

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):

$$\frac{\partial u}{\partial x} - 2 \frac{\partial^2 u}{\partial y^2} = 2 \quad (1)$$

With boundary conditions: $u(x, 0) = 0$, $u(x, 1) = 0$, and initial condition: $u(0, y) = 0$, our objectives were to:

1. Derive the analytical solution of the parabolic equation with its given initial and boundary conditions.
2. Use the Crank-Nicolson scheme and central difference scheme to discretize the equation (using either a finite-volume or a finite-difference based method).
3. Find the numerical solution of the parabolic equation and compare it with the analytical solution using LU decomposition as the linear system solver.

The results of these objectives will be of use for further projects, as parabolic partial differential equations like the one above can be used to describe heat conduction and viscous boundary layers.

2. Derivation of the Analytical Solution

As mentioned in the supplemental materials, the equation $\frac{\partial u}{\partial x} - 2 \frac{\partial^2 u}{\partial y^2} = 2$ is inhomogenous, therefore to solve it one must use superposition in addition to the conventional separation of variables technique. The solution, u is expressed as $u(x, y) = v(x, y) + f(y)$, and the original equation is re-written as:

$$\frac{\partial v}{\partial x} - 2 \frac{\partial^2 v}{\partial y^2} - 2 \frac{\partial^2 f}{\partial y^2} = 2 \quad (2)$$

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

$$\frac{\partial v}{\partial x} - 2 \frac{\partial^2 v}{\partial y^2} = 0 \quad (3)$$

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:

$$X'(x)Y(y) - 2X(x)Y''(y) = 0 \quad (4)$$

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

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

The equation above can then be separated into two ordinary differential equations:

$$X'(x) + \lambda X(x) = 0 \quad (6)$$

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

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 the equation for $Y(y)$ can be re-written as:

$$Y''(y) + \alpha^2 Y(y) = 0 \quad (8)$$

The general solution to which is:

$$Y(y) = A \cos(\alpha y) + B \sin(\alpha y) \quad (9)$$

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 $Y(1) = B \sin(\alpha) = 0$. For a non-trivial solution, we need:

$$\sin(\alpha) = 0 \implies \alpha = n\pi, \quad n = 1, 2, 3, \dots \quad (10)$$

Therefore:

$$Y_n(y) = B_n \sin(n\pi y) \quad (11)$$

Recall that $\alpha^2 = \frac{\lambda}{2}$, therefore $\lambda_n = 2(n\pi)^2$. Thus, we can write:

$$X'(x) + 2(n\pi)^2 X(x) = 0 \quad (12)$$

The general solution to which is:

$$X_n(x) = A_n e^{-2(n\pi)^2 x} \quad (13)$$

Therefore, the solution to the homogenous equation is:

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

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:

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

We can recognize this as a Fourier sine series on the interval $[0, 1]$, therefore:

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

Evaluating this integral gives:

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

For even values of n , $A_n = 0$. For odd values of n , $A_n = -\frac{4}{(n\pi)^3}$. Thus, we have:

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

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

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

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 :

$$\iiint_V \left(\int_x^{x+\Delta x} \frac{\partial u}{\partial x} dx \right) dV = \int_x^{x+\Delta x} \left(\iiint_V \left(2 \frac{\partial^2 u}{\partial y^2} + 2 \right) dV \right) dx \quad (20)$$

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

$$\iiint_V (u(x + \Delta x) - u(x)) dV = (u(x + \Delta x) - u(x)) \Delta y = \Delta y (u_i^{n+1} - u_i^n) \quad (21)$$

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

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

After evaluating the surface integral, the derivative is discretized using a central-difference scheme:

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

Thus, the right-hand side reduces to:

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

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

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

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

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

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

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

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

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

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

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

We can now program the numerical solver shown in Appendix A (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

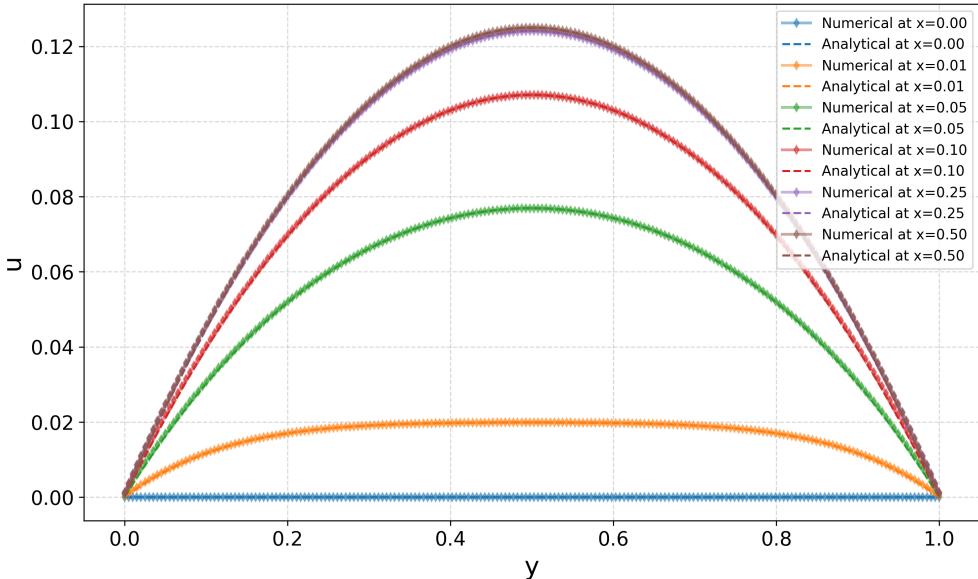


Figure 1: Comparison of numerical and analytical solution profiles at selected x -locations.

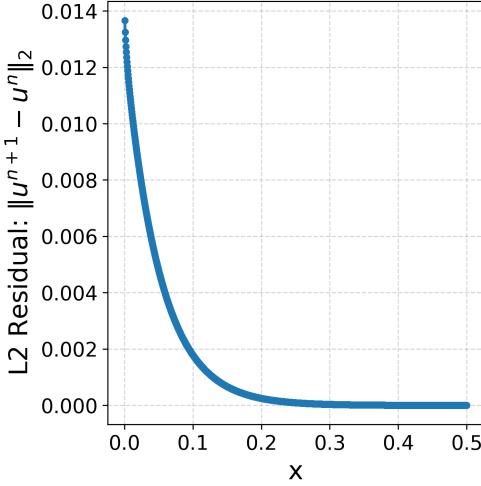


Figure 2: L2 residual vs. x

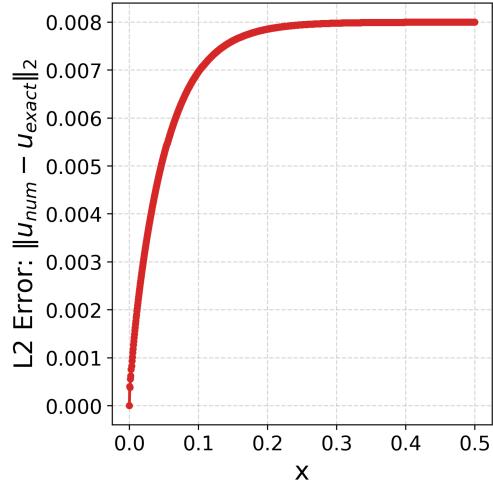


Figure 3: L2 error between numerical and analytical solutions vs. x

5. Discussion of Results

5.1. General description

The numerical solution of the parabolic equation found using the Crank-Nicholson and central difference schemes shows excellent agreement with the analytical solution derived in Section 2. Figure A 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 $e^{-2x\pi^2}$, the solution approaches the steady state profile

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

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 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. It must be noted that 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 numerical scheme is both accurate and stable for this problem.

A. Copy of Program Listing

```
1 # Christian DiPietrantonio
2 # ME M311: Computational Methods to Viscous Flows
3 # Computer Assignment 01
4 # 02/16/2026
5
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from scipy.linalg import lu_factor, lu_solve
9
10 # -----
11 # 1. Parameters and Mesh Sizing
12 # -----
13
14 def set_parameters():
15     params = {}
16
17     # Domain in y
18     params["y_min"] = 0
19     params["y_max"] = 1
20     params["N"]      = 200 # Number of cells (finite volume method)
21     params["dy"]     = (params["y_max"] - params["y_min"]) / params["N"]
22         # Cell size in y
23
24     # Stepping in x
25     params["x_max"] = 0.5 # Final x
26     params["Nx"]    = 1000 # Number of steps in x
27     params["dx"]    = params["x_max"] / params["Nx"] # Step size in x
28
29     # Source term
30     params["S"] = 2.0
31
32     # y-array for plotting and analytical solution evaluation
33     params["y"] = np.linspace(params["y_min"], params["y_max"],
34                               params["N"])
35
36     # Initial condition u(y,0) = 0
37     u0 = np.zeros(params["N"])
38
39     return params, u0
40
41 # -----
42 # 2. Build Matrices
43 # -----
44
45 def build_matrices(params):
```

```

44     N = params["N"]
45     dx = params["dx"]
46     dy = params["dy"]
47
48     r = dx / dy**2
49
50     A = np.zeros((N, N))
51     B = np.zeros((N, N))
52
53     # ----- Bottom Boundary: i = 0, u(x,0) = 0 -----
54     #LHS
55     A[0, 0] = 1 + 3*r
56     A[0, 1] = -r
57
58     #RHS
59     B[0, 0] = 1 - 3*r
60     B[0, 1] = r
61
62     # ----- Interior Cells: i = 1, ..., N-2 -----
63     for i in range(1, N-1):
64         #LHS
65         A[i, i-1] = -r
66         A[i, i] = 1 + 2*r
67         A[i, i+1] = -r
68
69         #RHS
70         B[i, i-1] = r
71         B[i, i] = 1 - 2*r
72         B[i, i+1] = r
73
74     # ----- Top Boundary: i = N-1, u(x,1) = 0 -----
75     #LHS
76     A[N-1, N-2] = -r
77     A[N-1, N-1] = 1 + 3*r
78
79     #RHS
80     B[N-1, N-2] = r
81     B[N-1, N-1] = 1 - 3*r
82
83     return A, B
84
85 # -----
86 # 3. Build RHS vector for a given  $u^n$ 
87 #
88
89 def build_rhs(u_n, params, B):
90     dx = params["dx"]

```

```

91     S = params["S"]
92     N = params["N"]
93
94     rhs = B @ u_n
95
96     # Add source term contribution
97     rhs += S * dx * np.ones(N)
98
99     return rhs
100
101 # -----
102 # 4. Step in x using LU decomposition
103 # -----
104
105 def step_in_x(params, u0):
106     A, B = build_matrices(params)
107
108     # LU factorization of A
109     lu, piv = lu_factor(A)
110
111     u = u0.copy()
112     N = params["N"]
113     Nx = params["Nx"]
114
115     # Initialize storage for x-location and solution profiles
116     x_vals = np.zeros(Nx + 1)
117     u_store = np.zeros((Nx + 1, N))
118
119     for n in range(Nx + 1):
120         x_now = n * params["dx"]
121         x_vals[n] = x_now
122         u_store[n, :] = u.copy()
123
124         # Break after storing solution at final x-location to avoid
125         # an extra solve
126         if n == Nx:
127             break
128
129         rhs = build_rhs(u, params, B)
130         u = lu_solve((lu, piv), rhs)
131
132     return x_vals, u_store
133
134 # -----
135 # 5. Post processing
136 # -----

```

```

137 def analytical_solution(x, y, n_terms=100):
138     """Compute the analytical solution (Eq. 19) of the parabolic PDE
139     """
140
141     u_exact = 0.5 * y * (1 - y)
142
143     for k in range(n_terms):
144         n = 2*k + 1
145         A_n = -4 / (n * np.pi)**3
146         decay = np.exp(-2 * x * (n * np.pi)**2)
147         u_exact += A_n * decay * np.sin(n * np.pi * y)
148
149     return u_exact
150
151 def post_process(params, x_vals, u_store):
152     y = params["y"]
153
154     x_targets = [0.0, 0.01, 0.05, 0.1, 0.25, 1.0]
155
156     errors = []
157     x_error_vals = []
158
159     # Plot 1: Numerical vs Analytical Solutions at selected x
160     # locations
161     plt.figure(figsize=(10, 6))
162
163     colors = ['tab:blue', 'tab:orange', 'tab:green', 'tab:red', 'tab'
164             :purple', 'tab:brown']
165
166     for i, x_target in enumerate(x_targets):
167         idx = np.argmin(np.abs(x_vals - x_target))
168         x_now = x_vals[idx]
169         u_num = u_store[idx, :]
170         u_exact = analytical_solution(x_now, y)
171
172         error = np.linalg.norm(u_num - u_exact, ord=2)
173         errors.append(error)
174         x_error_vals.append(x_now)
175
176         plt.plot(y, u_num, '-d', color=colors[i], alpha=0.5, label=f
177                 'Numerical at x={x_now:.2f}', linewidth=2, markersize=4)
178         plt.plot(y, u_exact, '--', color=colors[i], label=f
179                 'Analytical at x={x_now:.2f}', linewidth=1.5)
180
181     plt.xlabel('y', fontsize=18)
182     plt.ylabel('u', fontsize=18)
183     plt.xticks(fontsize=14)

```

```

179 plt.yticks(fontsize=14)
180 plt.grid(True, linestyle='--', alpha=0.5)
181 plt.legend()
182 plt.tight_layout()
183 plt.savefig('profiles_comparison.png', dpi=300, bbox_inches='tight')
184
185 # Plot 2/3: Residual, Error vs x
186 all_errors = []
187 residuals = []
188
189 for n in range(len(x_vals)):
190     # Error
191     u_num = u_store[n, :]
192     u_exact = analytical_solution(x_vals[n], y)
193     all_errors.append(np.linalg.norm(u_num - u_exact, ord=2))
194
195     # Residual
196     if n < len(x_vals) - 1: # skip last point since we don't
197         have u at x_{n+1}
198         u_next = u_store[n+1, :]
199         residuals.append(np.linalg.norm(u_next - u_num, ord=2))
200
201 # Plot 2: residual vs x
202 plt.figure(figsize=(5, 5))
203 plt.plot(x_vals[1:], residuals, '-o', color = 'tab:blue',
204           linewidth=2, markersize=4)
205 plt.xlabel('x', fontsize=18)
206 plt.ylabel(r'L2 Residual: $\|u^{n+1} - u^n\|_2$', fontsize=18)
207 plt.xticks(fontsize=14)
208 plt.yticks(fontsize=14)
209 plt.grid(True, linestyle='--', alpha=0.5)
210 plt.tight_layout()
211 plt.savefig('residual_vs_x.png', dpi=300, bbox_inches='tight')
212
213 # Plot 3: error vs x
214 plt.figure(figsize=(5, 5))
215 plt.plot(x_vals, all_errors, '-o', color='tab:red', linewidth=2,
216           markersize=4)
217 plt.xlabel('x', fontsize=18)
218 plt.ylabel(r'L2 Error: $\|u_{num} - u_{exact}\|_2$', fontsize
219 =18)
220 plt.xticks(fontsize=14)
221 plt.yticks(fontsize=14)
222 plt.grid(True, linestyle='--', alpha=0.5)
223 plt.tight_layout()
224 plt.savefig('error_vs_x_all.png', dpi=300, bbox_inches='tight')

```

```
221
222 # -----
223 # 5. Main Driver
224 #
225
226 def main():
227     params, u0 = set_parameters()
228     x_vals, u_store = step_in_x(params, u0)
229     post_process(params, x_vals, u_store)
230
231 if __name__ == "__main__":
232     main()
```

B. Other Calculations

MATRIX FORM OF GOVERNING EQUATIONS, $r = \frac{\Delta x}{\Delta y^2}$:
 BOTTOM BC:

$$(1 + 3r)u_0^{n+1} - ru_1^{n+1} = (1 - 3r)u_0^n + ru_1^n + 2\Delta x \quad (31)$$

INTERIOR CELLS:

$$-ru_{i-1}^{n+1} + (1 + 2r)u_i^{n+1} - ru_{i+1}^{n+1} = ru_{i-1}^n + (1 - 2r)u_i^n + ru_{i+1}^n + 2\Delta x \quad (32)$$

TOP BC:

$$-ru_{N-2}^{n+1} + (1 + 3r)u_{N-1}^{n+1} = ru_{N-2}^n + (1 - 3r)u_{N-1}^n + 2\Delta x \quad (33)$$