# Homework 2 - Couette Flow

Aurelia Meiqi Brook

Professor Sreenivasan

Introduction to Fluid Dynamics

April 21, 2022

## 1 Abstract

Given parallel planar Couette flow, numerical solutions to the partial differential equation formed by the streamwise velocity are investigated. Comparison of error margins due to the numerical method used to invert matrices is done, resulting in the Thomas Algorithm having a much lower time complexity and producing a more stable solution to the problem. Following this is an in-depth view of how errors change with different spatial and temporal resolutions, as well as the convergence of planar Couette flow to it's steady-state solution. As expected, smaller time steps and higher spatial resolution both result in data that better approximates the exact solution. However, results tend to not deviate so far from one another unless the resolution parameters or the allowed tolerance in the round-off error are multiple orders of magnitude off from one another.

## 2 Introduction

Planar Couette flow is the flow of fluids with some viscosity $\nu$ beween two surfaces that are moving tangentially relative to one another. This relative motion induces flow due to the imposed shear stresses on the viscous fluid, otherwise known as shear-driven fluid motion. Assuming the surfaces are parallel, and the flow is fully developed, the Navier-Stokes equation simplifies to:

$$\frac{\partial u}{\partial t} = \nu \frac{\partial^2 u}{\partial^2 y} \tag{1}$$

where the streamwise velocity has both spacial and temporal coordinates. In order to solve for $u(y,t)$, we have been given the following boundary conditions and initial condition:

$$\begin{aligned} u(0,t) &= 0 \\ u(h,t) &= U \\ u(y,0) &= U\frac{y}{h} + U\sin\frac{\pi y}{h} \end{aligned} \tag{2}$$

## 3 Exact Solution

There are multiple ways to evaluate how close the numerical answer is to the exact solution, and in order to properly examine the error rate as the numerical solution converges on the actual solution, one must have an idea of what the actual solution is. Thankfully, with the given initial conditions and boundary points, the exact solution is solvable:

$$u(y,t) = y + e^{-\pi^2 t}\sin \pi y \tag{3}$$

## 4 Numerical Solution

After non-dimensionalizing the variables, the results above can also be discretized with finite difference approximation. We use a first order backward difference for the temporal derivative:

$$\frac{\partial u}{\partial t} \approx \frac{u_j^{n+1} - u_j^n}{\Delta t} \tag{4}$$

The second order central difference is then used in place of the spatial derivative:

$$\frac{\partial^2 u}{\partial^2 y} \approx \frac{u_{j+1}^{n+1} - 2u_j^{n+1} + u_{j-1}^{n+1}}{(\Delta y)^2} \tag{5}$$

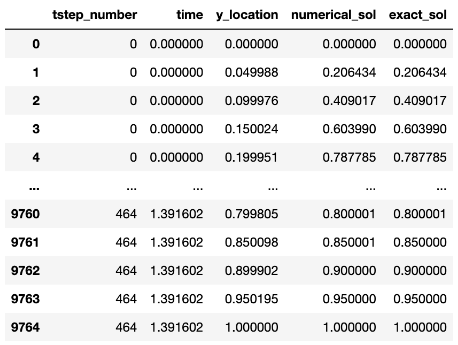
With the substitution $r = \frac{\Delta t}{(\Delta y)^2}$, these become a system of equations:

$$u_j^n = -(ru_{j-1}^{n+1} - (2r+1)u_j^{n+1} + ru_{j+1}^{n+1}) \tag{6}$$

Given a system of equations which can be solved by writing them in the form $Ax = b$, there are two ways to go about solving the PDE numerically: using the Thomas Algorithm, or just inverting the matrices $A$ that contain the coefficients applied to the vector $x$ which contain $u_j^{n+1}$ in order to solve for a vector $b$ which contains $u_j^n$.

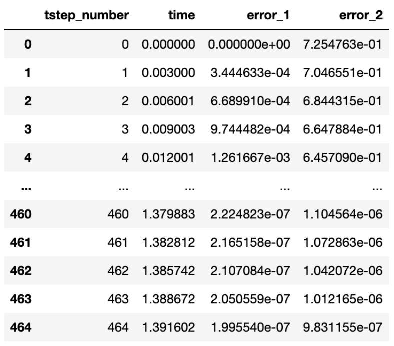
## 4.1 Methodology

It is worth noting that for the submitted file of sample solutions at $N = 21$ and $\Delta t = 0.003$, the Thomas algorithm will be used. This is a simplified Gaussian-elimination method, which can be used here since $A$ is tridiagonal. It is much more computationally efficient than manual matrix inversion, as it can be solved in $\mathcal{O}(n)$ operations, as opposed to $\mathcal{O}(n^3)$.



| | tstep_number | time | y_location | numerical_sol | exact_sol |
|---|---|---|---|---|---|
| **0** | 0 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| **1** | 0 | 0.000000 | 0.049988 | 0.206434 | 0.206434 |
| **2** | 0 | 0.000000 | 0.099976 | 0.409017 | 0.409017 |
| **3** | 0 | 0.000000 | 0.150024 | 0.603990 | 0.603990 |
| **4** | 0 | 0.000000 | 0.199951 | 0.787785 | 0.787785 |
| **...** | ... | ... | ... | ... | ... |
| **9760** | 464 | 1.391602 | 0.799805 | 0.800001 | 0.800001 |
| **9761** | 464 | 1.391602 | 0.850098 | 0.850001 | 0.850000 |
| **9762** | 464 | 1.391602 | 0.899902 | 0.900000 | 0.900000 |
| **9763** | 464 | 1.391602 | 0.950195 | 0.950000 | 0.950000 |
| **9764** | 464 | 1.391602 | 1.000000 | 1.000000 | 1.000000 |

Figure 1: Velocity data generated for the $N = 21$ and $\Delta t = 0.003$ case.



| | tstep_number | time | error_1 | error_2 |
|---|---|---|---|---|
| **0** | 0 | 0.000000 | 0.000000e+00 | 7.254763e-01 |
| **1** | 1 | 0.003000 | 3.444633e-04 | 7.046551e-01 |
| **2** | 2 | 0.006001 | 6.689910e-04 | 6.844315e-01 |
| **3** | 3 | 0.009003 | 9.744482e-04 | 6.647884e-01 |
| **4** | 4 | 0.012001 | 1.261667e-03 | 6.457090e-01 |
| **...** | ... | ... | ... | ... |
| **460** | 460 | 1.379883 | 2.224823e-07 | 1.104564e-06 |
| **461** | 461 | 1.382812 | 2.165158e-07 | 1.072863e-06 |
| **462** | 462 | 1.385742 | 2.107084e-07 | 1.042072e-06 |
| **463** | 463 | 1.388672 | 2.050559e-07 | 1.012165e-06 |
| **464** | 464 | 1.391602 | 1.995540e-07 | 9.831155e-07 |

Figure 2: Error data generated for the $N = 21$ and $\Delta t = 0.003$ case.

### 4.2 Alternative Methods

Alternatively, one can also use numpy.linalg.iniv to compute the multiplicative inverse of a matrix. This solution is less stable, as will be examined later on in the Additional Analysis section of the report.

## 5 Error Analysis

### 5.1 Comparison to Exact Solution

One way of calculating error is to compare the numerical solution to the exact solution at every increment in the independent variables. This can be done using the root-mean-squared method:

$$E_1(t_n) = (\frac{1}{N_j}\sum_{j}^{N}|u_j^n - u(y_j, t_n)|^2)^{\frac{1}{2}} \tag{7}$$

This method does not include boundary points, so $N_j = N - 2$. Results are as follows:

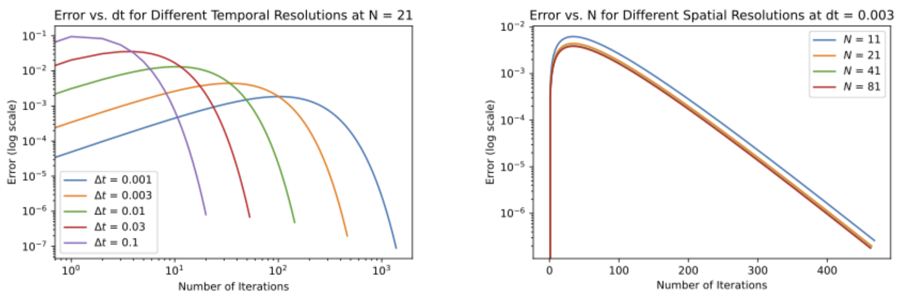


Figure 3: Examination of error trends as spatial and temporal resolution is varied.

### 5.2 Comparison to Steady-State Solution

The other approach would be to compare the numerical solution to the steady-state solution, effectively measuring the data's convergence to a steady-state solution:

$$E_1(t_n) = (\frac{1}{N_j}\sum_{j}^{N}|u_j^n - u(y_j, \infty)|^2)^{\frac{1}{2}} \tag{8}$$

Convergence at different resolutions are shown below:

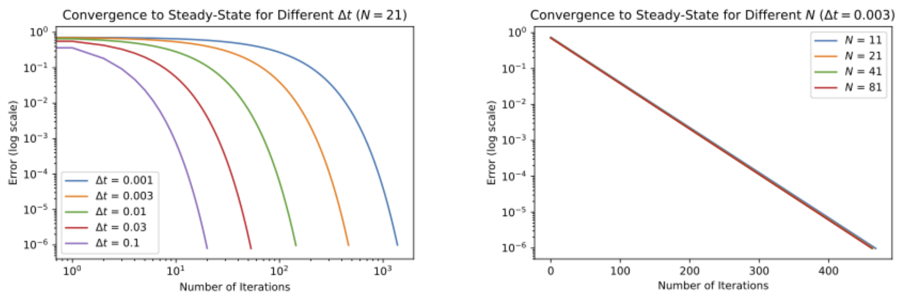


Figure 4: Spatial and temporal resolution is varied to demonstrate changes in convergence rate.

# 6 Additional Analysis

## 6.1 Alternative Numerical Methods

In order to properly verify that the Thomas algorithm was a better choice than simply using np.linalg.inv, the empirical time complexity was calculated using python's time package, which tracks the computation time for completing tasks.

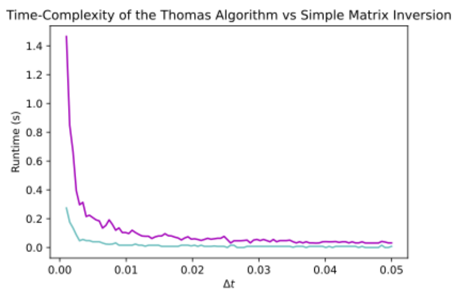


Figure 5: Comparison of the possible matrix inversion methods, where the Thomas algorithm is in cyan, and np.linalg.inv is in magenta.

As can be seen, not only does the Thomas algorithm have a consistently lower time complexity for varied $\Delta t$, the solution method is generally more stable and consistent.

## 6.2 Variations in $\epsilon$

In order to place limits on the allowed tolerance in the round-off error, $E_2$ is set to be lower than $\epsilon = 10^{-6}$, so the convergence rate will be obtained until $E_2 < \epsilon$. To this end, the convergence of the numerically calculated velocities to the steady-state solution was redone under new values for $\epsilon$.

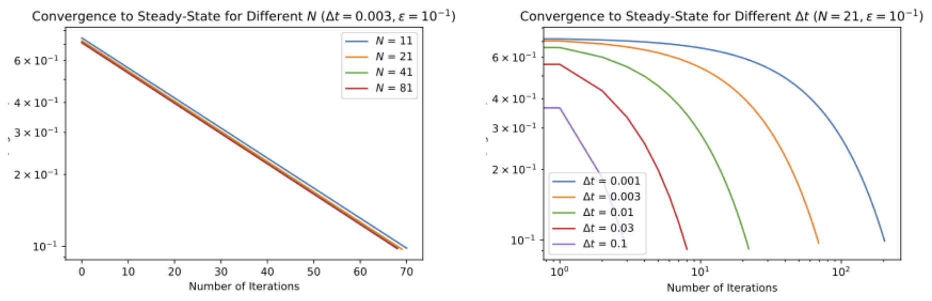


Figure 6: Convergence at different resolutions and redone at new limitations for $E_2$.

It can be expected that increasing $\epsilon$ affects the convergence rate of the numerical solution, however it is worth noting that $\epsilon$ had to be changed by many orders of magnitude until a noticeable difference was generated. The y-axes are in log scale (as they also were in the original computation of these error rates), and it can be seen that the error rate is much higher but the rate of convergence is quite similar. The relationship between error and iteration number maintains its shape for the most part, particularly as iterations increase. However, the resolution is worse than the $\epsilon = 10^{-6}$'s version.

## 7 Conclusion

Solutions to the planar Couette flow partial differential equation are obtainable numerically when given boundary and initial conditions. Convergence of the numerical solution to the exact solution and the steady-state solution increases with both spatial and temporal resolution, as expected. Variations in round-off error can greatly affect the convergence rate of the numerical solution to the steady-state solution. Alternative methods can also work to solving the PDE for planar Couette flow, however they are far less efficient. Overall, not much of the data gained strayed far from expectation.