Christian DiPietrantonio
ME M311: Computational Methods to Viscous Flows
February 19, 2026

Computer Assignment 01: Numerical and Analytical Solutions to Parabolic Partial Differential Equations: prerequisite for solving boundary layer problems

1. Description of the Problem

Given the following second-order linear parabolic partial differential equation (PDE):

$$\frac{\partial u}{\partial x} - 2\frac{\partial^2 u}{\partial y^2} = 2 \tag{1}$$

with boundary conditions: $u\,(x,0) = 0$, $u\,(x,1) = 0$, and initial condition: $u(0,y) = 0$, the objectives of this project are to:

1. Derive the analytical solution.
2. Use the Crank-Nicholson and central difference schemes to discretize the equation.
3. Find the numerical solution and compare it with the analytical solution using LU decomposition as the linear system solver.

Parabolic PDEs, such as the one above, can be used to describe heat conduction and viscous boundary layers, making this problem a useful foundation for later work.

2. Derivation of the Analytical Solution

As mentioned in the supplemental materials, Equation 1 is inhomogeneous. To solve it, one must use the superposition and separation of variables techniques. First, the solution is expressed as $u(x,y) = v(x,y) + f(y)$, and the original equation is re-written as:

$$\frac{\partial v}{\partial x} - 2\frac{\partial^2 v}{\partial y^2} - 2\frac{\partial^2 f}{\partial y^2} = 2 \tag{2}$$

By setting $\frac{\partial^2 f}{\partial y^2} = -1$, the equation above can be re-written as:

$$\frac{\partial v}{\partial x} - 2\frac{\partial^2 v}{\partial y^2} = 0 \tag{3}$$

The separation of variables technique can now be used to solve the equation above. To start, one can assume that $v(x,y) = X(x)Y(y)$, therefore:

$$X'(x)Y(y) - 2X(x)Y''(y) = 0 \tag{4}$$

Dividing both sides by $X(x)Y(y)$, and rearranging gives:

$$\frac{X'(x)}{X(x)} = 2\frac{Y''(y)}{Y(y)} = -\lambda \tag{5}$$

Equation 5 can then be separated into two ordinary differential equations:

$$X'(x) + \lambda X(x) = 0 \tag{6}$$

$$Y''(y) + \frac{\lambda}{2}Y(y) = 0 \tag{7}$$

From $u(x,0)=0$, $f(0)=0$ and $v(x,0)=0$. From $u(x,1)=0$, $f(1)=0$ and $v(x,1)=0$. Finally, from $u(0,y)=0$, $v(0,y)=-f(y)$. Therefore, the boundary conditions for $Y(y)$ are $Y(0)=0$ and $Y(1)=0$. Now, define $\alpha^2=\frac{\lambda}{2}$, therefore Equation 7 can be re-written as:

$$Y''(y)+\alpha^2 Y(y)=0 \tag{8}$$

The general solution to which is:

$$Y(y)=A\cos(\alpha y)+B\sin(\alpha y) \tag{9}$$

Applying the boundary condition $Y(0)=0$ gives $A=0$. Therefore, $Y(y)=B\sin(\alpha y)$. Applying the boundary condition $Y(1)=0$ gives $Y(1)=B\sin(\alpha)=0$. A non-trivial solution requires:

$$\sin(\alpha)=0 \implies \alpha=n\pi, \quad n=1,2,3,... \tag{10}$$

Therefore:

$$Y_n(y)=B_n\sin(n\pi y) \tag{11}$$

Recall that $\alpha^2=\frac{\lambda}{2}$, therefore $\lambda_n=2(n\pi)^2$. Thus, Equation 6 can be expressed as:

$$X'(x)+2(n\pi)^2X(x)=0 \tag{12}$$

The general solution to which is:

$$X_n(x)=A_ne^{-2(n\pi)^2x} \tag{13}$$

Therefore, the solution to the homogeneous equation is:

$$v(x,y)=\sum_{n=1}^{\infty}A_ne^{-2(n\pi)^2x}\sin(n\pi y) \tag{14}$$

From $\frac{\partial^2 f}{\partial y^2}=-1$ and the boundary conditions $f(0)=0$ and $f(1)=0$, $f(y)=\frac{y}{2}(1-y)$. Again, at $u(0,y)=0$, $v(0,y)=-f(y)$. Therefore:

$$\sum_{n=1}^{\infty}A_n\sin(n\pi y)=-\frac{y}{2}(1-y) \tag{15}$$

Which can be recognized as a Fourier sine series on the interval $[0,1]$, therefore:

$$A_n=-2\int_0^1\frac{y}{2}(1-y)\sin(n\pi y)dy=-\int_0^1 y(1-y)\sin(n\pi y)dy \tag{16}$$

Evaluating this integral gives:

$$A_n=\frac{2\left((-1)^n-1\right)}{(n\pi)^3} \tag{17}$$

For even values of n, $A_n=0$. For odd values of n, $A_n=-\frac{4}{(n\pi)^3}$. So Equation 15 becomes:

$$v(x,y)=\sum_{n=1,3,5,...}^{\infty}-\frac{4}{(n\pi)^3}e^{-2x(n\pi)^2}\sin(n\pi y) \tag{18}$$

And finally, the analytical solution of the parabolic equation can be expressed as:

$$\boxed{u(x,y) = \sum_{n=1,3,5,\ldots}^{\infty} -\frac{4}{(n\pi)^3} e^{-2x(n\pi)^2} \sin(n\pi y) + \frac{y}{2}(1-y)} \tag{19}$$

3. Description of the Numerical Method

To discretize Equation 1, the entire equation is integrated over the specified x interval, Δx, and the control volume, V:

$$\iiint\limits_V \left(\int_x^{x+\Delta x} \frac{\partial u}{\partial x}\, dx \right) dV = \int_x^{x+\Delta x} \left(\iiint\limits_V \left(2\frac{\partial^2 u}{\partial y^2} + 2 \right) dV \right) dx \tag{20}$$

For the left-hand side, integrating over x gives:

$$\iiint\limits_V \left(u(x+\Delta x) - u(x)\right)\, dV = \left(u(x+\Delta x) - u(x)\right)\Delta y = \Delta y \left({u_i}^{n+1} - {u_i}^{n}\right) \tag{21}$$

On the right-hand side, the divergence theorem is used to convert the integral over the control volume into a surface integral:

$$\oiint\limits_A \left(2\frac{\partial u}{\partial y}\hat{n}_y \right) dA + 2\Delta y \tag{22}$$

After evaluating the surface integral, resulting derivatives are discretized using a central-difference scheme:

$$2\left(\frac{u_{i+1} - u_i}{\Delta y}\right) - 2\left(\frac{u_i - u_{i-1}}{\Delta y}\right) + 2\Delta y \tag{23}$$

Thus, the right-hand side reduces to:

$$\int_x^{x+\Delta x} \left(2\left(\frac{u_{i+1} - u_i}{\Delta y}\right) - 2\left(\frac{u_i - u_{i-1}}{\Delta y}\right) + 2\Delta y \right) dx \tag{24}$$

Next, the integration $\int_x^{x+\Delta x} u\, dx$ is approximated using:

$$\int_x^{x+\Delta x} u dx = \left(\theta u^n + (1-\theta)u^{n+1}\right)\Delta x \tag{25}$$

For the Crank-Nicholson scheme ($\theta = 0.5$), Equation 24 becomes:

$$\frac{\Delta x}{\Delta y}\left(\left({u_{i+1}}^n + {u_{i+1}}^{n+1}\right) - 2\left({u_i}^n + {u_i}^{n+1}\right) + \left({u_{i-1}}^n + {u_{i-1}}^{n+1}\right)\right) + 2\Delta y \Delta x \tag{26}$$

Finally, by equating the left and right-hand sides and dividing both by Δy, the following discretized governing equation for any interior cell ($i = 1, 2, ..., N-2$), is obtained:

$$\boxed{{u_i}^{n+1} - {u_i}^n = \frac{\Delta x}{\Delta y^2}\left(\left({u_{i+1}}^n + {u_{i+1}}^{n+1}\right) - 2\left({u_i}^n + {u_i}^{n+1}\right) + \left({u_{i-1}}^n + {u_{i-1}}^{n+1}\right)\right) + 2\Delta x} \tag{27}$$

For the boundary at $y = 0$, where $u(x, 0) = 0$, the discretized governing equation (assuming the cell ID for the first cell is $i = 0$) is:

$$\boxed{{u_0}^{n+1} - {u_0}^{n} = \frac{\Delta x}{\Delta y^2}\left(\left({u_1}^{n} + {u_1}^{n+1}\right) - 3\left({u_0}^{n} + {u_0}^{n+1}\right)\right) + 2\Delta x} \tag{28}$$

For the boundary at $y = 1$, where $u(x, 1) = 0$, the discretized governing equation (assuming the cell ID for the last cell is $i = N - 1$) is:

$$\boxed{{u_{N-1}}^{n+1} - {u_{N-1}}^{n} = \frac{\Delta x}{\Delta y^2}\left(\left({u_{N-2}}^{n} + {u_{N-2}}^{n+1}\right) - 3\left({u_{N-1}}^{n} + {u_{N-1}}^{n+1}\right)\right) + 2\Delta x} \tag{29}$$

The numerical solver shown in Appendix A can now be programmed (using LU decomposition as the linear system solver) to solve the discretized governing equations above and compare the results with the analytical solution derived in the previous section.

4. Presentation of Results

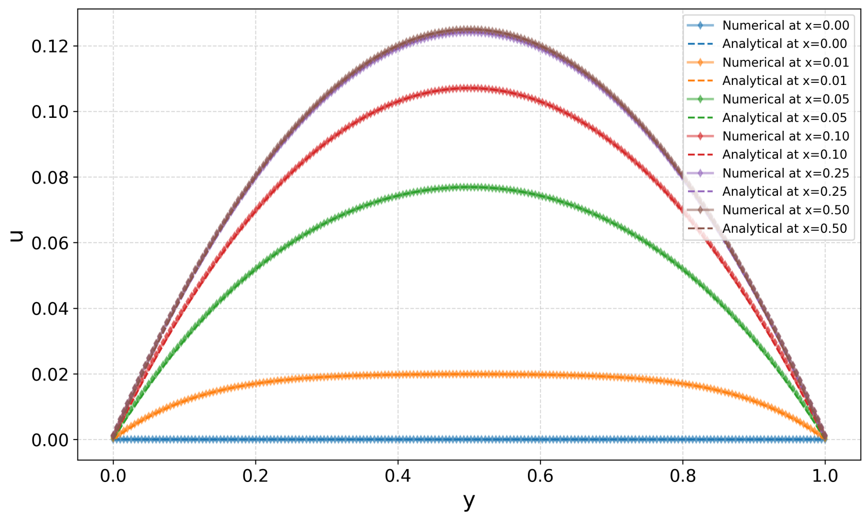

Figure 1: Comparison of numerical and analytical solution profiles at selected x-locations.

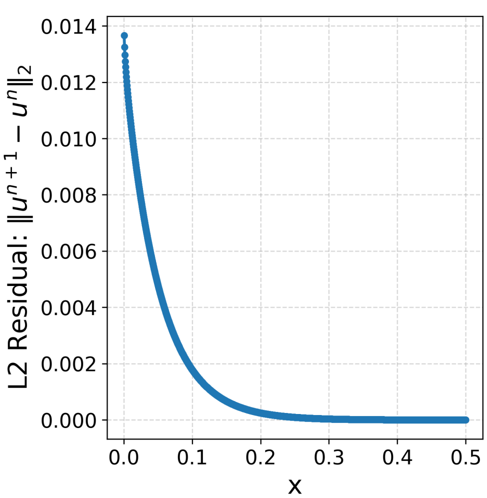

Figure 2: L2 residual vs. x

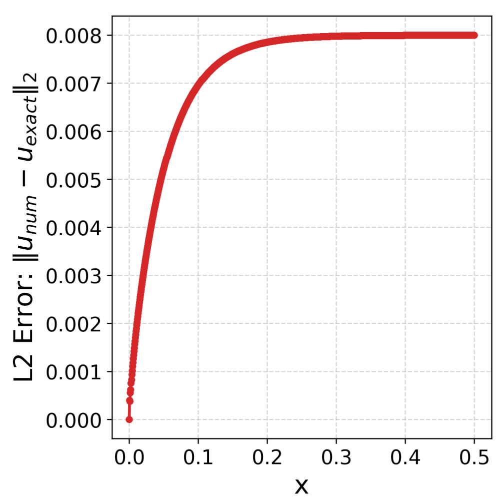

Figure 3: L2 error between numerical and analytical solutions vs. x

5. Discussion of Results

5.1. General description

The numerical solution of the parabolic equation obtained using the Crank-Nicholson and central difference schemes exhibits excellent agreement with the analytical solution derived in Section 2. Figure 1 compares numerical and analytical profiles of u at several x-locations, and for all values of x, the two curves lie nearly on top of one another. This demonstrates that the discretization accurately captures both the transient development in the x-direction and the parabolic shape in the y-direction. The analytical solution contains an exponentially decaying transient term proportional to $e^{-2x\pi^2}$, and the solution approaches the steady state profile

$$u(y) = \frac{y}{2}(1 - y) \tag{30}$$

as x increases. At roughly $x = 0.25$, the transient component of the solution has decayed to about 1% of its initial magnitude, and the solution is effectively steady. This behavior is confirmed in the residual plot (Figure 2), as the L2 residual initially decreases rapidly, then remains unchanged at a value of roughly zero, indicating no meaningful change between successive steps in x. Finally, the parabolic shape the solution approaches as x increases is physically intuitive, because each cell contributes a constant source term of $2\Delta x$, while the boundary conditions enforce $u = 0$ at $y = 0$ and $y = 1$. Therefore, the cells further from the boundaries will have larger values of u, because the diffusive flux in the y-direction must travel a greater distance before reaching the boundaries. As a result, the gradient of u decreases as the distance from the boundaries increases, so diffusion removes u more slowly.

5.2. Accuracy and stability

The numerical and analytical profiles at each x-location are nearly indistinguishable, indicating high accuracy. Figure 3 shows the error at every x-step, from which one can observe that the error does not increase past 8×10^{-3}, thus demonstrating the accuracy of the numerical solution across the entire domain. Initially, significant fluctuations in the error were observed in the first few time steps of the simulation. This occurred because the step size in x was too large relative to the rapid evolution of the transient. Decreasing Δx reduced these fluctuations at the cost of increasing the computational power required for the simulation. Nevertheless, one can now observe that there are no significant oscillations, instabilities, or unphysical values in the solution profiles (Figure 1) or the error curve (Figure 3). This is to be expected, as the Crank-Nicholson scheme is unconditionally stable. Finally, the combination of small errors and decaying residuals confirms the numerical scheme is both accurate and stable for this problem.

A. Copy of Program Listing

```python
# Christian DiPietrantonio
# ME 5311: Computational Methods to Viscous Flows
# Computer Assignment 01
# 02/19/2026

import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import lu_factor, lu_solve

# ------------------------------------------------------------------
# 1. Parameters and Mesh Sizing
# ------------------------------------------------------------------

def set_parameters():
    params = {}

    # Domain in y
    params["y_min"] = 0
    params["y_max"] = 1
    params["N"]     = 200  # Number of cells (finite volume method)
    params["dy"]    = (params["y_max"] - params["y_min"]) / params["N"]  # Cell size in y

    # Stepping in x
    params["x_max"] = 0.5 # Final x
    params["Nx"]    = 1000 # Number of steps in x
    params["dx"]    = params["x_max"] / params["Nx"] # Step size in x

    # Source term
    params["S"] = 2.0

    # y-array for plotting and analytical solution evaluation
    params["y"] = np.linspace(params["y_min"], params["y_max"], params["N"])

    # Initial condition u(y,0) = 0
    u0 = np.zeros(params["N"])

    return params, u0

# ------------------------------------------------------------------
# 2. Build Matrices
# ------------------------------------------------------------------

def build_matrices(params):
```

```
    N = params["N"]
    dx = params["dx"]
    dy = params["dy"]

    r = dx / dy**2 # Defined to simplify matrix coefficients

    A = np.zeros((N, N))
    B = np.zeros((N, N))

    # ----- Bottom Boundary: i = 0, u(x,0) = 0 ---------------------
    #LHS (coefficients from Crank-Nicholson discretization)
    A[0, 0] = 1 + 3*r
    A[0, 1] = -r

    #RHS (coefficients from Crank-Nicholson discretization)
    B[0, 0] = 1 - 3*r
    B[0, 1] = r

    # ----- Interior Cells: i = 1,..., N-2 -------------------------
    for i in range(1, N-1):
        #LHS
        A[i, i-1] = -r
        A[i, i]   = 1 + 2*r
        A[i, i+1] = -r

        #RHS
        B[i, i-1] = r
        B[i, i]   = 1 - 2*r
        B[i, i+1] = r

    # ----- Top Boundary: i = N-1, u(x,1) = 0 ----------------------
    #LHS
    A[N-1, N-2] = -r
    A[N-1, N-1] = 1 + 3*r

    #RHS
    B[N-1, N-2] = r
    B[N-1, N-1] = 1 - 3*r

    return A, B

# ------------------------------------------------------------------
# 3. Build RHS vector for a given u^n
# ------------------------------------------------------------------

def build_rhs(u_n, params, B):
    dx = params["dx"]
```

```python
    S   = params["S"]
    N   = params["N"]

    rhs = B @ u_n

    # Add source term contribution
    rhs += S * dx * np.ones(N)

    return rhs

# ------------------------------------------------------------------
# 4. Step in x using LU decomposition
# ------------------------------------------------------------------

def step_in_x(params, u0):
    A, B = build_matrices(params)

    # LU factorization of A
    lu, piv = lu_factor(A)

    u = u0.copy()
    N = params["N"]
    Nx = params["Nx"]

    # Initialize storage for x-location and solution profiles
    x_vals = np.zeros(Nx + 1)
    u_store = np.zeros((Nx + 1, N))

    for n in range(Nx + 1):
        x_now = n * params["dx"]
        x_vals[n] = x_now
        u_store[n, :] = u.copy()

        # Break after storing solution at final x-location to avoid
           an extra solve
        if n == Nx:
            break

        rhs = build_rhs(u, params, B)
        u = lu_solve((lu, piv), rhs)

    return x_vals, u_store

# ------------------------------------------------------------------
# 5. Post processing
# ------------------------------------------------------------------

```

```python
def analytical_solution(x, y, n_terms=100):
    """Compute the analytical solution (Eq. 19) of the parabolic PDE"""

    u_exact = 0.5 * y * (1 - y)

    for k in range(n_terms):
        n = 2*k + 1 # sum over odd integers only
        A_n = -4 / (n * np.pi)**3
        decay = np.exp(-2 * x * (n * np.pi)**2)
        u_exact += A_n * decay * np.sin(n * np.pi * y)

    return u_exact

def post_process(params, x_vals, u_store):
    y = params["y"]

    x_targets = [0.0, 0.01, 0.05, 0.1, 0.25, 1.0]

    errors       =[]
    x_error_vals = []

    # Plot 1: Numerical vs Analytical Solutions at selected x locations
    plt.figure(figsize=(10, 6))

    colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab:purple', 'tab:brown']

    for i, x_target in enumerate(x_targets):
        idx = np.argmin(np.abs(x_vals - x_target))
        x_now = x_vals[idx]
        u_num = u_store[idx, :]
        u_exact = analytical_solution(x_now, y)

        error = np.linalg.norm(u_num - u_exact, ord=2)
        errors.append(error)
        x_error_vals.append(x_now)

        plt.plot(y, u_num, '-d', color=colors[i], alpha=0.5, label=f'Numerical at x={x_now:.2f}', linewidth=2, markersize=4)
        plt.plot(y, u_exact, '--', color=colors[i], label=f'Analytical at x={x_now:.2f}', linewidth=1.5)

    plt.xlabel('y', fontsize=18)
    plt.ylabel('u', fontsize=18)
    plt.xticks(fontsize=14)
```

```python
    plt.yticks(fontsize=14)
    plt.grid(True, linestyle='--',alpha=0.5)
    plt.legend()
    plt.tight_layout()
    plt.savefig('profiles_comparison.png', dpi=300, bbox_inches='
        tight')

    # Plot 2/3: Residual, Error vs x
    all_errors = []
    residuals = []

    for n in range(len(x_vals)):
        # Error
        u_num = u_store[n, :]
        u_exact = analytical_solution(x_vals[n], y)
        all_errors.append(np.linalg.norm(u_num - u_exact, ord=2))

        # Residual
        if n < len(x_vals) - 1: # skip last point since we don't
            have u at x_{n+1}
            u_next = u_store[n+1, :]
            residuals.append(np.linalg.norm(u_next - u_num, ord=2))

    # Plot 2: residual vs x
    plt.figure(figsize=(5, 5))
    plt.plot(x_vals[1:], residuals, '-o', color = 'tab:blue',
        linewidth=2, markersize=4)
    plt.xlabel('x', fontsize=18)
    plt.ylabel(r'L2 Residual: $\|u^{n+1} - u^n\|_2$', fontsize=18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.savefig('residual_vs_x.png', dpi=300, bbox_inches='tight')

    # Plot 3: error vs x
    plt.figure(figsize=(5, 5))
    plt.plot(x_vals, all_errors, '-o', color='tab:red', linewidth=2,
         markersize=4)
    plt.xlabel('x', fontsize=18)
    plt.ylabel(r'L2 Error: $\|u_{num} - u_{exact}\|_2$', fontsize
        =18)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.grid(True, linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.savefig('error_vs_x_all.png', dpi=300, bbox_inches='tight')
```

```python

# ------------------------------------------------------------------
# 5. Main Driver
# ------------------------------------------------------------------

def main():
    params, u0 = set_parameters()
    x_vals, u_store = step_in_x(params, u0)
    post_process(params, x_vals, u_store)

if __name__ == "__main__":
    main()
```