc_t0(x[m], y[n])
            if m > 1:
                A[k, k - 1] = 1 * kappa / dx ** 2
            if n < N - 2:
                A[k, k + N - 2] = 1 * kappa / dx ** 2
            if m < N - 2:
                A[k, k + 1] = 1 * kappa / dx ** 2
            if n > 1:
                A[k, k - (N - 2)] = 1 * kappa / dx ** 2

    # Solve with Forward Euler
```

```
    for k in range(Nt - 1):
        U_int[:, (k + 1):(k + 2)] = U_int[:, k:(k + 1)] + dt * (A @ U_int
    [:, k:(k + 1)])

    U = np.zeros((N, N, Nt))
    U[1:(N-1), 1:(N-1), :] = U_int.reshape((N-2, N-2, Nt))

    return U[5, 5, -1], U
```

Appendix 3: Code for Trapezoidal method

```
def trapezoidal3(Nt):
    x = np.linspace(x0, xf, N)
    y = np.linspace(y0, yf, N)
    dx = x[1] - x[0]
    t = np.linspace(t0, tf, Nt)
    dt = t[1] - t[0]

    N_total = (N - 2) * (N - 2)
    A = np.zeros((N_total, N_total))
    U_int = np.zeros((N_total, Nt))

    for n in range(1, N-1):
        for m in range(1, N-1):
            k = point2ind(m, n)
            A[k, k] = -4 * kappa / dx ** 2
            U_int[k, 0] = bc_t0(x[m], y[n])
            if m > 1:
                A[k, k - 1] = 1 * kappa / dx ** 2
            if n < N - 2:
                A[k, k + N - 2] = 1 * kappa / dx ** 2
            if m < N - 2:
                A[k, k + 1] = 1 * kappa / dx ** 2
            if n > 1:
                A[k, k - (N - 2)] = 1 * kappa / dx ** 2

    # Solve with Trapezoidal Method
    I = np.eye(N_total)
    A_lhs = (I - (dt / 2) * A)
    A_rhs = (I + (dt / 2) * A)
    for k in range(Nt - 1):
        U_int[:, (k + 1):(k + 2)] = np.linalg.solve(A_lhs, A_rhs @ U_int
    [:, k:(k + 1)])

    U = np.zeros((N, N, Nt))
    U[1:(N-1), 1:(N-1), :] = U_int.reshape((N-2, N-2, Nt))

    return U[5, 5, -1], U
```