Numerical Solutions to the Navier-Stokes Equations

Augustas Macijauskas

Student ID 10455606

Supervised by Prof Matthias Heil

Course code MATH30022

May 2022

Contents

1 Introduction

In this project we are going to investigate numerical approaches to solving the Navier-Stokes equations

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u}$$

$$\nabla\cdot\mathbf{u} = 0$$

which are a set of nonlinear PDEs that govern incompressible viscous fluid flow. Above, the three-dimensional velocity field

$$\mathbf{u}(x,y,z,t) = \begin{bmatrix} u(x,y,z,t) \\ v(x,y,z,t) \\ w(x,y,z,t) \end{bmatrix}$$

is the quantity that we are after. The equations also contain the pressure in the fluid $p = p(x,y,z,t)$ and two constitutive parameters, the density of the fluid ρ and the viscosity μ which are positive constants.

Finding solutions to these equations is important as they come up in many scientific and engineering scenarios which include modelling fluid flow in pipes, air flow past a plane's wing, ocean currents, weather, and many others. Unfortunately, the fact that the equations are highly nonlinear means that analytic solutions are not available in most cases of interest. This is why we turn our interest to investigating numerical approaches to have a chance at solving the equations.

Our aim for this project is therefore to find ways to approximate solutions to these equations numerically. To have a concrete problem to run and compare our algorithms on, we are going to consider the well-known driven cavity problem [3, Chapter 6]. The problem requires to find an approximation to the incompressible viscous two-dimensional flow

$$\mathbf{u}(x,y,t) = \begin{bmatrix} u(x,y,t) \\ v(x,y,t) \end{bmatrix}$$

on the $L \times L$ square (see figure 1) with boundary conditions of no slip and no penetration on the bottom and the two sides and a given horizontal velocity on the top wall

$$\mathbf{u} = 0 \text{ on } y = 0 \text{ and } 0 \le x \le L$$

$$\text{and } \mathbf{u} = 0 \text{ on } x = 0 \text{ or } x = L \text{ and } 0 \le y \le L$$

$$\text{and } u = U(x) \text{ and } v = 0 \text{ on } y = L \text{ and } 0 \le x \le L$$

starting from an initial state of rest.

After we find the flow field $\mathbf{u}$, we are also going to be interested in finding the viscous force on the top wall

$$F = \int_0^L \mu \left.\frac{\partial u}{\partial y}\right|_{y=L} dx$$

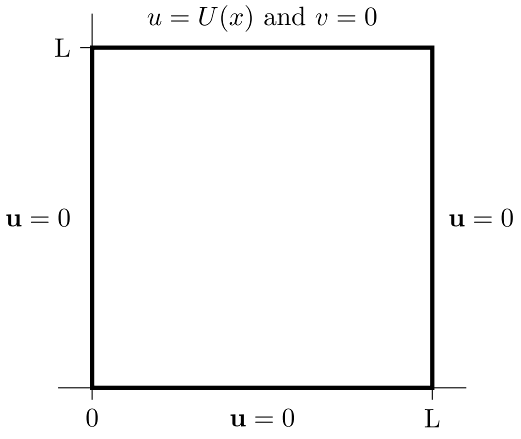


Figure 1: The boundary conditions on the $L \times L$ square.

The project was initially inspired by chapters 1 and 2 of E. J. Hinch's book "Think Before You Compute" [5], which the theory in chapters 1-3 is mainly based on. From there, the original contributions made are: (1) filling in the details for the sizes of discretization errors which were not presented in the book; (2) exploring alternative approaches, such as sparse linear system solver and Newton-Raphson iteration, to solve the Navier-Stokes equations, with particular emphasis on improving computation time and memory usage; (3) implementing the methods in the Python programming language from scratch.

The structure of this report is therefore as follows. Chapter 2 will discuss the physics of the problem and how to transform the equations to a simpler form by getting rid of the pressure term. Chapter 3 will discuss how to discretize the equations to be able to solve them using a computer and will present derivation for the sizes of the errors of discretization. Finally, chapters 4-5.3 will present the initial (i.e. presented in [5, Chapter 2]) and the alternative algorithms that can be used to solve the problem and will analyze their accuracy, speed and memory cost. The code implementations for the numerical methods in Python programming language can be found in the appendices or in the GitHub repository (external link, last accessed: 2022 05 17).

This report assumes knowledge of PDEs and vector calculus at the level of MATH20401 Partial Differential Equations and Vector Calculus A or an equivalent course. Knowledge of numerical analysis (e.g. MATH20602 Numerical Analysis 1) or fluid mechanics (e.g. MATH35001 Viscous Fluid Flow) can be beneficial, but is not essential because all the material that is required will be presented here.

2 The governing PDEs and the physics of the problem

2.1 The physics of the problem

Physically, the requirement that the velocity is divergence free is sometimes also called the *equation of continuity* and represents the conservation of mass constraint. More specifically, it is derived from the requirement that mass flux into a spatially fixed control volume is equal to the rate of change of mass in that volume. Likewise, the other three equations are the continuum equivalent of Newton's second law, i.e. the conservation of momentum requirement [6, Chapters 1.2.5, 2.1 and 2.2].

2.2 Boundary conditions on the top of the lid

We have not specified the boundary condition on the top of the lid above. For our toy problem, we choose to impose

$$U(x) = U_0 \sin^2(\pi x/L)$$

We note that $U(0) = U(L) = 0$, so that the requirements on the velocity components from both the top and the sides are the same and equal to zero at the top corners. Therefore, this boundary condition is easy to deal with because it will not cause us any trouble in terms of producing inconsistencies in the flow.

2.3 Non-dimensionalization

It turns that we can reduce the number of parameters to just a single parameter called the Reynolds number Re. To achieve this, we scale the velocity $\mathbf{u}$ with U_0, lengths x and y with L, time with L/U_0 and pressure inertially with ρU_0^2, so that $Re = U_0 L\rho/\mu$. Having just a single parameter gives us the desirable property that problems with different values of the physical parameters, but identical Reynolds numbers, will have the same solution.

The non-dimensionalized form of the problem can then be derived starting from the Navier-Stokes equations as follows (a more detailed derivation can be found in [6, Chapter 4.1.3])

$$\rho\left(\frac{U_0^2}{L}\frac{\partial\tilde{\mathbf{u}}}{\partial\tilde{t}} + \left(\frac{U_0}{L}\tilde{\mathbf{u}}\cdot\tilde{\nabla}\right)U_0\tilde{\mathbf{u}}\right) = -\frac{\rho U_0^2}{L}\tilde{\nabla}\tilde{p} + \mu\frac{U_0}{L^2}\tilde{\nabla}^2\tilde{\mathbf{u}}$$

$$\frac{U_0}{L}\tilde{\nabla}\cdot\tilde{\mathbf{u}} = 0$$

where the tilde notation is used to denote non-dimensional quantities. Dividing through by $\rho U_0^2/L$ and U_0/L we get

$$\frac{\partial\mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u} = -\nabla p + \frac{1}{Re}\nabla^2\mathbf{u}$$

$$\nabla \cdot \mathbf{u} = 0$$

where we have dropped the tilde notation for brewity. For the rest of this project, we will be using this form of the equations and the fact that they are non-dimensional will be implicitly assumed.

Similarly, the non-dimensional boundary conditions can be found to be

$$\mathbf{u} = 0 \text{ on } y = 0 \text{ and } 0 \leq x \leq 1$$

$$\text{and } \mathbf{u} = 0 \text{ on } x = 0 \text{ or } x = 1 \text{ and } 0 \leq y \leq 1$$

$$\text{and } u = \sin^2(\pi x) \text{ and } v = 0 \text{ on } y = 1 \text{ and } 0 \leq x \leq 1$$

and the initial condition becomes

$$\mathbf{u}(x, y, t = 0) = 0 \text{ for } 0 \leq x \leq 1 \text{ and } 0 \leq y \leq 1$$

Note that here we have used the same notation as in the dimensional version of the equations for brewity, and for the rest of this report this notation will refer to the non-dimensionalized version of the equations and the corresponding physical quantities.

Physically, the Reynolds number $Re = U_0 L\rho/\mu$ measures the relative importance of inertial to viscous terms in the Navier-Stokes equations [5, Chapter 1.2]. To avoid the many of the complications of solving the problem numerically for very small or very large Reynolds numbers, we are going to use a modest value of $Re = 10$ throughout this project. Another reason for this is that the analytical approximations for low/high Reynolds number will not work for this value, so looking for a numerical solution is the only possible way to solve the problem.

The non-dimensionalized force on the top of the lid that we are going to be interested in will then become

$$F = \int_0^1 \left. \frac{\partial u}{\partial y} \right|_{y=1} dx$$

2.4 The streamfunction-vorticity formulation

We can exploit the fact that our problem of interest is two-dimensional by using the so-called streamfunction-vorticity formulation of the Navier-Stokes equations. This formulation also gets rid of the pressure term from the equations which helps avoid the complications that arise from having to correctly account for pressure using numerical methods.

We start rewritng the equations by first noticing that the conservation of mass constraint on the two-dimensional velocity $\nabla \cdot \mathbf{u} = 0$ can be satisfied by representing the velocity using a streamfunction $\psi(x, y, t)$[5, Chapter 2.1]:

$$u = \frac{\partial \psi}{\partial y} \text{ and } v = -\frac{\partial \psi}{\partial x}$$

Indeed, then we have that

$$\nabla \cdot \mathbf{u} = \frac{\partial}{\partial x}\frac{\partial \psi}{\partial y} - \frac{\partial}{\partial y}\frac{\partial \psi}{\partial x} = 0$$

since we can assume that the streamfunction is smooth enough so that we can exchange the order of the partial derivatives.

In two dimensions, we know that the vorticity is

$$\omega = \nabla \times \mathbf{u} = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y} = -\frac{\partial}{\partial x}\frac{\partial \psi}{\partial x} - \frac{\partial}{\partial y}\frac{\partial \psi}{\partial y} = -\nabla^2 \psi$$

where we note that we take the curl in two dimensions.

We can then make things easier for ourselves and remove pressure from the equations by taking the curl of the non-dimensionalized momentum equations and using vector calculus identities to obtain [6, Chapter 4.1.2]

$$\nabla \times \left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u}\right) = \nabla \times \left(-\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u}\right)$$

$$\implies \frac{\partial}{\partial t}(\nabla \times \mathbf{u}) + (\mathbf{u} \cdot \nabla)(\nabla \times \mathbf{u}) - ((\nabla \times \mathbf{u}) \cdot \nabla)\mathbf{u} = 0 + \frac{1}{Re}\nabla^2(\nabla \times \mathbf{u})$$

$$\implies \frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla)\omega - (\omega \cdot \nabla)\mathbf{u} = \frac{1}{Re}\nabla^2 \omega$$

where we can exchange the order of partial derivatives due to smoothness and we note that the pressure term disappears because the curl of grad is zero. Further, the vortex-streching term $(\omega \cdot \nabla)\mathbf{u}$ is zero for our two-dimensional flow which leads us to the vorticity equation

$$\frac{\partial \omega}{\partial t} + (\mathbf{u} \cdot \nabla)\omega = \frac{1}{Re}\nabla^2 \omega$$

To make more sense of the advection of vorticity term $(\mathbf{u} \cdot \nabla)\omega$, we note that it can be rewritten as

$$(\mathbf{u} \cdot \nabla)\omega = \psi_y \omega_x - \psi_x \omega_y$$

where the subscripts in this case denote partial differentiation.

For the boundary conditions, the no normal component of velocity on all sides makes the sides streamlines of the flow. It turns out that $\psi = \text{const}$ on a streamline, and by convention we set that constant to zero [4, Chapter 8] and so impose

$$\psi = 0$$

To account for the tangential velocity, we impose

$$u = \frac{\partial \psi}{\partial y} = \sin^2(\pi x) \text{ on } y = 1 \text{ and } 0 \leq x \leq 1$$

$$u = \frac{\partial \psi}{\partial y} = 0 \text{ on } y = 0 \text{ and } 0 \leq x \leq 1$$

$$-v = \frac{\partial \psi}{\partial x} = 0 \text{ on } x = 0 \text{ or } x = 1 \text{ and } 0 \leq y \leq 1$$

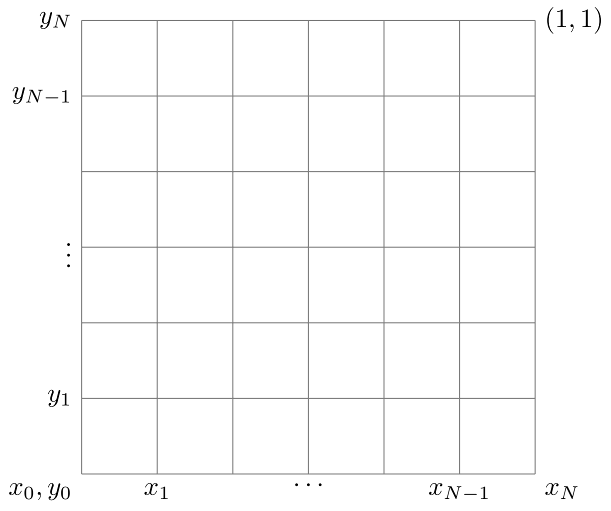


Figure 2: The uniform spatial grid.

3 Discretization of the equations

3.1 Discretizing using finite differences

To be able to solve the problem numerically on a computer, we have to represent the continuous functions $\psi(x, y, t)$ and $\omega(x, y, t)$ in a discrete manner. To do this, we are going to use the simplest possible approach and store the values of the unknown functions at a discrete number of points in time and a discrete number of points on a two-dimensional grid. The simplest such representation can be obtained by using equally spaced points in time separated by a time-step of size Δt and over equally spaced points on a two-dimensional grid separated by a space-step h in each direction. More precisely, if we use $N+1$ equally spaced points in the x and y directions, then the space-step size will be $h = 1/N$ and we will have a uniform grid of points $(x_i = ih, y_j = jh)$ for $i = 0, \ldots, N$ and $j = 0, \ldots, N$, see figure 2. Note that i and j will have these ranges for the rest of this report, unless specified otherwise. We can then approximate our functions by

$$\omega_{i,j}^n \approx \omega(x = ih, y = jh, t = n\Delta t)$$

and similarly for the streamfunction and other physical quantities.

Since we are solving partial differential equations, we are going to have to approximate derivatives of functions. A simple and accurate approach to approximate the derivative f' at $x = ih$ in one dimension is the so-called central differencing [1, Chapter 7.1.1]

$$f_i' = \frac{f_{i+1} - f_{i-1}}{2h} + O(h^2)$$

which has second order error.

The second derivative can similarly be approximated by central differences by applying the above equation at fictitious mid-points [1, Chapter 7.1.2]

$$f_i'' = \frac{\left(f'_{i+\frac{1}{2}} \approx \frac{f_{i+1}-f_i}{h}\right) - \left(f'_{i-\frac{1}{2}} \approx \frac{f_i - f_{i-1}}{h}\right)}{h} =$$

$$= \frac{f_{i+1} - 2f_i + f_{i-1}}{h^2} + O(h^2)$$

where we note that evaluating at mid-points means that we have h instead of $2h$ in the denominators in the top equation. Again, this approximation has second order error.

To find the Laplacian of ψ, we simply add the numerical approximations of the second derivatives in x and in y, obtaining

$$(\nabla^2\psi)_{i,j} = \frac{\psi_{i+1,j} - 2\psi_{i,j} + \psi_{i-1,j}}{h^2} + \frac{\psi_{i,j+1} - 2\psi_{i,j} + \psi_{i,j-1}}{h^2}$$

$$= \frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}}{h^2}$$

which will retain the second order error and we note that we can use an identical discretization for $\nabla^2\omega$.

3.2 Decoupling the problem

We want to find solutions for the streamfunction and the vorticity to simultanously satisfy

$$\nabla^2\psi = -\omega \text{ and}$$

$$\frac{\partial\omega}{\partial t} + (\mathbf{u}\cdot\nabla)\omega = \frac{1}{Re}\nabla^2\omega$$

We could discretize these equations on a grid using the approximations for derivatives from above and we would call the resulting set of equations the monolithic equations. However, instead of trying to solve all of these equations at once, but it is easier to make progress if we decouple the equations.

First, we will solve the so-called Poisson problem

$$\nabla^2\psi = -\omega \text{ with boundary condition } \psi = 0$$

from which we will get the streamfunction ψ at some point in time t given the vorticity ω at that time t.

Then, using the found approximation the streamfunction at time t, we will be able to evaluate the time derivative of the vorticity

$$\implies \frac{\partial\omega}{\partial t} = -(\psi_y\omega_x - \psi_x\omega_y) + \frac{1}{Re}\nabla^2\omega$$

and so find the vorticity at the next time-step. We will then iterate this process to step in time.

One complication that we are going ot have is that the boundary conditions that come with the vorticity evolution equation are in terms of normal derivatives of the streamfunction and not the vorticity, so we will have to choose the vorticity on the boundary so that the normal derivatives of the streamfunction take the correct imposed values (see chapter 3.4).

3.3 Discretization of the Poisson problem

Using the above approximations, we can finally formulate the Poisson problem on the unit square. If we use the spatial grid as defined above, on the interior points $i = 1, \dots, N-1$, $j = 1, \dots, N-1$ we have to solve [5, Chapter 2.3]

$$\frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}}{h^2} = -\omega_{i,j}$$

along with

$$\psi = 0 \text{ for } i = 0 \text{ and } N, j = 0, \dots, N \text{ and for } j = 0 \text{ and } N, i = 0, \dots, N$$

on the boundary. This is a system of linear equations, but a very sparse one, so we will later see how to exploit this when looking for a solution to the system.

3.4 Discretization of the vorticity evolution equation

Using the same grid that we defined above, we can now discretize

$$\frac{\partial \omega}{\partial t} = -(\psi_y \omega_x - \psi_x \omega_y) + \frac{1}{Re} \nabla^2 \omega$$

with the initial condition

$$\omega(x, y, t = 0) = 0$$

to be able to step step the vorticity in time. We get that we can step from step n to step $n+1$ at the interior points $i = 1, \dots, N-1$, $j = 1, \dots, N-1$ by using central differencing in space and forward differencing in time [5, Chapter 2.7]

$$\omega_{i,j}^{n+1} = \omega_{i,j}^{n} -$$
$$-\Delta t \left(\frac{\psi_{i,j+1}^n - \psi_{i,j-1}^n}{2h} \frac{\omega_{i+1,j}^n - \omega_{i-1,j}^n}{2h} + \frac{\psi_{i+1,j}^n - \psi_{i-1,j}^n}{2h} \frac{\omega_{i,j+1}^n - \omega_{i,j-1}^n}{2h} \right) +$$
$$+ \frac{\Delta t}{Re} \frac{\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1} - 4\omega_{i,j}}{h^2}$$

We note that to solve for the interior points we need the values of ψ and ω on the boundary. We know that $\psi = 0$ on the boundary, but we do not know what ω is on the boundary. Instead, we are given that the normal derivatives of ψ are equal to the imposed tangential velocities

$$\frac{\partial \psi}{\partial n} = U_{\text{wall}}$$

We therefore need to find approximation for ω on the boundary.

Consider the bottom wall $y = 0$. Recall that, by definition,

$$\omega = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$$

On this wall, we know that $v = 0$ since there is no flow across the wall, so that only

$$\omega = -\frac{\partial u}{\partial y}$$

remains. We can now use this to approximate the vorticity at a quarter point from the boundary and take this as our approximation to the vorticity on the boundary [5, Chapter 2.7]

$$\omega_{i,1/4} = -\frac{u_{i,1/2} - U_{\text{wall}}}{h/2} \text{ for } i = 1, \ldots, N-1$$

For this, we need to approximate $u_{\frac{1}{2}}$ which we can do by recalling our definition of u in terms of ψ

$$u = \frac{\partial \psi}{\partial y} \approx \frac{\psi_{i,1} - \psi_{i,0}}{h} \text{ for } i = 1, \ldots, N-1$$

Therefore, we get that our approximation for the vorticity at the quarter point is:

$$\omega_{i,0} \approx \omega_{i,1/4} = -\frac{(\psi_{i,1} - \psi_{i,0})/h - U_{\text{wall}}}{h/2} \text{ for } i = 1, \ldots, N-1$$

We must be careful to check the size of the errors that we are committing by using this approximation since due to the nature of our PDEs these errors can propagate and make the overall accuracy of our method smaller. We can use Taylor's theorem to find

$$\psi_{i,1} = \psi_{i,0} + \left.\frac{\partial \psi}{\partial y}\right|_{i,0} h + \left.\frac{\partial^2 \psi}{\partial y^2}\right|_{i,0} \frac{h^2}{2} + O(h^3) =$$

$$= \psi_{i,0} + U_{\text{wall}} h - \omega_{i,0} \frac{h^2}{2} + O(h^3) \text{ for } i = 1, \ldots, N-1$$

where we exploit the fact that $(\partial\psi/\partial y)_{i,0} = U_{\text{wall}}$ on $y = 0$, and $\omega = -\nabla^2 \psi$ with $\partial^2\psi/\partial x^2 = 0$ on the wall since $\psi = 0$ is our boundary condition, so that $\omega_{i,0} = (\partial^2\psi/\partial y^2)_{i,0}$. We can then rearrange this to get that

$$\omega_{i,0} = -\frac{\psi_{i,1} - \psi_{i,0} - U_{\text{wall}} h}{h^2/2} + O(h) =$$

$$= -\frac{(\psi_{i,1} - \psi_{i,0})/h - U_{\text{wall}}}{h/2} + O(h) \text{ for } i = 1, \ldots, N-1$$

which is not good for us since it is going to break the second order accuracy that we achieved with our discretization. We can improve the accuracy of the approximation for the value on the boundary by linearly extrapolating between the quarter point and the one full interior point

$$\omega_{i,0} = (4\omega_{i,1/4} - \omega_{i,1})/3 + O(h^2)$$

The derivation for this is not given in the book, but we do it here for completeness. We start by showing the extrapolation part. Suppose that for $i = 1, \ldots, N-1$ we write

$$\omega_i(y) = A + By$$

We are interested in extrapolating for $\omega_{i,0} = \omega_i(y = 0) = A$. Note that

$$\omega_i(y = h/4) = A + Bh/4 = \omega_{i,1/4}$$

$$\text{and } \omega_i(y = h) = A + Bh = \omega_{i,1}$$

By taking a linear combination of these equations, we get

$$4A + Bh - A - Bh = 4\omega_{i,1/4} - \omega_{i,1}$$

$$\implies \omega_{i,0} = A = (4\omega_{i,1/4} - \omega_{i,1})/3$$

To find the error, we use Taylor expansions

$$\omega(x = ih, y = 1) = \omega_(x = ih, y = 0) + \left.\frac{\partial\omega}{\partial y}\right|_{x=ih,y=0} h + O(h^2)$$

$$\omega(x = ih, y = h/4) = \omega(x = ih, y = 0) + \left.\frac{\partial\omega}{\partial y}\right|_{x=ih,y=0} \frac{h}{4} + O(h^2)$$

By taking four times the second equation minus the first, we obtain

$$\omega(x = ih, y = h/4) - \omega(x = ih, y = 1) = 3\omega(x = ih, y = 0) + O(h^2)$$

so that

$$\omega(x = ih, y = 0) = \frac{4\omega(x = ih, y = h/4) - \omega(x = ih, y = 1)}{3} + O(h^2)$$

Similarly, for i and j in $i = 1, \ldots, N-1$, it can be shown that

$$\omega_{i,N} = -\frac{U_{\text{wall}} - (\psi_{i,N} - \psi_{i,N-1})/h}{h/2} + O(h) \text{ on } y = 1$$

$$\omega_{0,j} = \frac{-(\psi_{1,j} - \psi_{0,j})/h - U_{\text{wall}}}{h/2} + O(h) \text{ on } x = 0$$

$$\omega_{N,j} = \frac{U_{\text{wall}} - (-(\psi_{N,j} - \psi_{N-1,j})/h)}{h/2} + O(h) \text{ on } x = 1$$

and we can use the same formula as above to linearly extrapolate to get the second order accurate approximations.

3.5 Solving directly for the steady state

We will be interesting in finding the steady state solution of the system (note that we know one exists for our driven cavity problem, but that is not the case in general). It will turn out that our calculations will be much faster if we exploit the fact that $\partial\omega/\partial t = 0$ at the steady state and then step directly to it. To this end, in addition to the iterative time-stepping solution method considered above, we will also look at how to frame the problem as finding a root of a nonlinear system of equations $\mathbf{f}(\mathbf{x}) = \mathbf{0}$ and then use Newton-Raphson method to solve it (the implementation details are presented in chapter 5.2).

Our equations reduce to

$$\nabla^2\psi = -\omega \iff \nabla^2\psi + \omega = 0 \text{ and}$$

$$-(\psi_y\omega_x - \psi_x\omega_y) + \frac{1}{Re}\nabla^2\omega = 0$$

along with the usual boundary and initial conditions. We can then discretize these identically as we did above to obtain a set of nonlinear equations for the unknown streamfunction ψ and vorticity ω that are all equal to zero on the right-hand side. This is exactly what we want, and we can then proceed with our root-finding algorithm to simultaneously find a solution to both the Poisson problem and the vorticity evolution equation.

We now look at the details for discretizing the equations and the form of $\mathbf{f}$. In our case, the entries of $\mathbf{f}$ will be the will be the interior point values of the equations presented at the beginning of this chapter discretized on our usual grid

$$\frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}}{h^2} + \omega_{i,j} = 0 \text{ and}$$

$$\frac{\omega_{i+1,j} + \omega_{i-1,j} + \omega_{i,j+1} + \omega_{i,j-1} - 4\omega_{i,j}}{Re \cdot h^2} +$$

$$+\frac{\psi^n_{i+1,j} - \psi^n_{i-1,j}}{2h}\frac{\omega^n_{i,j+1} - \omega^n_{i,j-1}}{2h} - \frac{\psi^n_{i,j+1} - \psi^n_{i,j-1}}{2h}\frac{\omega^n_{i+1,j} - \omega^n_{i-1,j}}{2h} = 0$$

along with the usual boundary and initial conditions presented in chapters 3.3 and 3.4. However, we need to rewrite the boundary conditions for vorticity so that they also have a zero on the right-hand side. For example, for the $y = 1$ boundary, we rewrite as

$$\omega_{i,N} = (4\omega_{i,N-1/4} - \omega_{i,N-1})/3 = \left(-4\frac{U_{\text{wall}} - (\psi_{i,N} - \psi_{i,N-1})/h}{h/2} - \omega_{i,N-1}\right)/3$$

$$\implies 3\omega_{i,N}h^2 = -8(U_{\text{wall}}h - (\psi_{i,N} - \psi_{i,N-1})) - \omega_{i,N-1}h^2$$

$$\implies (3\omega_{i,N} + \omega_{i,N-1})h^2 + 8(U_{\text{wall}}h - (\psi_{i,N} - \psi_{i,N-1})) = 0$$

and similarly for the other boundaries.

With this form of $\mathbf{f}(\mathbf{x})$, we find that we have a nonlinear system of $2(N-1)^2 + 4(N-1)$ equations (twice $(N-1)^2$ for the interior points for ψ and ω, and $(N-1)$ for the boundary conditions of ω on each side) which is going to be the dimension of **f**. We note that we do not include the equations for the boundary conditions $\psi = 0$ and the four corners of the grid are not included for the vorticity too since there the equations just give the trivial $0 = 0$ equation.

3.6 Discretizing the fluid velocity

After finding the streamfunction and the vorticity using one of the methods above, we will be interested in calculating the flow field which is why we embarked on solving the Navier-Stokes equations in the first place. The steady state flow components on the interior can be approximated by using second-order accurate central differencing

$$u = \frac{\partial \psi}{\partial y} \approx \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2h}$$

$$v = -\frac{\partial \psi}{\partial x} \approx -\frac{\psi_{i+1,j} - \psi_{i-1,j}}{2h}$$

and we note that we know the velocity components on the boundaries from the no slip and no penetration boundary conditions.

3.7 Discretizing the calculation of the force

We will also want to calculate the non-dimensionalized force on the top of the plate which we can do as follows:

$$F = \int_0^1 \left.\frac{\partial u}{\partial y}\right|_{y=1} dx \approx \sum_{i=0}^{N} \left.\frac{\partial^2 \psi}{\partial y^2}\right|_{j=N} h$$

We can evaluate the shear stress as

$$\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{j=N} = \left.\frac{\partial^2 \psi}{\partial y^2}\right|_{j=N-1} + O(x)$$

by using a Taylor expansion. From here, we can use central differencing on $\frac{\partial^2 \psi}{\partial y^2}$ at $j = N-1$ to obtain

$$\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{j=N} = \frac{\psi_{i,N} - 2\psi_{i,N-1} + \psi_{i,N-2}}{h^2} + O(x)$$

We would like to have second order error, so we can again extrapolate linearly from the two interior points $N-1$ and $N-2$ onto the boundary N like so

$$\frac{\partial^2 \psi}{\partial y^2} = A + By$$

and we note that we want to extrapolate for

$$\left.\frac{\partial^2 \psi}{\partial y^2}\right|_{j=N} = \frac{\partial^2 \psi}{\partial y^2}(y=1) = A + B$$

Just as we did above for the boundary conditions for the vorticity, we note that

$$\frac{\partial^2\psi}{\partial y^2}(y = 1 - h) = A + B - Bh = \frac{\partial^2\psi}{\partial y^2}_{i,N-1} \text{ and}$$

$$\frac{\partial^2\psi}{\partial y^2}(y = 1 - 2h) = A + B - 2Bh = \frac{\partial^2\psi}{\partial y^2}_{i,N-2}$$

By taking two times the first equation minus the second one, we get

$$2(A + B) - 2Bh - (A + B) + 2Bh = 2\frac{\partial^2\psi}{\partial y^2}_{i,N-1} - \frac{\partial^2\psi}{\partial y^2}_{i,N-2}$$

$$\implies \left.\frac{\partial^2\psi}{\partial y^2}\right|_{j=N} = A + B = 2\frac{\partial^2\psi}{\partial y^2}_{i,N-1} - \frac{\partial^2\psi}{\partial y^2}_{i,N-2}$$

For the errors, we can use a similar argument as above for the vorticity on the boundary as and expand $\frac{\partial^2\psi}{\partial y^2}_{i,N-1}$ and $\frac{\partial^2\psi}{\partial y^2}_{i,N-2}$ around $\frac{\partial^2\psi}{\partial y^2}_{i,N}$ to find that this extrapolated value is also second order accurate.

4 Numerical approaches for the Poisson problem

Recall that we want to solve

$$\frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}}{h^2} = -\omega_{i,j}$$

on the interior with $\psi = 0$ on the boundary. so that the so-called stencil for this problem is that $\psi_{i,j}$ only depends on the values above, below and to the sides of it (see figure 3). This is a system of $(N-1)^2$ linear equations (since we know ψ on the boundary), so the three methods discussed below will just be solvers for linear systems of equations.

The Poisson problem is a linear PDE, so we can choose ω such that we can find an analytic solution for ψ. In this project, we will test our numerical methods by asking them to solve

$$\nabla^2\psi = -2\pi^2 \sin\pi x \sin\pi y \text{ in } 0 \le x \le 1, 0 \le y \le 1$$

with $\psi = 0$ on the boundary for which we know that the analytic solution is

$$\psi = \sin\pi x \sin\pi y$$

4.1 The iterative SOR method

The simplest method to solve the system above is iterative successive over-relaxation (SOR) method which is a modification of the Gauss-Seidel method

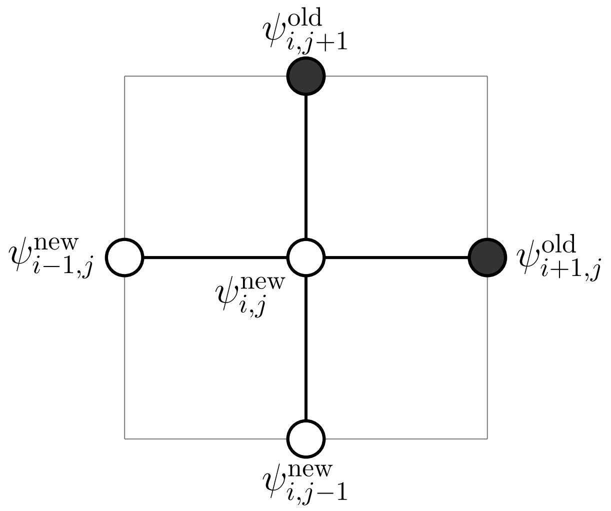

$\psi_{i,j+1}^{\text{old}}$
$\psi_{i-1,j}^{\text{new}}$
$\psi_{i,j}^{\text{new}}$
$\psi_{i+1,j}^{\text{old}}$
$\psi_{i,j-1}^{\text{new}}$

Figure 3: The stencil for the Poisson problem

[5, Chapter 2.3]. The usual Gauss-Seidel method updates the values by sweeping through the interior of the grid, first with $i = 1$ and $j = 1 \to N-1$, then with $i = 2$ and $j = 1 \to N-1$, continuing until $i = N-1$ and $j = 1 \to N-1$ (see figure 4). We can rearrange the discretized equation above to get that at each grid point, the old value is replaced by using the update rule

$$\psi_{i,j}^{\text{new}} = (\psi_{i+1,j}^{\text{old}} + \psi_{i-1,j}^{\text{new}} + \psi_{i,j+1}^{\text{old}} + \psi_{i,j-1}^{\text{new}} + h^2\omega_{i,j})/4$$

Note that superscripts "new" are used for the values that have already been updated in the current sweep, while "old" is used for will only be updated later in the sweep.

The method requires about N^2 sweeps to converge [5, Chapter 2.3], and N^2 values have to be updated during each sweep, giving a total of $O(N^4)$ total operations to solve a system of size $N \times N$. This is already an improvement over the $O(N^6)$ operations needed to directly invert an $N \times N$ matrix.

We can further reduce the number of operations by modifying the iteration scheme to include a relaxation parameter r. The SOR iterate is then

$$\psi_{i,j}^{\text{new}} = (1-r)\psi_{i,j}^{\text{old}} + r(\psi_{i+1,j}^{\text{old}} + \psi_{i-1,j}^{\text{new}} + \psi_{i,j+1}^{\text{old}} + \psi_{i,j-1}^{\text{new}} + h^2\omega_{i,j})/4$$

The number of sweeps required for convergence then becomes only $O(N)$ [5, Chapter 2.3], given that we choose an optimal value for the relaxation parameter r. The optimal value of the parameter is problem dependent, but it turns out that the optimal value for our problem and for large N [5, Chapter 2.3] is

$$r_{\text{optimal}} = \frac{2}{1+\pi/N}$$

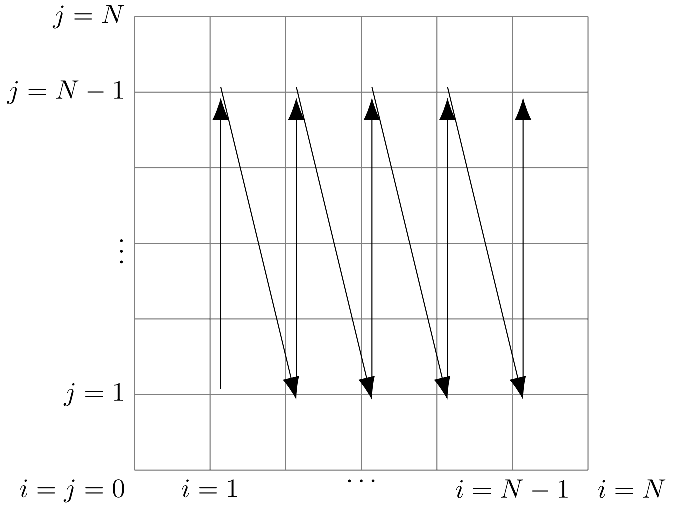


Figure 4: The sweeping through the grid of the Gauss-Seidel method

When coding up our solution, it will be enough to do $4N$ sweeps to converge to a solution, so that there will be $O(N^3)$ operations in total. More details on the convergence analysis and the choice of optimal relaxation parameter for this method please refer to [5, Chapter 2.3] and [2, Chapter 7].

4.2 Non-iterative methods

Alternatively, we can look at non-iterative methods to solve the linear system. To this end, we convert the system to matrix-vector form $A\mathbf{x} = \mathbf{b}$ of size $(N-1)^2$. In our case, the vector of unknowns $\mathbf{x}$ will contain the flattened values of ψ

$$\mathbf{x} = \begin{bmatrix} \psi_{1,1} & \cdots & \psi_{1,N-1} & \psi_{2,1} & \cdots & \psi_{2,N-1} & \cdots & \psi_{N-1,N-1} \end{bmatrix}^T$$

and similarly the vector $\mathbf{b}$ will contain the known values of the vorticity ω

$$\mathbf{b} = -\begin{bmatrix} \omega_{1,1} & \cdots & \omega_{1,N-1} & \omega_{2,1} & \cdots & \omega_{2,N-1} & \cdots & \omega_{N-1,N-1} \end{bmatrix}^T$$

Finally, the matrix A will represent the relationship between $\psi_{i,j}$ to the values $\psi_{i+1,j}$, $\psi_{i-1,j}$, $\psi_{i,j+1}$ and $\psi_{i,j-1}$. Therefore, A will have $-4/h^2$ on the diagonal and four values (or a smaller number of values accordingly if the point is adjacent to the boundary) of $1/h^2$ at the corresponding columns of each row to correctly express the relationship.

Now that we have the matrix-vector form of the system, we can use functions implemented in the `scipy` Python library [8] to find a solution.

4.2.1 Dense linear solver

The naive approach is to treat the system formed above as a dense linear system. The number of operations required to solve such a such system is $O(N^6)$ for a system of size $N \times N$. Another thing to note is that the SOR method operated in place, whereas now we will need to store the matrix A of size $O(N^4)$ in the computer's memory which will limit the grid size N that we can use.

4.2.2 Sparse linear solver

It quickly becomes clear that approaching the system as a dense one has substantial disadvantages. However, we note that the matrix A will only contain up to five non-zero entries in each row which means that it is extremely sparse, especially for larger values of the grid size N. We can exploit this to save both on computation time and memory. The `scipy` library has a special API for dealing with sparse matrices that only stores the non-zero entries of a sparse matrix and has a range of specialized algorithms to perform computations with such matrices faster. We will use the specialized sparse linear system solver. Then, the memory required to store the sparse matrix will be $O(N^2)$, which is optimal since we need the same amount of memory for the vector of unknowns anyway, and the number of operation required to solve the system will be $O(N^3)$. This is asymptotically the same as for the iterative SOR method, but we will see that the sparse solver will be much faster in reality since it has been heavily optimized over the years by the maintainers of the `scipy` library and adapted to utilize multiple CPU cores.

4.3 Results

To check our solvers, we recall that we have chosen the vorticity ω for the Poisson problem such that the exact solution for the streamfunction is $\psi = \sin \pi x \sin \pi y$. Note that we only visualize the solution from the sparse solver since all of the solvers produced essentially the same approximations.

Qualitatively, we can see from figure 5 that the produced approximation indeed has the correct caplike shape of $\sin \pi x \sin \pi y$. Further, figure 6 shows that the horizontal and the vertical middle slices of ψ are (at least graphically) indistinguishable from the sine curve, as they should be. The trend in figure 7 shows the errors go to zero as h^2 which is also the expected behaviour. Quantitatively, we get that the maximum error over the whole grid is $O(10^{-5})$ for $h = 0.01$, so that our approximation is accurate to four significant figures.

4.3.1 Computation time

Since the different methods give the same accuracy, we would like to use the one that has the smallest execution time. Figures 8, 9 and 10 show that the iterative SOR and the sparse solvers indeed have the expected $O(N^3)$ complexity, whereas the dense solver is $O(N^6)$. Therefore, even though our SOR iteration

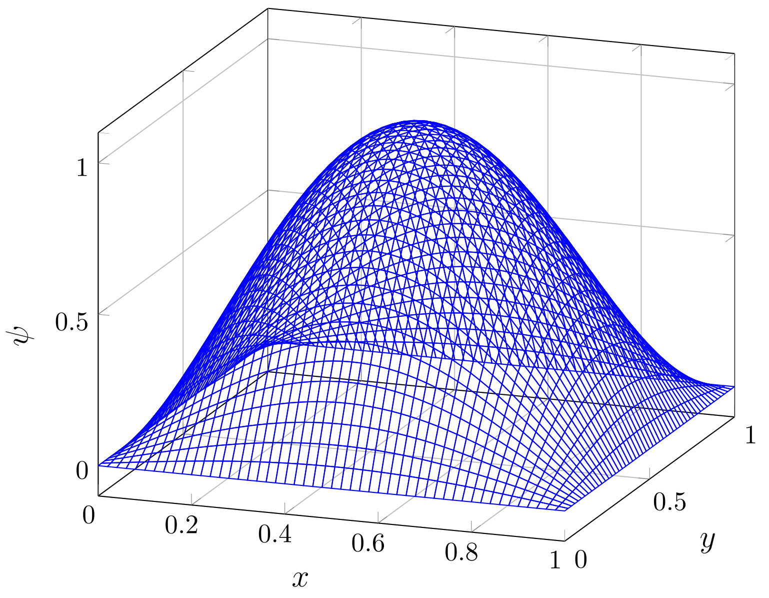


Figure 5: The approximation to ψ produced by the sparse solver.

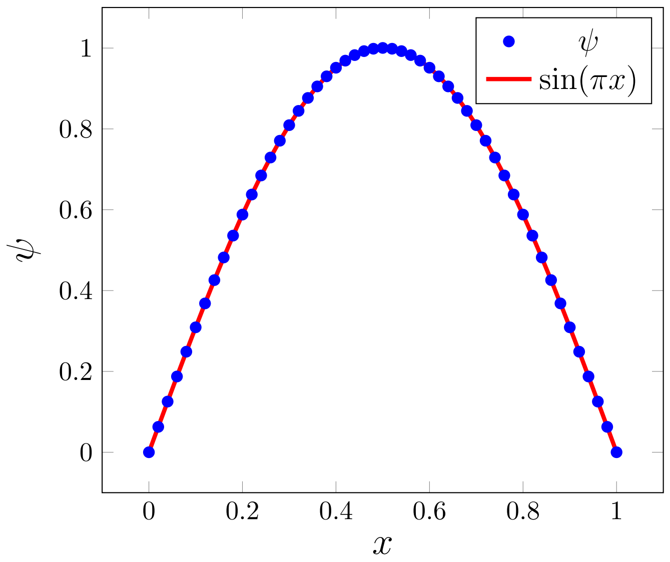
implementation is not optimized and runs only on a single CPU thread, it outperforms the highly optimized dense solver from `scipy` that can utilize multiple CPU cores for larger values of N due to its better asymptotic complexity. Therefore, we are left to choose between the SOR and the sparse solvers. However, table 1 shows that the sparse solver is the clear winner and runs in less that half a second even for the quite large value of $N = 168$. This is it exploits sparsity, is carefully optimized and can run on multiple CPU cores.

Table 1: Running times in seconds of the SOR and sparse solvers

$N \rightarrow$	7	14	28	56	84	112	168
SOR	0.016	0.141	1.162	9.501	32.079	77.298	255.031
Sparse	0.001	0.002	0.007	0.031	0.076	0.226	0.419

4.3.2 Memory usage comparison

Memory usage for each method was obtained by monitoring the program's execution, logging the amount of memory used over time and comparing it to the amount of memory that was used before the method was started. The results are presented in figures 11, 12 and 13 for different values of N. We can see that, as expected, the SOR solver operates in place, the sparse solver's usage is linear in N^2 (i.e. linear in the size of the grid), and the dense solver's usage is linear in N^4 because it is formulated as a dense $N^2 \times N^2$ matrix. For this reason, the



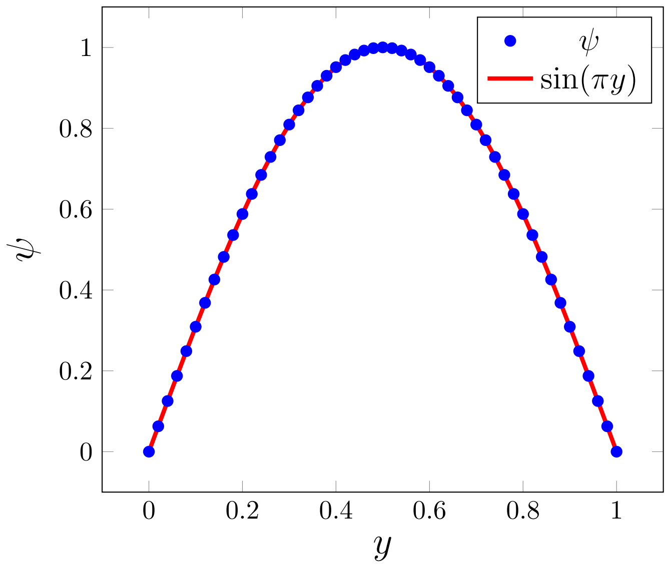


Figure 6: The horizontal ($y = 1/2$) and the vertical ($x = 1/2$) middle slices of the approximation of ψ produced by the sparse solver.

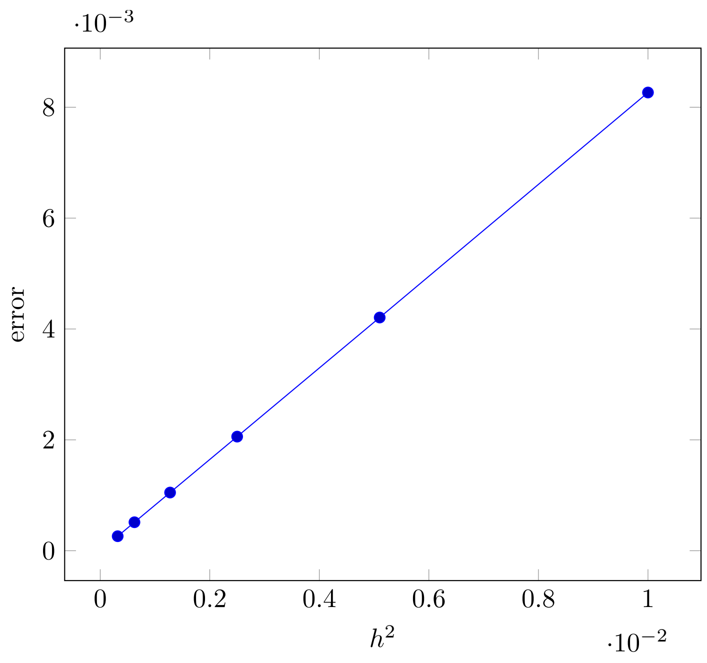


Figure 7: Error sizes the sparse solver for $N = 10, 14, 20, 28, 40, 56$.

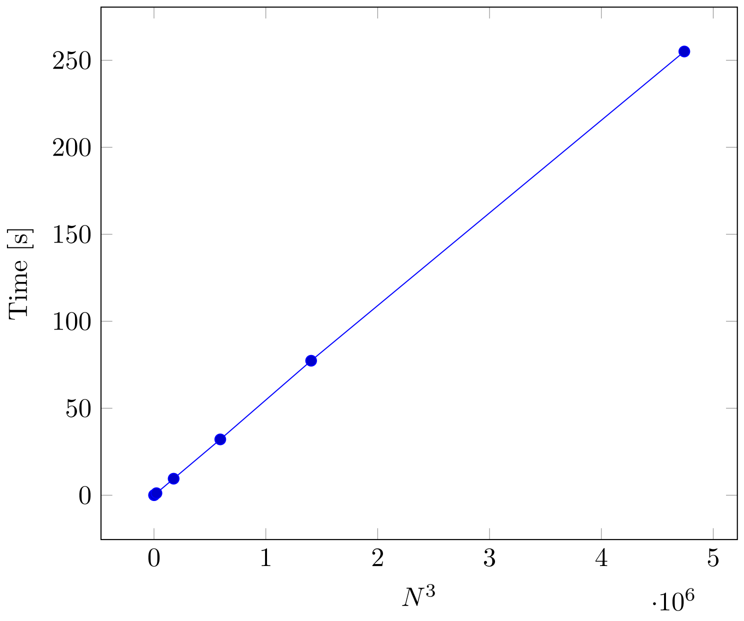


Figure 8: Computation time for the SOR solver for $N = 7, 14, 28, 56, 84, 112, 168$.

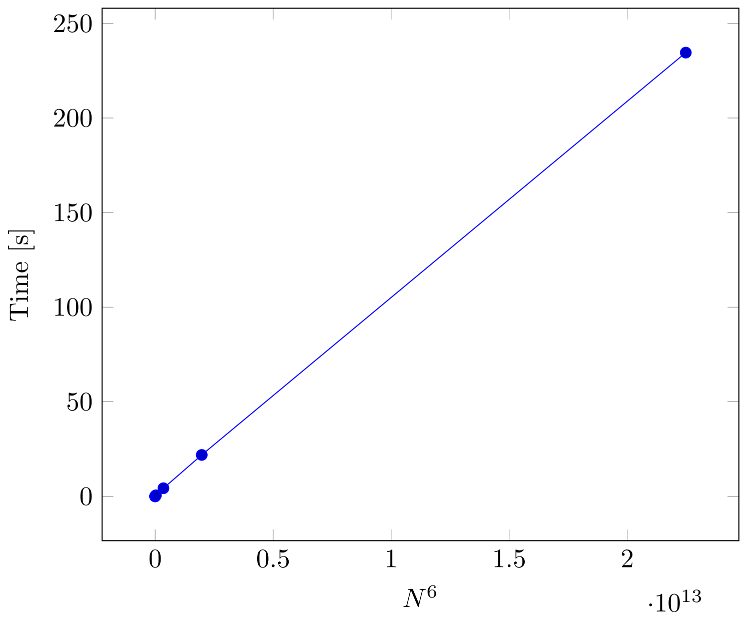


Figure 9: Computation time for the direct solver for $N = 7, 14, 28, 56, 84, 112, 168$.

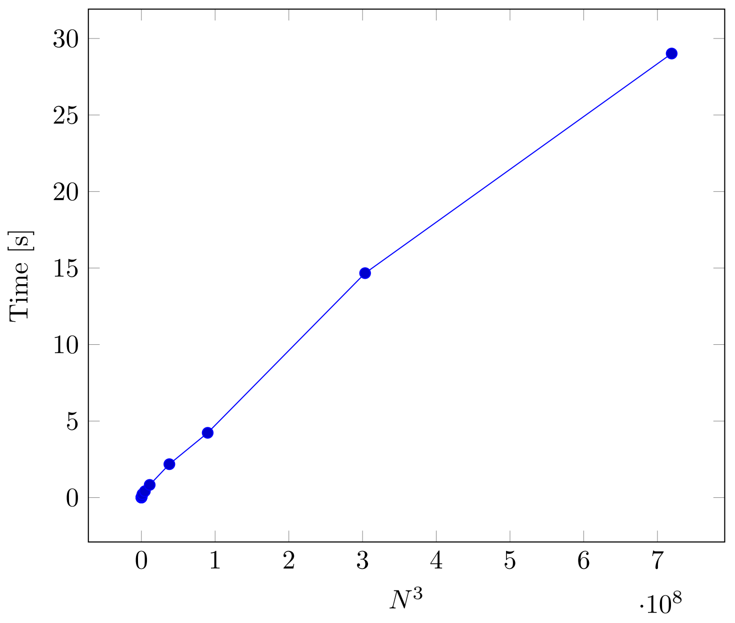


Figure 10: Computation time for the sparse solver for $N = 7, 14, 28, 56, 84, 112, 168, 224, 336, 448, 672, 896$.

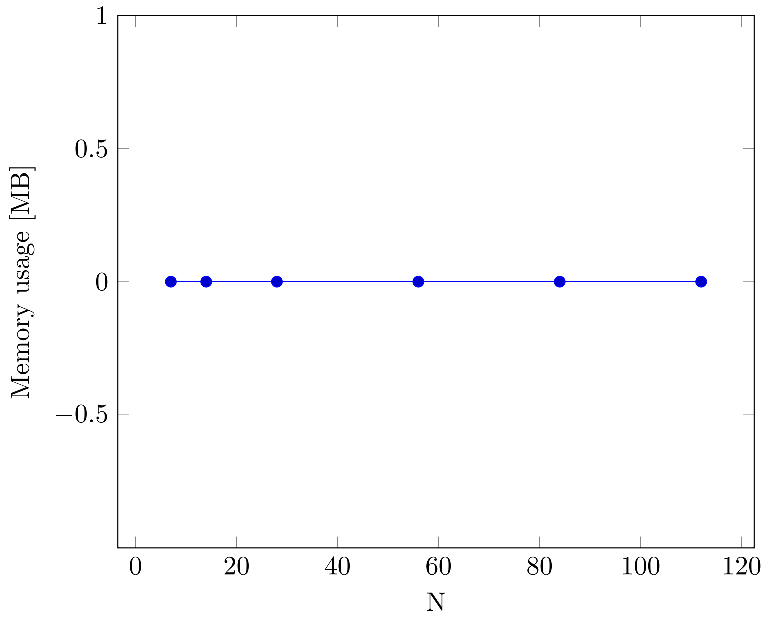


Figure 11: Memory usage for the SOR solver for $N = 7, 14, 28, 56, 84, 112$.

dense solver quickly starts running out of RAM and becomes unviable to use as we increase the grid resolution. Clear winner in terms of memory usage is the SOR solver, but the sparse solver also produces satisfactory results and can be used without problem even for finer grid resolutions (the largest valued tried in this project was $N = 896$).

5 Numerical approaches for the full Navier-Stokes equations

In chapters 3.4 and 3.5 we discussed how to discretize the Navier-Stokes equations (which, recall, we frame in the form of the vorticity evolution equation in this project) and now we are going to look at how to actually implement the different methods on a computer to find the steady state solution. We are going to find that the approach that steps directly to the steady state will be much faster than iterative time-stepping, but we will also discuss why the former approach might not always be valid.

5.1 Solving the problem by time-stepping

The iteration process can be summarized as follows:

1. Initialize the problem.

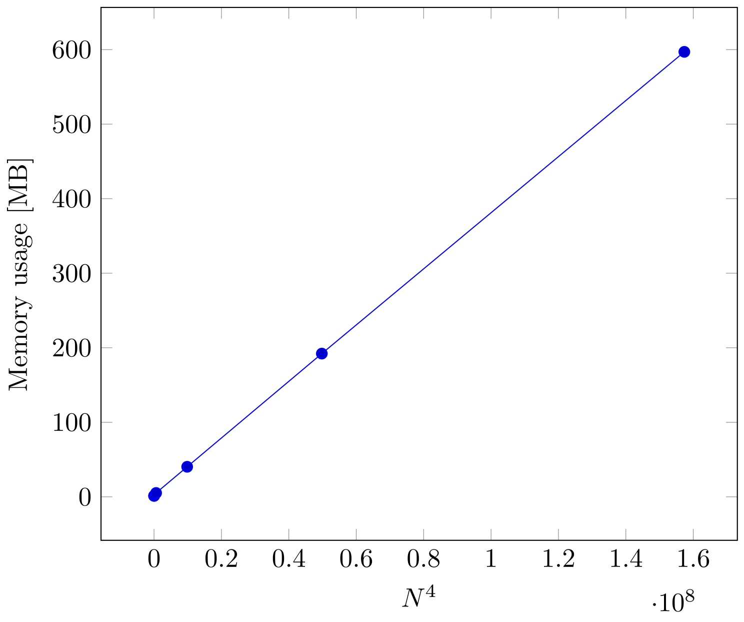


Figure 12: Memory usage for the dense solver for $N = 7, 14, 28, 56, 84, 112$.

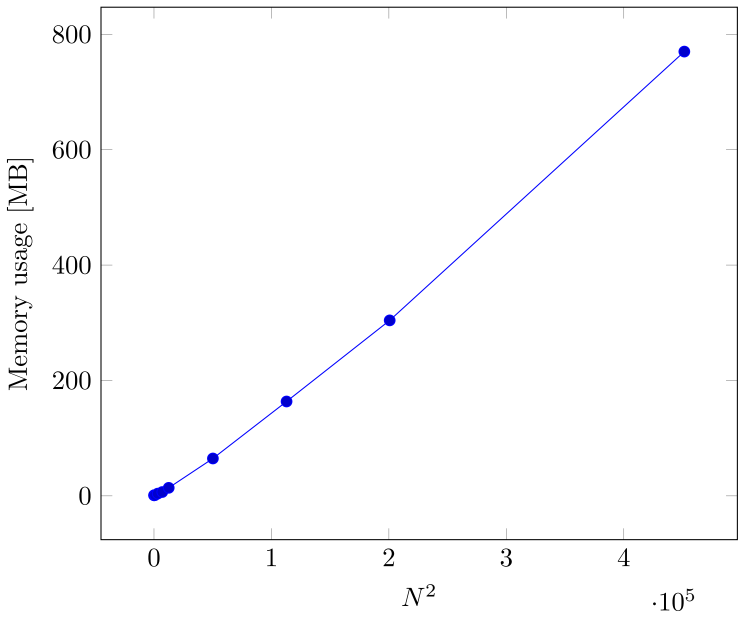


Figure 13: Memory usage for the sparse solver for $N = 7, 14, 28, 56, 84, 112, 224, 336, 448, 672$.

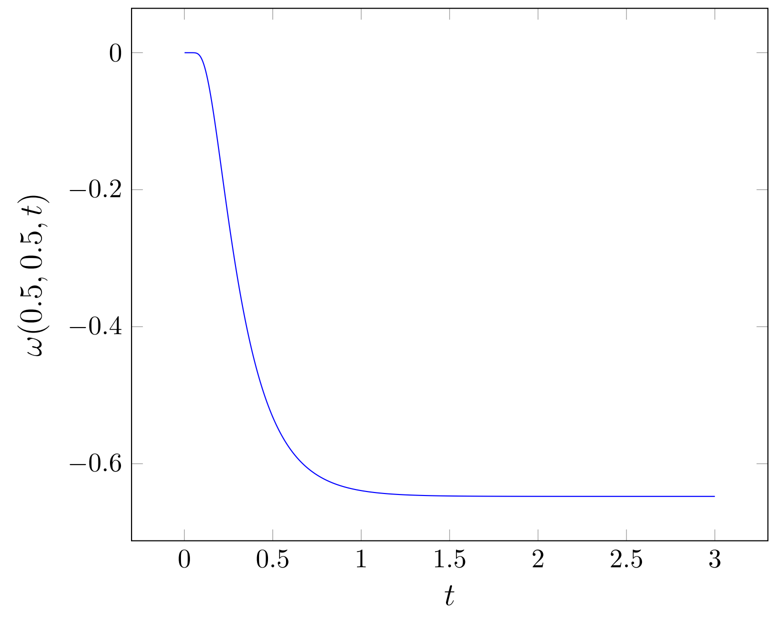


Figure 14: Evolution of the vorticity ω at the middle of the cavity for $N = 40$, $Re = 10$ and $\Delta t = 0.00125$.

2. Update the streamfunction by solving Poisson problem given the vorticity at time t.
3. Calculate the vorticity on the boundaries given the new streamfunction values.
4. Step the vorticity to time-step $t + \Delta t$ where we recall that Δt is the time-step size.
5. Repeat steps 2-4 until a desired time $t = t_{final}$ is reached.

The time until which we run the method is problem-dependent, but can be determined by running a few experiments and checking how long we have to iterate for until the physical quantities settled down (and if they do not, we integrate for as long as we want to see the behaviour of the system for). Figure 14 shows that in our case it is enough to step until $t_{final} = 3$ to reach he steady state which we will be doing for the rest of this chapter.

It is important to choose the spatial resolution h and the time-step size Δt small enough so that our solution does not become numerically unstable, as illustrated in figure 15. It turns out that there are three conditions that we have to satisfy [5, Chapter 2.8]:

1. Stability condition for the time-step size $\Delta t < Re \cdot h^2/4$.
2. Small "grid-Reynolds-number" conditon $h < 1/Re$.

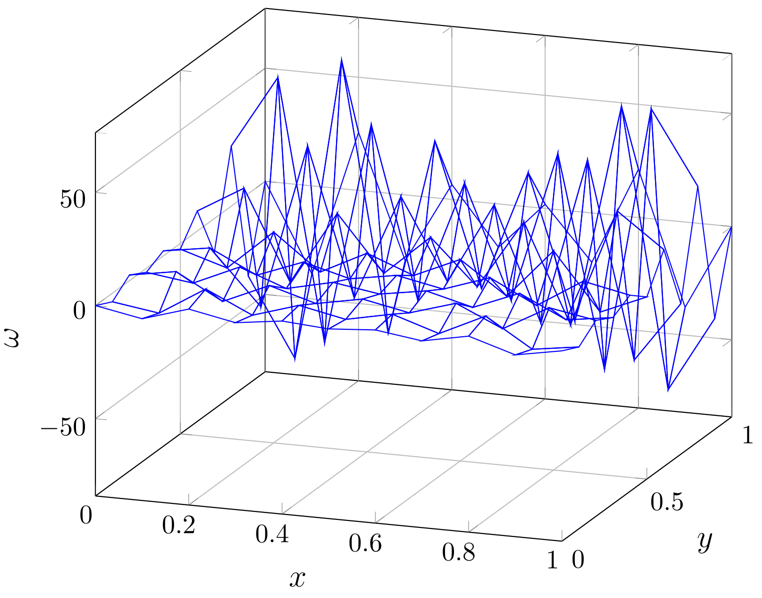


Figure 15: Numerical instability due to time-step size Δt being too large. The vorticity ω for $N = 10$, $Re = 10$ and $\Delta t = 0.035$.

3. Courant–Friedrichs–Lewy (CFL) condition $\Delta t < h/u_{\max}$ where $u_{\max}$ is the maximum size of the velocity on the grid.

It turns out that if we satisfy the first two conditions, then the last one will be automatically satisfied. To be on the safe side, we are going to set the time-step size at 80% of the stability condition, i.e. $\Delta t = Re \cdot h^2/5$. We also note that it turns out that satisfying the above conditions will also ensure that the $O(\Delta t)$ time errors are of the same size as the $O(h^2)$ spatial errors, so there is no need to look for a more advanced second-order time-stepping scheme or further decrease Δt [5, Chapter 2.9].

The stability condition for the time-step size is severely restrictive and will lead to our calculations needing to run much longer to reach the desired final time. More concretely, in order to reach some finite $O(1)$ time we will need to perform $O(1/\Delta t) \propto O(N^2)$ iterations. Recall we will also have to solve the Poisson problem at each step at $O(N^3)$ cost, giving a total of $O(N^5)$ running time. Therefore, doubling the grid resolution N will take about 32 times longer to run. As for the memory usage, the steps after solving the Poisson equation are going to use up $O(N^2)$ memory, and the Poisson problem itself requires $O(N^2)$ memory too given that we use the sparse solver discussed in chapter 4.2.2, so the total is $O(N^2)$. This is optimal considering that storing the matrices for ψ and ω require $O(N^2)$ memory.

5.2 Solving directly for the steady state

As mentioned in chapter 3.5, we will be using the Newton-Raphson iteration method to find a root of our nonlinear vector valued-function $\mathbf{f}(\mathbf{x})$. Starting from an initial guess for the vector of unknowns $\mathbf{x}^0$, the Newton-Raphson iteration algorithm finds a root of the known vector-valued function $\mathbf{f}(\mathbf{x})$ by using the following update rule [7, Chapter 4.3]

$$\mathbf{x}^{k+1} = \mathbf{x}^k + [J(\mathbf{x}^k)]^{-1}\mathbf{f}(\mathbf{x}^k) \text{ for } k \geq 0$$

where $(J(\mathbf{x}))^{-1}$ is the inverse of the Jacobian matrix $J(\mathbf{x})$ of first order partial derivatives of $\mathbf{f}$. We will use $\mathbf{x}^0 = \mathbf{0}$ as the initial guess. The iterations will be stopped when $|\mathbf{f}(\mathbf{x})| \leq \text{TOL}$ in which case $\mathbf{f}(\mathbf{x}) \approx \mathbf{0}$ and the algorithm has converged, or the maximum allowed number of iterations is exceeded. Here TOL is the tolerance parameter, which is usually set to some small value, such as 10^{-8}, and the maximum number of iterations is usually set from around 10 to 20.

In practice, for $k \geq 0$ we will actually iterate as

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Delta\mathbf{x} \text{ with}$$

$$\Delta\mathbf{x} = -[J(\mathbf{x}^k)]^{-1}\mathbf{f}(\mathbf{x}^k) \iff J(\mathbf{x}^k)\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}^k)$$

We do this to save on computation time as instead of having to numerically invert the matrix $J(\mathbf{x}^k)$, which would take $O(N^6)$ time, we will be solving the sparse linear system $J(\mathbf{x}^k)\Delta\mathbf{x} = -\mathbf{f}(\mathbf{x}^k)$ which, as we have already seen in our discussions above, can be done very efficiently in $O(N^3)$ operations. The sparsity of the system comes from the fact comes from the fact that each row of $\mathbf{f}$ will only depend on a small number of entries of the input vector $\mathbf{x}$.

We have specified an explicit formula for the entries of $\mathbf{f}$ in chapter 3.5, but it is not so easy to do so for the Jacobian matrix $J(\mathbf{x})$. Fortunately, we can approximate its j^{th} column numerically by using forward differencing

$$J_j(\mathbf{x}) = \frac{\mathbf{f}(\mathbf{x} + h\mathbf{e}_j) - \mathbf{f}(\mathbf{x})}{h}$$

where $\mathbf{e}_j$ is the j^{th} column of the identity matrix (i.e. the j^{th} standard basis vector) and h is the spatial step size. We can then just stack these approximations column by column to obtain the full matrix

$$J(\mathbf{x}) = \begin{bmatrix} J_1(\mathbf{x}) & \cdots & J_m(\mathbf{x}) \end{bmatrix}$$

where m is the dimension of $\mathbf{f}$.

The advantage of using Newton-Raphson method to approximate the steady state is that, given that the initial guess is close enough to the root, it is going to converge quadratically, which means that the error at the next time step is going to be less than or equal to the square of the error at the current time step (a more rigorous treatment of the convergence guarantees for this method can be found in [7, Chapter 4.3]). This means that instead of the $O(N^2)$ iterations required to reach the steady state by stepping the vorticity, we can find an approximation in just a few iterations (usually less that 10), each of which has a cost of $O(N^3)$ to solve a sparse linear system of size of $O(N^2)$. The total running time is therefore $O(N^3)$ which is a substantial improvement over the previous method. In terms of memory, the system that is solved is sparse, so only $O(N^2)$ memory is required which is optimal.

The only disadvantage of assuming $\partial\omega/\partial t = 0$ and finding a root of the resulting nonlinear equations is the subtle fact that when solving a new problem it is not known in advance whether the steady state will be stable, or if one exists in the first place. The root finding algorithm (Newton-Raphson in our case) would just try to approximate the steady state solution. This means it would find the unstable solution without us knowing about it, or not converge at all (only stopping when the maximum number of iterations is reached) and we would be left with nothing if the steady state does not exist. In contrast, with a time-stepping algorithm we would be able to tell that the steady state solution is unstable as we step closer and closer to it, or would find the approximation to the unsteady solution. Moreover, even if a steady state exists, Newton-Raphson might not converge to it out of the box as it requires the initial guess to be close enough. Therefore, even though the performance improvement of using the method is vast, we might be left with nothing if we are not careful enough.

5.3 Results

5.3.1 Physical quantities

The steady state vorticity, streamfunction and the approximate flow field can be seen in figures 16, 17 and 18, respectively. We can see that the resulting flow is a vortex around the point $(0.5, 0.8)$ at which the streamfunction has a minimum and so it's derivatives (which are related to the velocity components) are zero. We can also see that velocities are higher closer to the top of the cavity and negligible towards $y = 0$ which is also consistent with the streamfunction ψ being shifted more towards the top and having a rapidly changing magnitude which implies large first derivatives. The streamfunction is symmertic about $x = 1/2$ which is expected for this moderate value of Reynolds number. Finally, the plot for the flow field shows very weak reversed eddy currents at the bottom corners.

The vorticity is largest and negative next to the moving lid, and positive on the sidewalls. This makes sense if we recall that $\omega = \partial v/\partial x - \partial u/\partial y$ and notice that the horizontal velocity u is positive and quickly increasing with y and the vertical velocity v is negligible and does not change much with x next to the top boundary. Similarly, v increases with x and u barely change with y on the sides which gives the positive vorticity there.

The steady state force on the top of the lid can be calculated to be $F = 3.9047$ using $N = 56$. We can also see approximately linear relationship between force and h^2 in figure 19 which gives reassurance that we indeed have second order accuracy for the calculation of force, just as derived above.

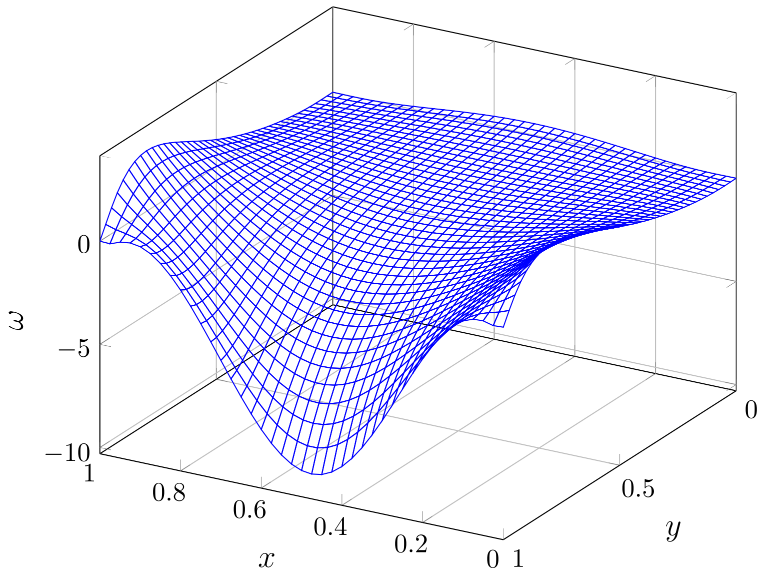


Figure 16: The approximation to the vorticity ω produced by the time-stepping algorithm.

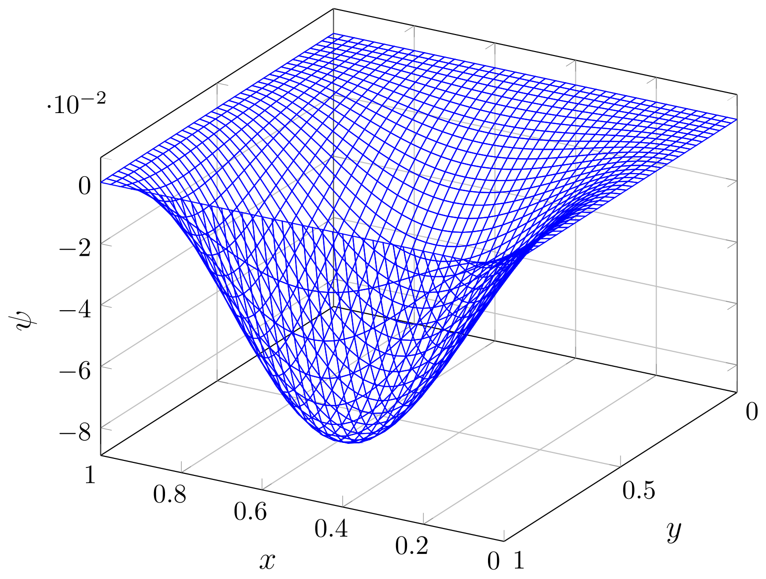


Figure 17: The approximation to the streamfunction ψ produced by the time-stepping algorithm.

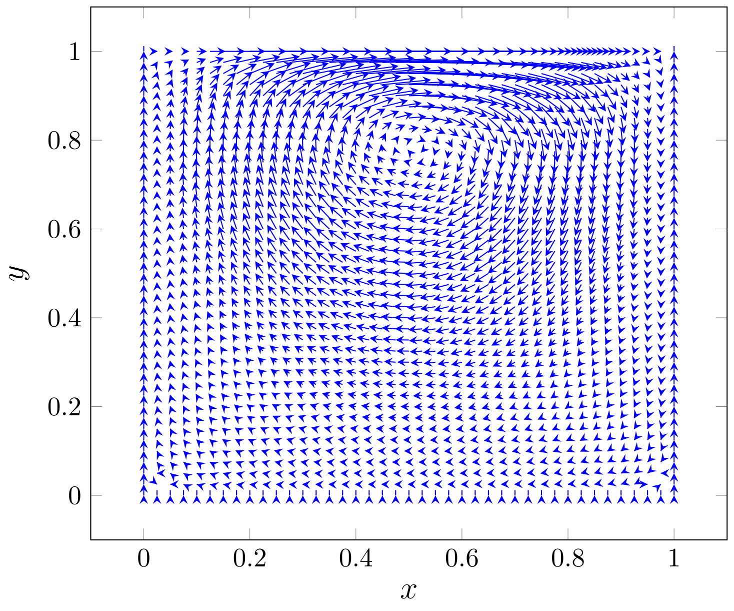


Figure 18: The approximation to the flow field. Note that there is no flow on the bottom and the sides, but the graphing package incorrectly tries to put an arbitrary arrow there anyway.

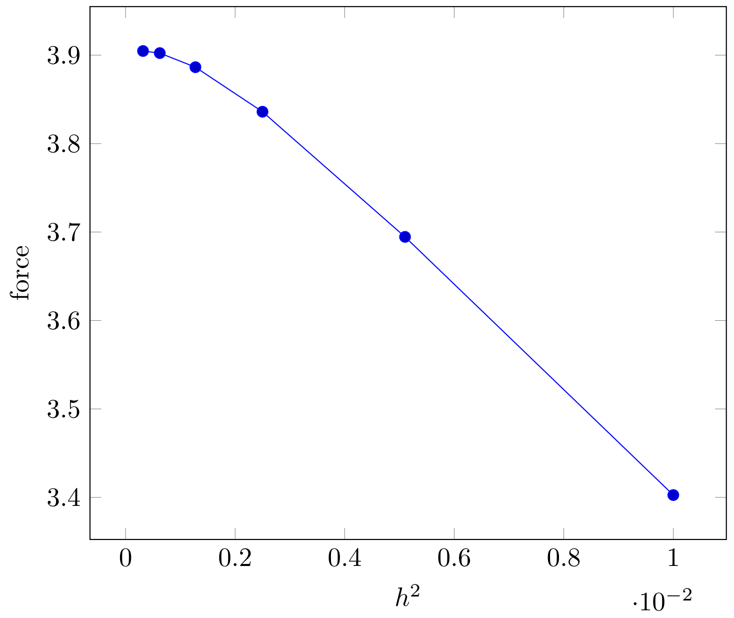


Figure 19: The steady state force at $Re = 10$ for $N = 10, 14, 20, 28, 40, 56$.

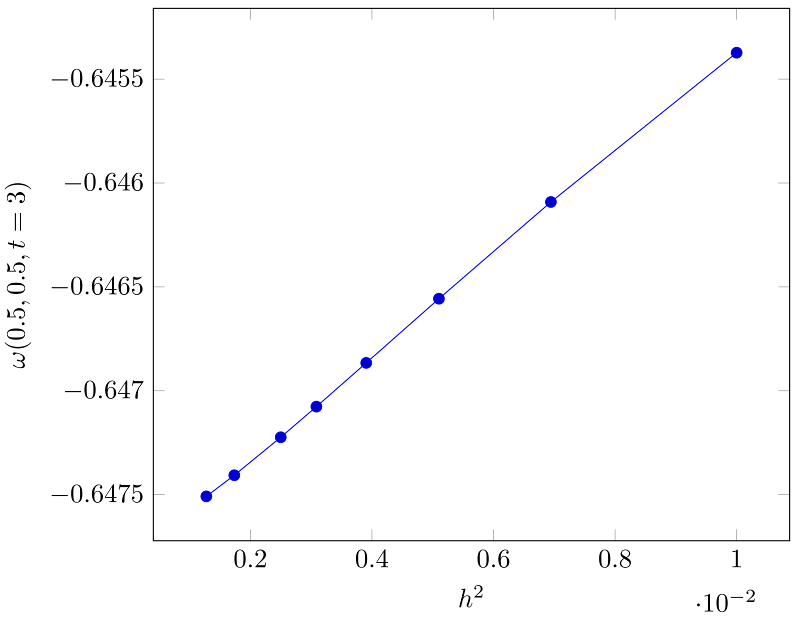


Figure 20: The steady state vorticity at the middle $\omega(0.5, 0.5, t = 3)$ against h^2 for $N = 10, 12, 14, 16, 18, 20, 24, 28$.

Further reassurances of accuracy of our method are the linear relationship between the steady state vorticity at the middle $\omega(0.5, 0.5, t = 3)$ and h^2, as well as Δt. These can be observed in figures 20 and 21. Note that the errors made due to changing the time-step size Δt are of the same size as changing the spatial-step size h^2 which the proportional to the distance between the lines for different grid resolutions in figure 21. This confirms that our choice of $\Delta t = Re \cdot h^2/5$ maintains the second order spatial accuracy of our discretization which we expected by our analysis from before.

5.3.2 Computation time

The trends in figures 22 and 23 show that, as expected, the iterative time-stepping method indeed runs in $O(N^5)$ time, whereas the Newton-Raphson method takes $O(N^3)$ time. We notice that for $N = 70$, the iterative algorithm takes over fives minutes to run, while Newton-Raphson finishes in less that twenty seconds. Therefore, whenever possible, it is desirable to employ the latter method in our experiments, especially for larger grid resolutions.

6 Conclusion

In this project we looked at how to numerically solve the famous Navier-Stokes equations that govern fluid flow. We considered the driven cavity problem and

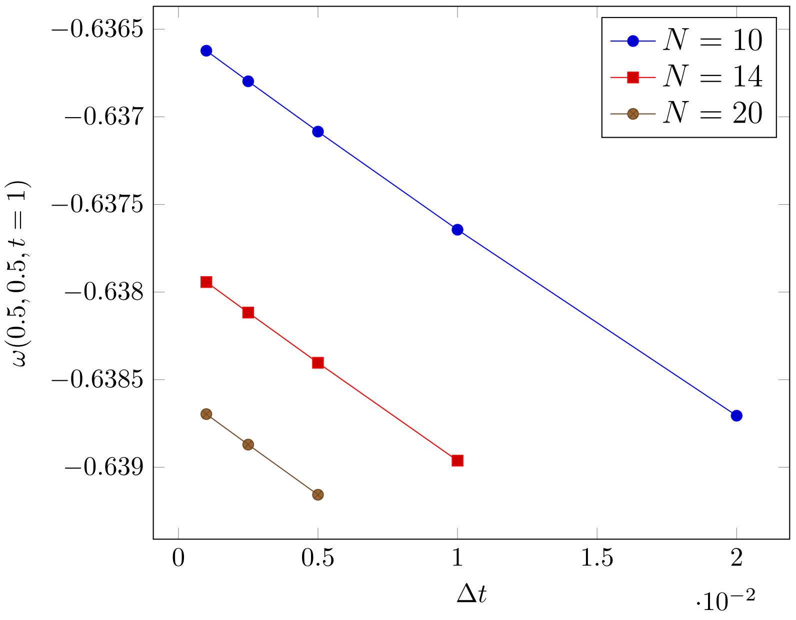


Figure 21: Test for $O(\Delta t)$ accuracy in $\omega(0.5, 0.05, t = 1)$ for $Re = 10$ and $N = 10, 14, 20$.

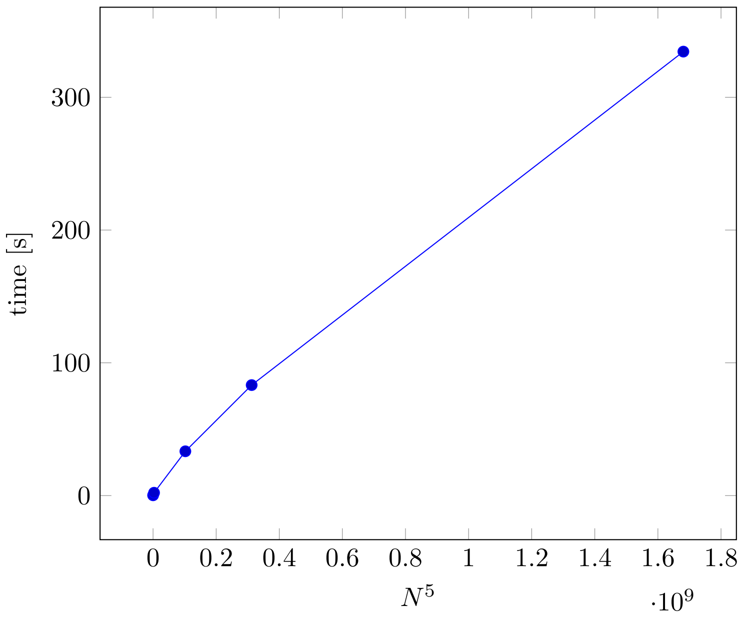


Figure 22: Speed of the iterative solver for $N = 10, 20, 40, 50, 70$.

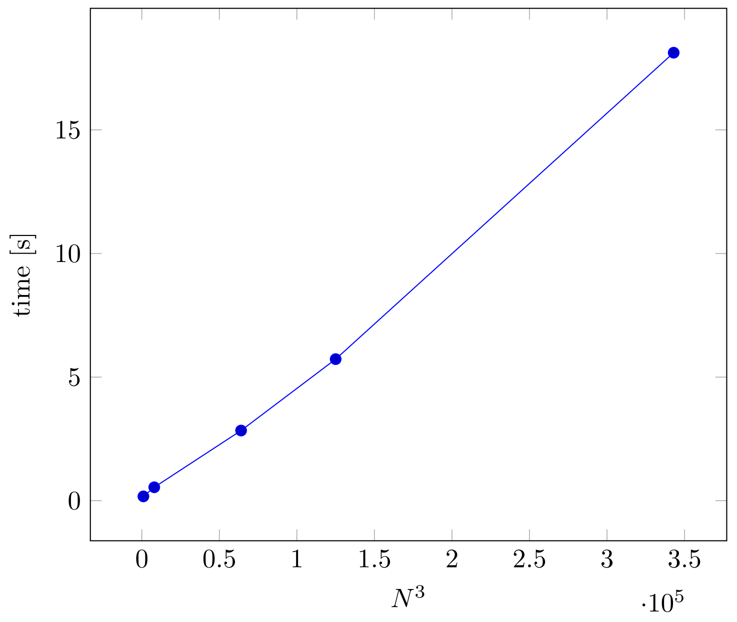


Figure 23: Speed of the Newton-Raphson solver for $N = 10, 20, 40, 50, 70$.

how to discretize it and what the resulting error sizes will be. We then decoupled the problem into the Poisson problem and the vorticity evolution equation and looked at how we can solve each of them, and finally how to iterate between them to get the final approximation.

We found that for the Poisson problem, the best method to use in terms of computation time and having acceptable memory usage is the sparse linear system solver from the `scipy` Python library. For the vorticity evolution equation, we can only use the iterative time-stepping algorithm in general. However, we have also seen that if we assume that a steady state exists, then we can find a solution for it much faster using Newton-Raphson algorithm for finding roots of nonlinear equations.

Some possible future extensions for this project are:

- Making Newton-Raphson method more general, as in this work we have only considered the method under the steady state assumption. However, as long as we can formulate a given problem as finding a root of a vector-valued nonlinear function, we would be able to apply Newton-Raphson method to solve it.
- Investigate whether performing computations on a GPU instead of a CPU could give an improvement in terms of speed. GPU is a piece of hardware specialized to perform matrix computations incredibly fast. Our codes involve a lot of matrix operation, so it would be reasonable to expect that significant performance gains by using a GPU.

References

[1] Raphaël C. Assier and Alice B. Thompson. “MATH20401 Partial Differential Equations and Vector Calculus A Lecture Notes”. University Lecture Course. 2020.

[2] Abraham Berman and Robert J. Plemmons. *Nonnegative Matrices in the Mathematical Sciences.* Society for Industrial and Applied Mathematics, 1994.

[3] Gautam Biswas and Somenath Mukherjee. *Computational Fluid Dynamics.* Alpha Science International, 2014.

[4] Matthias Heil. “MATH35001 Viscous Fluid Flow Lecture Notes”. University Lecture Course. 2021.

[5] E. J. Hinch. *Think Before You Compute.* Cambridge University Press, 2020.

[6] Joseph H. Spurk and Nuri Aksel. *Fluid mechanics.* 3rd ed. Springer Nature, 2020.

[7] Endre Suli and David F. Mayers. *An introduction to numerical analysis.* Cambridge University Press, 2003.

[8] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.