Project 1

Solutions to Parabolic Partial Differential Equations

ME 5311

John Gomes

Professor Xinyu Zhao

February 2026

Introduction

The goal of this project is to study the differences between the analytical and numerical solution of the following partial differential equation. This exercise will help in understanding discretization methods and their accuracy and stability.

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

Where the boundary conditions are:

$$u(x,0) = 0,\ u(x,1) = 0 \tag{1.1}$$

And the initial condition is:

$$u(0,y) = 0 \tag{1.2}$$

Numerical Method Description

To solve this PDE, it must first be decoupled. First, looking at the spatial derivative, $\frac{\partial^2 u}{\partial y^2}$, a central finite difference scheme for any given node, i, can be used.

$$\left.\frac{\partial^2 u}{\partial y^2}\right|_i \approx \frac{u_{i+1} + u_{i-1} - 2u_i}{\Delta y^2} \tag{2}$$

The Crank-Nicolson scheme was used to discretize the solution in the x-direction. This implicit method averages derivative of $\frac{\partial u}{\partial x}$ between the j^{th} and $j^{th} + 1$ steps for a given node, i. The following equation is a general solution to a 1-D parabolic PDE, like equation 1, in the discretized form. For our application, $a = 2, S = 2$.

$$\frac{u_i^{j+1} - u_i^j}{\Delta x} - a * \frac{1}{2\Delta y^2}\left[\left(u_{i+1}^{j+1} + u_i^{j+1} - 2u_{i-1}^{j+1}\right) + \left(u_{i+1}^{j} + u_i^{j} - 2u_{i-1}^{j}\right)\right] = S \tag{3}$$

Rewriting this equation by factoring $r = \frac{a\Delta x}{2\Delta y^2} = \frac{\Delta x}{\Delta y^2}$ gives the following.

$$u_i^{j+1} - u_i^j = \frac{\Delta x}{\Delta y^2}\left[\left(u_{i+1}^{j+1} + u_i^{j+1} - 2u_{i-1}^{j+1}\right) + \left(u_{i+1}^{j} + u_i^{j} - 2u_{i-1}^{j}\right)\right] + 2\Delta x \tag{4}$$

The discretized solution can then be solved using Lower-Upper (LU) decomposition. The Crank-Nicolson solution shown above is implicit, meaning the solution of the future step is dependent on the current solution step.

$$Au = b \tag{5}$$

See below for definitions of A and b. Here, A contains the unknown values in a tridiagonal matrix. This shows the effect of the central node, as well as the nodes to the left and right.

It is important to note that boundary nodes will be explicitly defined and are not included in this matrix.

$$A = \begin{bmatrix} 1+2r & -r & 0 & \cdots & 0 \\ -r & 1+2r & -r & \cdots & 0 \\ 0 & -r & 1+2r & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & -r & 1+2r \end{bmatrix} \tag{6}$$

$$b_i = r\left(u_{i-1}^j + u_{i+1}^j\right) + (1-2r)u_i^j + 2\Delta x \tag{7}$$

Derivation of Analytical Solution

To validate the numerical solution, we must compare it to the analytical solution of the governing equation. The y boundary conditions are homogenous, so the solution can be expanded as follows. This will satisfy the boundary conditions in equation 1.1.

$$u(x,y) = \sum_{n=1}^{inf} a_n(x)\sin(n\pi y) \tag{8}$$

Then, the respective derivatives of this must be computed.

$$\frac{\partial u}{\partial x} = \sum_{n=1}^{inf} a_n'(x)\sin(n\pi y) \tag{9}$$

$$\frac{\partial^2 u}{\partial y^2} = \sum_{n=1}^{inf} -(n\pi)^2 a_n(x)\sin(n\pi y) \tag{10}$$

The source term also must be written in a similar format. It can be expressed as a sine function from the interval (0,1).

$$2 = \sum_{n=1}^{inf} b_n \sin(n\pi y) \tag{11}$$

Integrating this over the interval (0,1) with respect to y gives the following. Note that for any even n, the equation for b_n equals zero.

$$b_n = 2\int_0^1 2\sin(n\pi y)dy \tag{12}$$

$$b_n = \frac{4}{n\pi}(1-(-1)^n) = \frac{8}{n\pi}, n = 1,3,5,\ldots \tag{13}$$

Rewriting the original PDE in expanded sine representation gives the following.

$$\sum_{n=1}^{inf} a_n(x)\sin(n\pi y) + 2\sum_{n=1}^{inf}(n\pi)^2 a_n(x)\sin(n\pi y) = \sum_{n=1}^{inf} b_n \sin(n\pi y) \tag{14}$$

Each component is a function of $\sum_{n=1}^{inf}\sin(n\pi y)$, therefore this can be written as the following ODE.

$$a_n'(x) + 2(n\pi)^2 a_n(x) = b_n \tag{15}$$

Using an integrating factor:

$$a_n'(x)e^{2(n\pi)^2 x} + e^{2(n\pi)^2 x}2(n\pi)^2 a_n(x) = b_n e^{2(n\pi)^2 x} \tag{16}$$

Notice the left-hand side of the equation is simply the result of the product rule, therefore:

$$a_n e^{2(n\pi)^2 x} = \int b_n e^{2(n\pi)^2 x}\,dx = \frac{b_n}{2(n\pi)^2}e^{2(n\pi)^2 x} + C \tag{17}$$

The integral constant, C, can be solved using the initial condition $u(0, y) = 0$. Divide both sides of the equation by the integration factor, and the overall solution for $a_n(x)$ is as follows.

$$a_n(x) = \frac{b_n}{2(n\pi)^2}(-e^{-2(n\pi)^2 x} + 1) \tag{18}$$

Substituting in the solution for b_n in equation 13, we get the final analytical solution of the PDE.

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

Note that as n approaches infinity, the exponential term will disappear. Therefore, the steady state solution is simplified to the following.

$$u(x, y) = \sum_{n=1,3,5\ldots}^{inf} \frac{4}{(n\pi)^3}\sin(n\pi y) = \frac{1}{2}(y - y^2) \tag{20}$$

Results & Discussion

Accuracy

With the Python code shown in the Appendix, we can compare the profile of this parabolic PDE at different x locations for both analytical and numerical solutions. See the figure below for a comparison of the Crank-Nicolson numerical solution and analytical solution for various x-values. This plot includes $x = 0.02, 0.06, 0.30, 0.80$.

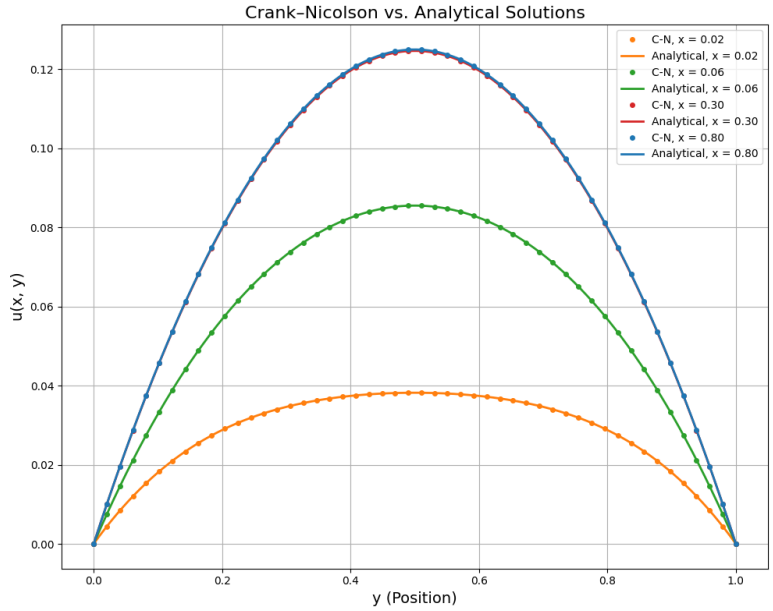

Figure 1: Crank-Nicolson Numerical Solution Validation

When looking at this problem, it is easiest to think of x as a time-like variable. Here, we can see that as x gets larger, it eventually converges onto the same path. This is best shown by zooming in.

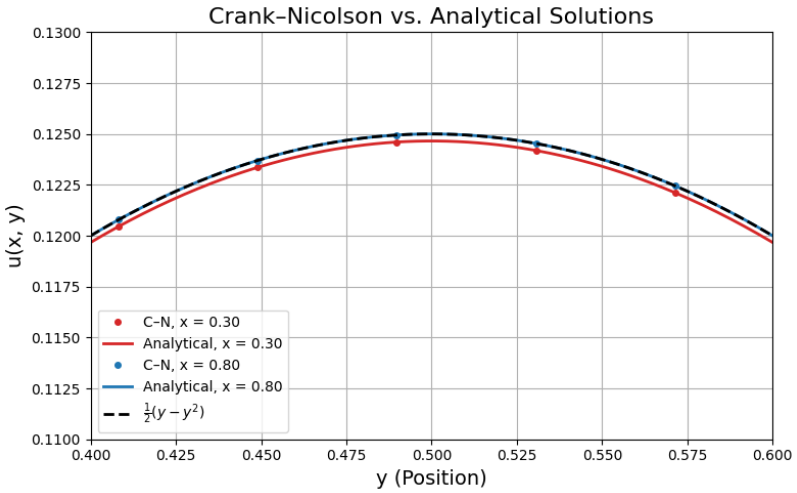

Figure 2: Crank-Nicolson Numerical Solution Validation (zoomed)

The steady state solution described in equation 20 is shown by the black dotted line. By $x = 0.3$, the system has nearly reached the steady state solution, though a difference is still noticeable. At $x = 0.8$, the solution is indistinguishable from the steady state solution. Through an iterative process, the numerical solution becomes indistinguishable at $x \approx 0.4$. The agreement between the steady state analytical and numerical solutions gives validity to the C-N scheme.

These results also indicate that the Crank-Nicolson scheme can trace analytical solutions with high fidelity, even in transient solutions (for example, $x = 0.06$). This is a second-order method, $O(\Delta x^2, \Delta y^2)$, which ensures errors are small with proper step sizes.

Stability

The Crank-Nicolson scheme is an implicit scheme; therefore, it is unconditionally stable. This implies that for any value of $r = \Delta x/\Delta y^2$, the Crank–Nicolson solution will remain bounded and will not diverge. For an explicit scheme, $r = \Delta x/\Delta y^2 < 0.5$ otherwise it will become unstable and diverge. Though, unconditional stability does not necessarily mean that the solution will be accurate. The plots above have a calculated $r = 0.48$. The plot below shows the same set of solutions with a much larger step size, $r = 48.0$.

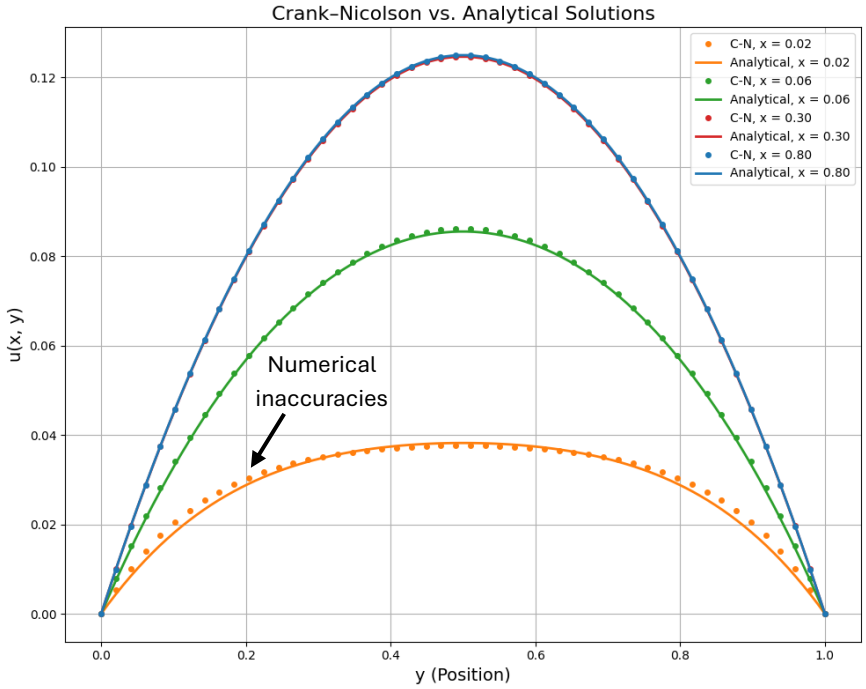

Figure 3: Crank-Nicolson Numerical Inaccuracy due to Larger Step Sizes

This shows that accuracy and stability are fundamentally independent. When using a much larger timestep, the C-N solutions begin to deviate from the analytical solutions. This is particularly noticeable for smaller values of x, where there is rapid change every iteration. This makes sense as C-N is a second-order accurate method.

Notably, as the solution approaches steady state, this error lessens. This is because the system is well defined with the source term and boundary conditions. The system is driven to the steady state solution from equation 20.

Appendix

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

```
#parameters

L=1 #domain y-length, defined above in BCs
X_total=1  #independent var length
Ny=50 #y partitions
Nx=5000 #x partitions, adjust to 50 for r=48
source_coeff=2 #2 on RHS of PDE
alpha=2 #d^2u/dy^2 coeff
dy=L/(Ny-1)
dx=X_total/Nx
y=np.linspace(0,L,Ny)

r=(alpha*dx)/(2*dy**2) #general r form for modularity
print(r)
y_an=np.linspace(0,L,100) #define for analytical sol
N_int=Ny-2
```

```
#Solving A, Au=b
A = np.zeros((N_int, N_int))
for i in range(N_int):  #create tri-diag mentioned in report
    A[i, i] = 1+2*r
    if i > 0:
        A[i, i-1] = -r
    if i < N_int - 1:
        A[i, i+1] = -r
lu, piv = lu_factor(A) #lu decomp tool from scipy.linalg- Google Gemini
aided with this
```

```
u = np.zeros(Ny)  #IC of u(0, y) = 0
x_vals = [100,300,1800,4000]  # x-steps to store numerical sols
u_store_numer={} #init
u_analytical={} #init

for n in range(1, Nx + 1):
    #Solving b, Au=b
    b = (r*u[:-2]+(1-2*r)*u[1:-1]+r*u[2:]+source_coeff*dx) #b equation
mentioned in report

    u[1:-1] = lu_solve((lu, piv), b) #lu_solve tool from scipy.linalg-
Google Gemini aided with this

    if n in x_vals:
        x_now=n*dx #analytical sol at a given x val
```

```
        u_store_numer[n]=u.copy() #store solution
        u_an=np.zeros_like(y_an)
        for k in range(1, 201, 2):  #100 terms
            sol_summation = (4/(k**3*np.pi**3))*(1-np.exp(-
2*k**2*np.pi**2*x_now))*np.sin(k*np.pi*y_an) #long form eqn from report
            u_an+=sol_summation
        u_analytical[n]=u_an
```

```
#plot
plt.figure(figsize=(10, 8))
colors = [ 'tab:orange', 'tab:green', 'tab:red','tab:blue']
markevery=2
for idx, n in enumerate(x_vals):
    x_now=n*dx
    label_text=f'x={x_now:.2f}'

    #numerical
    plt.plot(y, u_store_numer[n], 'o', color=colors[idx], markersize=4,
label=f'C-N, {label_text}')
    #analytical
    plt.plot(y_an, u_analytical[n], '-', color=colors[idx], linewidth=2,
label=f'Analytical, {label_text}')

#plt.xlim(0.4,0.6) #toggle for zoomed in plots
#plt.ylim(0.11,0.13) #toggle for zoomed in plots
plt.xlabel('y (Position)', fontsize=14)
plt.ylabel('u(x, y)', fontsize=14)
plt.title('Crank-Nicolson vs. Analytical Solutions', fontsize=16)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```

```
#plot 0.5*(y-y^2), throw in plot loop to add to plot
y_ss= np.linspace(0, 1, 100)
u_ss = 0.5*(y_ss-y_ss**2)
plt.plot(y_ss,u_ss,'k--',linewidth=2,label=r'$ \frac{1}{2}(y - y^2) $')
```