Solving The Navier-Stokes Equations In Two and Three Dimensions

Sam-Henry Whyman
H. H. Wills Physics Laboratory, University of Bristol, Tyndall Avenue, Bristol, BS8 1TL, U.K.
(Dated: May 16, 2015)

We have developed software (written in the C programming language) for the purpose of the computing a numerical solution to the Navier-Stokes equations in both two, and three dimensions, assuming incompressibile flow and adopting periodic boundary conditions. This is achieved by deriving a form of the Poisson equation in pressure, which is then solved using the successive-overrelaxation algorithm and thereafter used to update the velocity field. We design dimension-specific initial velocity fields whose time evolution is presumed to be predictable, for the purpose of verifying the programme. Having originally written the software for serial execution, we extend its functionality by implementing a parallel version using the Message Passing Interface (MPI), which allows compatibility with a high performance cluster.

INTRODUCTION

The Navier-Stokes equations are a key feature in fluid dynamics, and are a mathematical reflection of conservation of matter and Newtonian mechanics in the transportation of fluids. Despite having been developed over one hundred and fifty years prior to the present day, there remains much enigma in the Navier-Stokes equations; it is still not know whether in three dimensions a non-singular solution always exists. This uncertainty is sufficient for a one million dollar prize to be offered by the Clay Mathematics Institute, for a solution, or even a counter example [1]. As well as huge interest in the purely mathematical aspects, the Navier-Stokes equations also have diverse physical significance. Their occurrence in the fields of engineering, meteorology and aviation (for example) is especially important.

With the advent of the use of computers in scientific research, there came major advances in computational fluid dynamics, such as the work of Chorin [2]. This pioneering work employs the so-called projection method, whereby the time and spatial derivatives of the velocity and pressure are calculated, and then decomposed into the sum of two vector fields: one with zero divergence, and one with zero curl. These are then used to calculate the velocity and pressure later in time. With the development of increasingly powerful computers, fast and efficient numerical simulations of the Navier-Stokes equations have become possible, and many more techniques (such as the finite element and finite volume methods, which are generic techniques for solving partial differential equations) have been readily employed.

It is, in fact, the importance of fluid dynamics in meteorology and atmospheric physics which is the main motivation behind this work, in particular, convective weather phenomena. Needless to say, the dynamics of a thunderstorm are incredibly complex. Thermodynamics and turbulence must both be accounted for, as well as the coexistence of air, water vapour, ice and liquid water. The release of latent heat from the condensation of gaseous water to the liquid phase (and then to ice) is important in generating updrafts which help to fuel the growth of a thunderstorm. To a currently unknown extent, even atmospheric impurities are thought to play a role in the nucleation of raindrops. The incorporation of thermodynamics or turbulence into a numerical fluid simulation is beyond our current scope. However, we intend to set the foundation for potential further work in simulating convective weather: We simulate the non-turbulent flow of an incompressible, single-component fluid in a three dimensional domain with periodic boundary conditions.

The Navier-Stokes equations adopt the assumption that the fluid is a continuum, rather than divided into discrete particles. However, in order to obtain numerical solution, we must make a discretisation approximation, whereby the spatial dimensions are divided into a grid, and the velocity field exists for discrete moments in time. We will derive a form for a discrete Poisson equation which must be solved to find the pressure at the next time step. The velocity field is thereafter updated using this already updated pressure field. This is based closely upon the work of Fritsche [3]. We devise this scheme firstly for a two dimensional system, before expanding to three dimensions. After this, the code is parallelised using the Message Passing Interface (MPI).

Mathematical Background

For incompressible fluid flow with constant dynamic viscosity μ, the Navier-Stokes equations read:

$$\begin{aligned} \rho\left(\partial_t \boldsymbol{v} + \boldsymbol{v}\cdot\nabla\boldsymbol{v}\right) &= -\nabla p + \mu\nabla^2\boldsymbol{v} + \boldsymbol{f} \\ \nabla\cdot\boldsymbol{v} &= 0 \end{aligned} \tag{1}$$

where $\boldsymbol{v}$ is the fluid velocity, p is the pressure, ρ is the density of the fluid, and $\boldsymbol{f}$ is a vector representing body forces. We may rewrite the first equation in (1) as

$$\frac{D}{Dt}\boldsymbol{v} = -\nabla p + \frac{1}{Re}\nabla^2\boldsymbol{v} + \boldsymbol{g}, \tag{2}$$

where Re is the Reynolds number (defined as in [4]), and $\boldsymbol{g}$ is a vector representing body forces and is defined as

$$\boldsymbol{g} := \frac{1}{F_r^2\rho_\infty}\hat{\boldsymbol{f}}. \tag{3}$$

The dimensionless quantity F_r is defined as $F_r = v_\infty/\sqrt{l_\infty|\boldsymbol{f}|}$ and is known as the *Froude* number. The quantities v_∞, l_∞ and ρ_∞ are the reference velocity, length and density, respectively. The operator D/Dt in (2) is known as the *convective* derivative and is defined as

$$\frac{D}{Dt}\psi(\boldsymbol{r},t) := \partial_t\psi(\boldsymbol{r},t) + \left(\frac{d}{dt}\boldsymbol{r}\cdot\nabla\right)\psi(\boldsymbol{r},t), \tag{4}$$

where $\psi(\boldsymbol{r},t)$ is some arbitrary function of position and time. Using the definition of ∇ and ∇^2, we arrive at two separate equations for the x and y components of (1). Noting the second equation in (1), we add $\nabla\cdot\boldsymbol{v}$ to each of the aforementioned equations, which is merely the addition of zero. Making use of the chain rule, we arrive at three equations for a two dimensional system:

$$\begin{aligned}
\partial_t u + \partial_x u^2 + \partial_y(uv) - \frac{1}{Re}\left(\partial_x^2 u + \partial_y^2 u\right) + \partial_x p &= g_x \\
\partial_t v + \partial_y v^2 + \partial_x(uv) - \frac{1}{Re}\left(\partial_x^2 v + \partial_y^2 v\right) + \partial_y p &= g_y \\
\partial_x u + \partial_y v &= 0,
\end{aligned} \tag{5}$$

where u and v are the x and y components of the velocity field, respectively, and g_x and g_y are the x and y components of $\boldsymbol{g}$. We now develop an algorithm to solve (5) computationally.

NUMERICAL SOLUTION

There exist many different methods for solving (1) known to the field of Computational Fluid Dynamics (CFD), such as the Finite Volume Method [5], for example. Here, we use finite differences: all derivatives are approximated by the first non-negligible terms of the Taylor expansion of some function, say $\psi(\boldsymbol{r},t)$ in (4). With this in mind, we explain how an algorithm for calculating the three unknowns u, v and p is written.

Discretisation in Time

Our first task is to discretise the equations in (5), so that u, v and p are defined on a discrete time domain $t = nh_t, n \in \mathbb{Z}^+$ and h_t is a small time step. At time step n, we adopt the notation $u^{(n)}, v^{(n)}, p^{(n)}$ for the now discrete variables. We make the following definitions:

$$\begin{aligned}
F^{(n)} &:= u^{(n)} + h_t\left(\frac{1}{Re}\left(\partial_x^2 u^{(n)} + \partial_y^2 u^{(n)}\right) - \partial_x\left(\left(u^{(n)}\right)^2\right) - \partial_y\left(u^{(n)}v^{(n)}\right) + g_x\right) \\
G^{(n)} &:= v^{(n)} + h_t\left(\frac{1}{Re}\left(\partial_x^2 v^{(n)} + \partial_y^2 v^{(n)}\right) - \partial_y\left(\left(v^{(n)}\right)^2\right) - \partial_x\left(u^{(n)}v^{(n)}\right) + g_y\right).
\end{aligned} \tag{6}$$

Writing the time derivatives as forward finite differences:

$$(\partial_t u)^{(n)} = \frac{u^{(n+1)} - u^{(n)}}{h_t} \quad \text{and} \quad (\partial_t v)^{(n)} = \frac{v^{(n+1)} - v^{(n)}}{h_t}, \tag{7}$$

and substituting into (5), we arrive at

$$u^{(n+1)} = F^{(n)} - h_t \partial_x p^{(n+1)} + \mathcal{O}(h_t^2) \quad \text{and} \quad v^{(n+1)} = G^{(n)} - h_t \partial_y p^{(n+1)} + \mathcal{O}(h_t^2), \tag{8}$$

where we have used the fact that $\partial_x p^{(n)} = \partial_x p^{(n+1)} + \mathcal{O}(h_t)$ and $\partial_y p^{(n)} = \partial_y p^{(n+1)} + \mathcal{O}(h_t)$, which arises due to Taylor's theorem.

Discretisation in Space

In order to increase the stability of the solution, three staggered grids are used each for u, v and p. That is to say, the grids for u and v are parallel shifted *left* and *up* (respectively) with respect to the p grid, such that the point $p_{i,j}$ lies in the centre of a *cell*, bounded by $u_{i-1/2,j}$, $u_{i+1/2,j}$, $v_{i,j-1/2}$ and $v_{i,j+1/2}$. This is shown in FIG. 1. We create two new arrays, U and V which are used directly in the calculation. These are related to the orginal arrays u and v as follows:

$$\begin{aligned} u_{i,j} &= \frac{u_{i-1/2,j} + u_{i+1/2,j}}{2} = \frac{U_{i-1,j} + U_{i,j}}{2} \\ v_{i,j} &= \frac{v_{i-1/2,j} + v_{i+1/2,j}}{2} = \frac{V_{i-1,j} + V_{i,j}}{2}. \end{aligned} \tag{9}$$

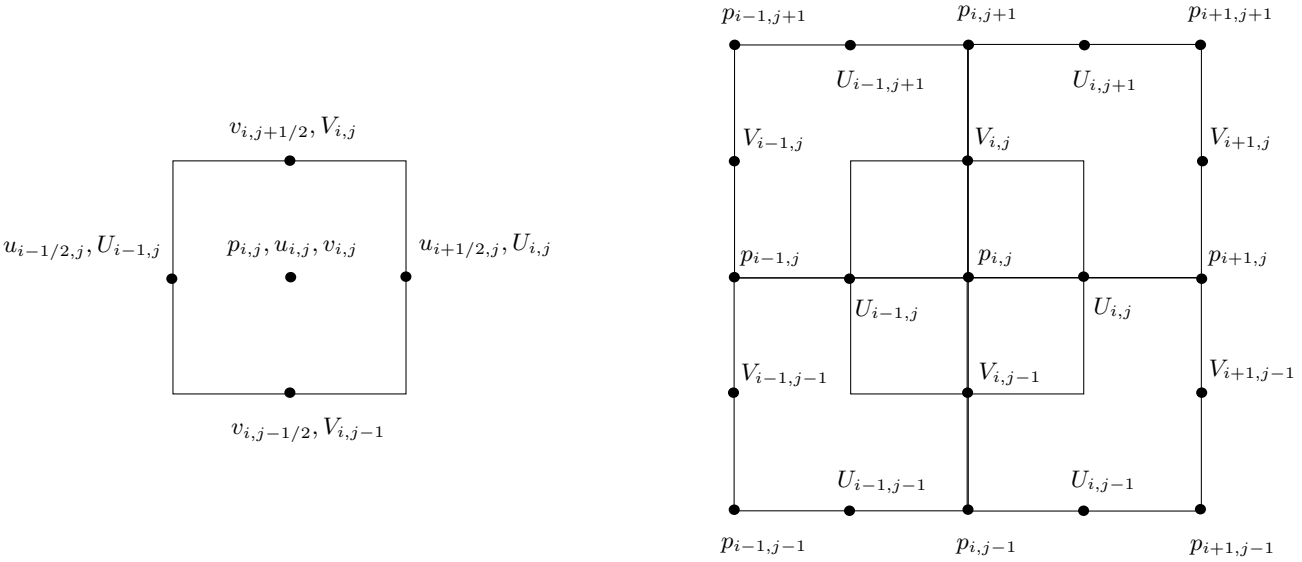


FIG. 1: Left: A section of the domain showing $p_{i,j}$, $u_{i,j}$ and $v_{i,j}$ all positioned at the centre of the solid black square. Right: A larger section of the space discretised domain. The pressure $p_{i,j}$ lies at the centre of the cell (as depicted in the left hand figure), and $U_{i,j}$ and $V_{i,j}$ are positioned so as to conform with the definitions in (9).

The averaging in (9) would normally mean that is is necessary to supply an extra 'boundary layer' of cells around the spatial domain. However, the use of periodic boundary conditions makes this a redundant feature, and so the size of the grid can be

defined simply as $N_i \times N_j$. This is explained further in the section titled **boundary conditions**. We are able to define all convective derivatives needed using central differences, for example,

$$[\partial_y (UV)]_{i,j} = \frac{1}{h_y}\left[\frac{U_{i,j+1}+U_{i,j}}{2}\frac{V_{i,j}+V_{i+1,j}}{2} - \frac{U_{i,j-1}+U_{i,j}}{2}\frac{V_{i,j-1}+V_{i+1,j-1}}{2}\right]. \tag{10}$$

All other convective derivatives can be derived easily [3, p. 21–23]. Using the definitions of U and V in (9), we are now in a position to rewrite (8) for a discrete grid:

$$\begin{aligned} U_{i,j}^{(n+1)} &= F_{i,j}^{(n)} - \frac{h_t}{h_x}\left(p_{i+1,j}^{(n+1)} - p_{i,j}^{(n+1)}\right) + \mathcal{O}(h_x) + \mathcal{O}(h_y) + \mathcal{O}(h_t^2) \\ V_{i,j}^{(n+1)} &= G_{i,j}^{(n)} - \frac{h_t}{h_y}\left(p_{i,j+1}^{(n+1)} - p_{i,j}^{(n+1)}\right) + \mathcal{O}(h_x) + \mathcal{O}(h_y) + \mathcal{O}(h_t^2), \end{aligned} \tag{11}$$

where $F_{i,j}$ and $G_{i,j}$ are defined in the same way as in (6), except that they are discretised on a grid.

The Poisson Equation

Substituting (8) into the third equation of (5), one obtains:

$$0 = \partial_x u^{(n+1)} + \partial_y v^{(n+1)} = \partial_x F^{(n)} - h_t \partial_x^2 p^{(n+1)} + \partial_y G^{(n)} - h_t \partial_y^2 p^{(n+1)} + \mathcal{O}(h_t^2). \tag{12}$$

Immediately, one recognises (12) to be a Poisson equation in pressure:

$$\nabla^2 p^{(n+1)} = \frac{1}{h_t}\left(\partial_x F^{(n)} + \partial_y G^{(n)}\right) + \mathcal{O}(h_t). \tag{13}$$

It is straightforward to write (13) in discrete form, again using finite differences:

$$\frac{p_{i+1,j}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i-1,j}^{(n+1)}}{h_x^2} + \frac{p_{i,j+1}^{(n+1)} - 2p_{i,j}^{(n+1)} + p_{i,j-1}^{(n+1)}}{h_y^2} \approx \frac{1}{h_t}\left(\frac{F_{i,j}^{(n)} - F_{i-1,j}^{(n)}}{h_x} + \frac{G_{i,j}^{(n)} - G_{i,j-1}^{(n)}}{h_h}\right), \tag{14}$$

where we have omitted the terms $\mathcal{O}(h_x)$, $\mathcal{O}(h_y)$ and $\mathcal{O}(h_t^2)$. We define

$$R_{i,j}^{(n)} := \frac{1}{h_t}\left(\frac{F_{i,j}^{(n)} - F_{i-1,j}^{(n)}}{h_x} + \frac{G_{i,j}^{(n)} - G_{i,j-1}^{(n)}}{h_y}\right). \tag{15}$$

Writing (14) as a matrix equation $Ap = R$, where A is a tri-diagonal matrix, it becomes clear that essentially we need to solve a system of linear equations with $N_i \times N_j$ unknowns, in a *fully implicit* regime [6, p. 1045]. To do so by methods such as LU decomposition would prove to be highly inefficient due to the size of the system. For this reason, we experiment with the technique of *successive-overrelaxation* (SOR) [6, p. 1061–1066], the algorithm for which, adapted for the specific problem, is summarised in algorithm 1.

The integer k is the current iteration, and $k_{\max}$ is the number of desired iterations. The superscripts $p_{i,j}^{[k]}$ now refer to the current *iteration* of the pressure field, and not the discretisation in time, as was represented by $p_{i,j}^{(n)}$. The quantity ω is known as the *relaxation parameter*, and $\omega \in [0, 2]$. $\omega = 1$ allows the SOR method to tend to the *Gauss-Siedel* approach. Smaller values lead to slower convergence and better stability, whereas larger values invoke faster convergence in general, but may sometimes lead to divergence. Note that we have not yet mentioned how large $k_{\max}$ ought to be: a stricter criterion for convergence is described in subsequent sections.
Once the pressure field has been obtained, it is substituted into (11), thus updating the velocity fields U and V. A summary of the full algorithm used to solve the Navier-Stokes equations in two dimensions is summarised in algorithm 2.

Algorithm 1 Sucessive Overrelaxation

function SOR
  for $k = 0, \ldots, k_{\max}$ **do**
    for $i = 0, \ldots, N_i$ **do**
      for $j = 0, \ldots, N_j$ **do**
        $$p_{i,j}^{[k+1]} = (1-\omega)p_{i,j}^{[k]} + \frac{\omega}{2/h_x^2+2/h_y^2}\left(\frac{p_{i+1,j}^{[k]}+p_{i-1,j}^{[k+1]}}{h_x^2} + \frac{p_{i,j+1}^{[k]}+p_{i,j-1}^{[k+1]}}{h_y^2} - R_{i,j}\right)$$
      end for
    end for
  end for
end function

Algorithm 2 The complete algorithm

function MAIN
  Initialise $t = 0$ and set initial conditions for $U^{(0)}$, $V^{(0)}$ and $p^{(0)}$
  while $t < 0$ **do**
    Set $F^{(n)}$, $G^{(n)}$ ▷ See equations (6)
    Set $R^{(n)}$ ▷ See equation (15)
    Set $k = 0$
    Call SOR on p and R
    Update $U^{(n+1)}$ and $V^{(n+1)}$ ▷ See equations (11)
    $t = t + h_t$
  end while
end function

BOUNDARY CONDITIONS

It has already been mentioned that for the sake of simplicity, periodic boundary conditions have been implemented. This enables the full domain to be used, and does not necessitate a boundary layer of cells. These boundary conditions are achieved using the following code (or as much of it as is required) whenever a quantity (such as R, for example) is calculated.

Algorithm 3 Period boundary conditions

```
for(i = 0; i < N_i; i++){
        for(j = 0; j < N_j; j++){
                int i_minus = i - 1;
                int j_minus = j - 1;
                int i_plus = i + 1;
                int j_plus = j + 1;

                if(i == 0){
                        i_minus = N_i - 1;
                }
                if(j == 0){
                        j_minus = N_j - 1;
                }
                if(i == N_i - 1){
                        i_plus = 0;
                }
                if(j == N_j - 1){
                        j_plus = 0;
                }
                Calculate_Quantity();
        }
}
```

TESTING AN ANALYTIC SOLUTION

It is often the case that solving the Navier-Stokes equations analytically is either challenging, impossible or impractical in physical situations. There are, of course, simpler situations where the solutions can be solved exactly. Let us start with the initial velocity field

$$\boldsymbol{v}^{(0)} = u(x,y)\hat{\boldsymbol{x}} + v(x,y)\hat{\boldsymbol{y}}. \tag{16}$$

The *vorticity* $\boldsymbol{\omega}$ of the flow field is defined as the curl of the velocity field: $\boldsymbol{\omega} = \nabla \times \boldsymbol{v} = (\partial_x v(x,y,t) - \partial_y u(x,y,t))\,\hat{\boldsymbol{z}}$. If we arrange so that the initial vorticity is a two dimensional Gaussian function of the form

$$\boldsymbol{\omega}^{(0)} = A \exp\left(-\frac{x^2}{a} - \frac{y^2}{b}\right)\hat{\boldsymbol{z}}, \tag{17}$$

where $a, b, A \in \mathbb{R}$, then solving the Navier-Stokes equations with this initial condition yields a curl which retains its Gaussian shape, with a 'width' which increases $\propto t^{1/2}$ [7, Ch. 9]. It is important to note that this *diffusion of a vortex sheet* is only a model setup, and real systems will rarely exhibit this exact behaviour. We might be tempted to set (16) so that

$$\begin{aligned} u(x,y) &= C \\ v(x,y) &= \frac{\sqrt{a\pi}A}{2} \exp\left(-\frac{y^2}{b}\right) \cdot \operatorname{erf}\left(\frac{x}{\sqrt{a}}\right), \end{aligned} \tag{18}$$

where $C \in \mathbb{R}$ is a constant, so that (17) is satisfied. A visualisation of this initial velocity field is presented in FIG. 2.

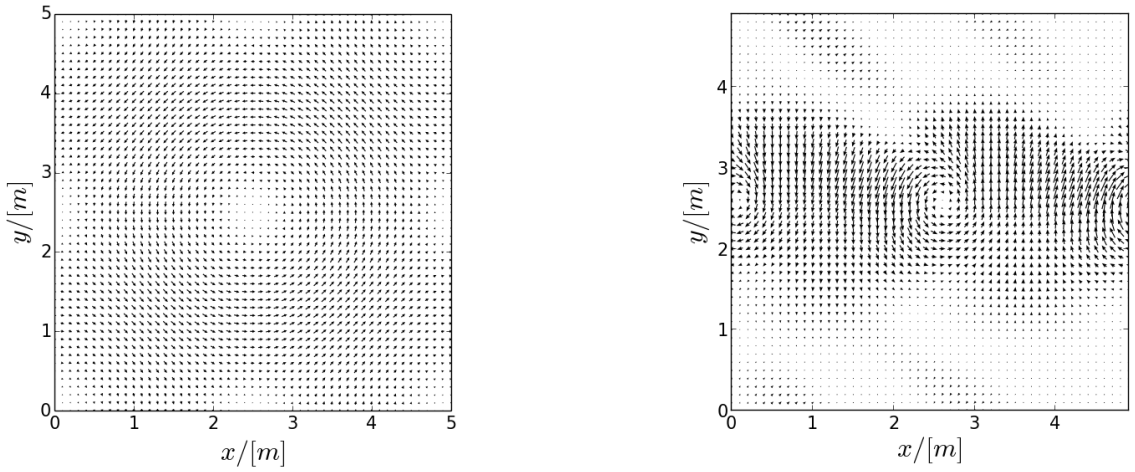


FIG. 2: Left: The initial velocity field described by (16) and (18). Right: The velocity field after 2 seconds. The boundary conditions allow undesired secondary vortices to form at the edge of the region, as well as 'self-interference' from the top and bottom edges. The grid size is 51 × 51. The following other parameters were used: $dx = dy = 0.1$ m, and $dt = 0.01$ s.

There are two major problems with using (16) as the initial velocity field. This first arises due to the use of periodic boundary conditions. As a result of the error function in (18), there is a discontinuity at the left and right hand boundaries of the grid. Due to periodicity, there occurs undesired circulation at the edges of the domain. This then affects the 'central circulation', distorting the expected Gaussian shape of the vorticity, as the system is evolved with time. An example of this can be noticed in FIG. 2.

The second problem is somewhat more serious: (16) is not divergence free. This is likely to cause numerical instability in the simulation, which is certainly not desirable, and will lead to inaccurate results.

A Divergentless Initial Field

We will make use of the Gaussian functions

$$g_{l,m}(x,y) = A\exp\left(-\frac{(x-l)^2}{a} - \frac{(y-m)^2}{b}\right), \tag{19}$$

where A, a and b are constants as in (17), and l, m are the coordinates about which the Gaussian is centred. We construct an initial velocity field which is proportional to the gradient of (19):

$$\begin{aligned} \nabla g_{l,m}(x,y) &= \partial_x g_{l,m}(x,y)\hat{\boldsymbol{x}} + \partial_y g_{l,m}(x,y)\hat{\boldsymbol{y}} \\ \Rightarrow \nabla g_{l,m}(x,y) &= -2A\exp\left(-\frac{(x-l)^2}{a} - \frac{(y-m)^2}{b}\right)\left(\frac{(x-l)}{a}\hat{\boldsymbol{x}} + \frac{(y-m)}{b}\hat{\boldsymbol{y}}\right). \end{aligned} \tag{20}$$

Alone, this is not adequate, since (20) is not divergence free. For a divergence free field, it is possible to subject (20) to a clockwise rotation of $\pi/2$, thereby ensuring that the field is everywhere normal to the contours of (19). Applying the rotation matrix

$$R = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}, \tag{21}$$

we define

$$\boldsymbol{v}^{(0)}_{l,m} := R\left(\nabla g_{l,m}(x,y)\right) = -2A\exp\left(-\frac{(x-l)^2}{a} - \frac{(y-m)^2}{b}\right)\left(\frac{(y-m)}{b}\hat{\boldsymbol{x}} - \frac{(x-l)}{a}\hat{\boldsymbol{y}}\right). \tag{22}$$

It is now possible to take advantage of the periodic boundary conditions. Before doing so, let us suppose that our region of interest is part of a larger, 'self-repeating' grid, as shown in FIG. 3. The grid is assembled so that each 'node' is separated by N_i and N_j in the x and y directions, respectively. At each node (l,m), a field of the form in (22) is centred.

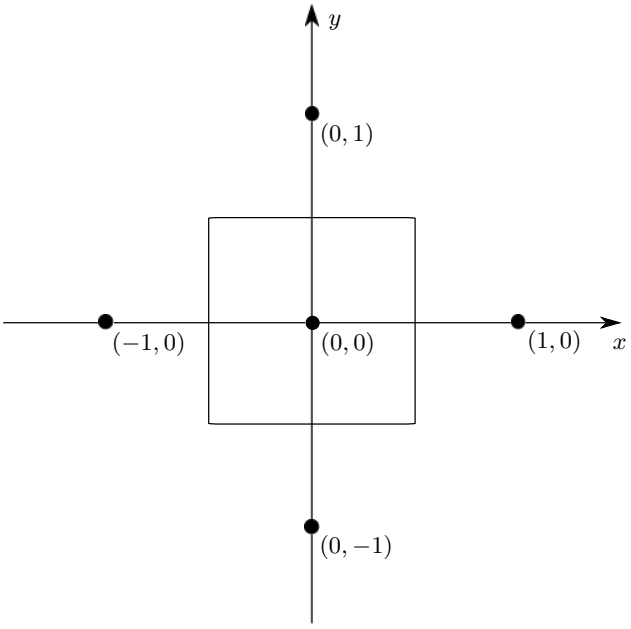


FIG. 3: The region of interest (solid black square) is surrounded by a lattice of identical regions. Each black circle represents a 'node' at (l,m), where $N_i = N_j$ and the lengths have been normalised. There are a maximum of $l_{\max}$ and $m_{\max}$ nodes in the positive and negative x and y directions, respectively. In principle, $l_{\max}, m_{\max} \to \infty$.

The total initial velocity field is then given by

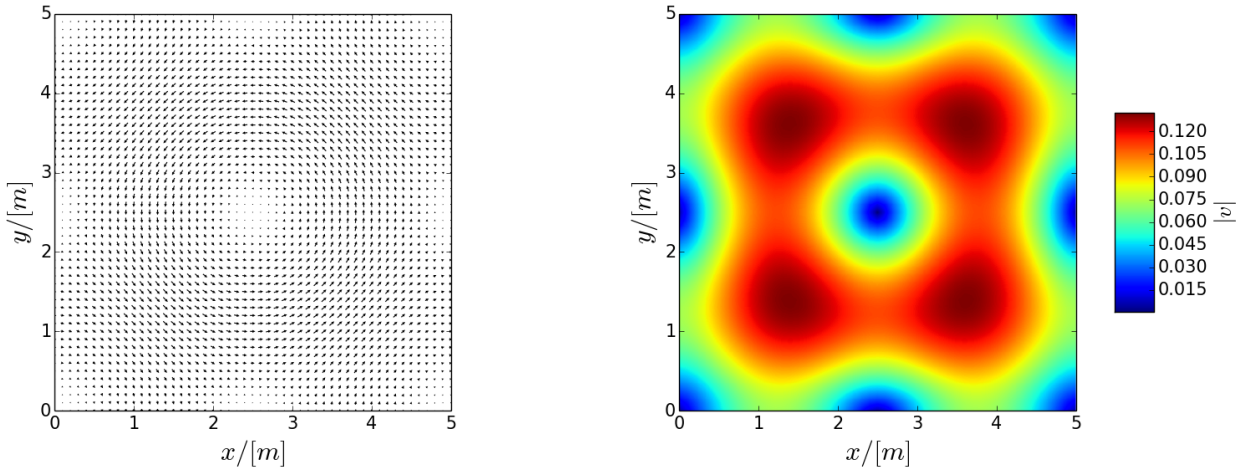


FIG. 4: The initial velocity field (23) in the unit cell. The left hand side shows a normalised vector plot, and the right hand side shows the magnitude of the field. The nine points of zero velocity are readily noticeable in the latter. The following parameters were used: $l_{\max} = m_{\max} = 10$, $A = 0.5$, $a = b = 6$, with all other parameters the same as in FIG. 2. In this plot, $l_{\max}$ and $m_{\max}$ are excessively large, given the quick exponential decay of the velocity field.

$$\boldsymbol{v}_{\text{tot}}^{(0)} = \sum_{l=-l_{\max}}^{l_{\max}} \sum_{m=-m_{\max}}^{m_{\max}} \boldsymbol{v}_{l,m}^{(0)}. \tag{23}$$

Assuming that the width of each Gaussian field $\boldsymbol{v}_{l,m}^{(0)}$ is comparable with the size of the primary cell, there will be nine points within the cell where interference causes zero velocity. These points lie at each corner, the midpoints of each edge, and at the origin of the cell's node. Furthermore, every cell will be identical: that is to say, the primary cell is a *unit cell*, with the entire pattern reproducible from translation of the unit cell by the set of lattice vectors $\boldsymbol{G} = lN_i\hat{\boldsymbol{x}} + mN_j\hat{\boldsymbol{y}}$. Such pattern is presented in FIG. 4 for the unit cell.
The implementation of periodic boundary conditions on the unit cell produces the same result, provided that $l_{\max}$ and $m_{\max}$ are sufficiently large. In this case, the velocity field is determined using (23).

Evolution With Time

Given an initial velocity field (23) over all 2D space, it is possible to draw an important conclusion to the expected time evolution of the field. That is, the *zeros* (FIG. 4) should stay fixed in space, and continue to have zero velocity. Meanwhile, the contours become more flat, albeit retaining normalised shape. The former effect was deemed to be appropriate to test the reliability of the numerical solution.

RESULTS

The simulation was run for various lengths of time, with the same parameters as described in FIG. 4. Contour plots of the magnitude of the velocity field are presented in FIG. 5. Two significant observations can be made, which conform with expectations:

i) there remain nine points where the magnitude of the velocity is zero, and

ii) the maximum speed (red regions in the magnitude contour plots) decays considerably with time.

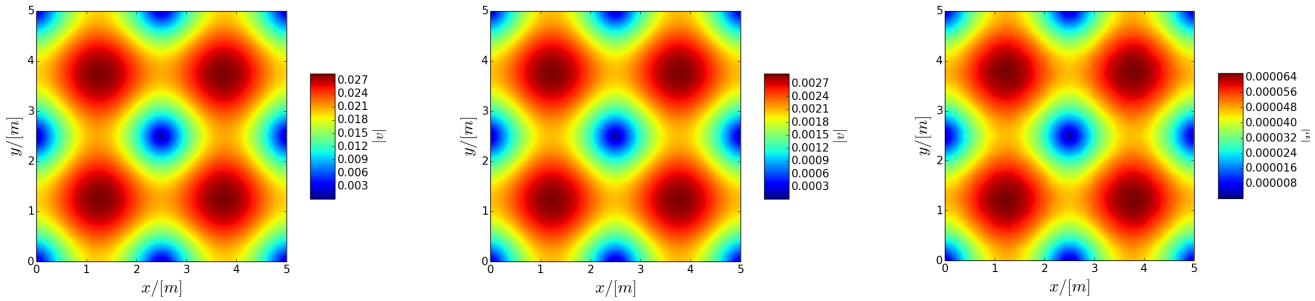


FIG. 5: From left to right: The magnitude of the velocity field after 100, 250 and 500 seconds, with initial conditions as in (23) and FIG. 4. The Reynolds number was chosen to be 100, since it was postulated that this would provide as resonably short decay time.

EXTENSION INTO 3D

It was possible to obtain (5) from (2) by making use of the fact that we have taken the divergence of the velocity field to be zero. Imposing the same conditions for a three dimensional velocity field $\boldsymbol{v} = u\hat{\boldsymbol{x}} + v\hat{\boldsymbol{y}} + w\hat{\boldsymbol{z}}$ we arrive at

$$\begin{aligned}
\partial_t u + \partial_x u^2 + \partial_y (uv) + \partial_z (uw) - \frac{1}{Re}\left(\partial_x^2 u + \partial_y^2 u + \partial_z^2 u\right) + \partial_x p &= g_x \\
\partial_t v + \partial_y v^2 + \partial_x (uv) + \partial_z (vw) - \frac{1}{Re}\left(\partial_x^2 v + \partial_y^2 v + \partial_z^2 v\right) + \partial_y p &= g_y \\
\partial_t w + \partial_z w^2 + \partial_x (uw) + \partial_y (vw) - \frac{1}{Re}\left(\partial_x^2 w + \partial_y^2 w + \partial_z^2 w\right) + \partial_z p &= g_z \\
\partial_x u + \partial_y v + \partial_z w &= 0.
\end{aligned} \tag{24}$$

Having discretised the above equations in (24) at time step n, we redefine F and G in (6), and introduce a new variable H so that

$$\begin{aligned}
F^{(n)} &:= u^{(n)} + h_t\left(\frac{1}{Re}\left(\partial_x^2 u^{(n)} + \partial_y^2 u^{(n)} + \partial_z^2 u^{(n)}\right) - \partial_x\left(\left(u^{(n)}\right)^2\right) - \partial_y\left(u^{(n)}v^{(n)}\right) - \partial_z\left(u^{(n)}w^{(n)}\right) + g_x\right) \\
G^{(n)} &:= v^{(n)} + h_t\left(\frac{1}{Re}\left(\partial_x^2 v^{(n)} + \partial_y^2 v^{(n)} + \partial_z^2 v^{(n)}\right) - \partial_y\left(\left(v^{(n)}\right)^2\right) - \partial_x\left(u^{(n)}v^{(n)}\right) - \partial_z\left(v^{(n)}w^{(n)}\right) + g_y\right) \\
H^{(n)} &:= w^{(n)} + h_t\left(\frac{1}{Re}\left(\partial_x^2 w^{(n)} + \partial_y^2 w^{(n)} + \partial_z^2 w^{(n)}\right) - \partial_z\left(\left(w^{(n)}\right)^2\right) - \partial_x\left(u^{(n)}w^{(n)}\right) - \partial_y\left(v^{(n)}w^{(n)}\right) + g_z\right).
\end{aligned} \tag{25}$$

Once again, we write the time derivatives as forward finite differences:

$$(\partial_t u)^{(n)} = \frac{u^{(n+1)} - u^{(n)}}{h_t}, \quad (\partial_t v)^{(n)} = \frac{v^{(n+1)} - v^{(n)}}{h_t}, \quad \text{and} \quad (\partial_t w)^{(n)} = \frac{w^{(n+1)} - w^{(n)}}{h_t}, \tag{26}$$

and substitute into (24) to obtain

$$\begin{aligned}
u^{(n+1)} &= F^{(n)} - h_t \partial_x p^{(n+1)} + \mathcal{O}(h_t^2), \\
v^{(n+1)} &= G^{(n)} - h_t \partial_y p^{(n+1)} + \mathcal{O}(h_t^2), \\
w^{(n+1)} &= H^{(n)} - h_t \partial_z p^{(n+1)} + \mathcal{O}(h_t^2).
\end{aligned} \tag{27}$$

The use of a staggered grid is also applicable in three dimensions. Following on from (9), we can write

$$u_{i,j,k} = \frac{U_{i-1,j,k} + U_{i,j,k}}{2}, \quad v_{i,j,k} = \frac{V_{i,j-1,k} + V_{i,j,k}}{2} \quad \text{and} \quad w_{i,j,k} = \frac{U_{i,j,k-1} + U_{i,j,k}}{2}. \tag{28}$$

The central difference method can be used to define (25), just as it was used in the two dimensional case. Finally, we are able to rewrite the Poisson equation (13) as

$$\frac{p^{(n+1)}_{i+1,j,k} - 2p^{(n+1)}_{i,j,k} + p^{(n+1)}_{i-1,j,k}}{h_x^2} + \frac{p^{(n+1)}_{i,j+1,k} - 2p^{(n+1)}_{i,j,k} + p^{(n+1)}_{i,j-1,k}}{h_y^2} + \frac{p^{(n+1)}_{i,j,k+1} - 2p^{(n+1)}_{i,j,k} + p^{(n+1)}_{i,j,k-1}}{h_x^2} \approx \frac{1}{h_t} R^{(n)}_{i,j,k}, \tag{29}$$

where $R^{(n)}_{i,j,k}$ has been redefined as

$$R^{(n)}_{i,j,k} := \frac{1}{h_t}\left(\frac{F^{(n)}_{i,j,k} - F^{(n)}_{i-1,j,k}}{h_x} + \frac{G^{(n)}_{i,j,k} - G^{(n)}_{i,j-1,k}}{h_y} + \frac{H^{(n)}_{i,j,k} - H^{(n)}_{i,j,k-1}}{h_z}\right). \tag{30}$$

Using these redefined terms, it is reasonable to conclude that the iterative step in the *successive-overrelaxation* algorithm can aslo be rewritten to account for the addition of an extra dimension:

$$p^{[m+1]}_{i,j,k} = (1-\omega)p^{[m]}_{i,j,k} + \frac{\omega}{\frac{2}{h_x^2} + \frac{2}{h_y^2} + \frac{2}{h_z^2}}\left(\frac{p^{[m]}_{i+1,j,k} + p^{[m+1]}_{i-1,j,k}}{h_x^2} + \frac{p^{[m]}_{i,j+1,k} + p^{[m+1]}_{i,j-1,k}}{h_y^2} + \frac{p^{[m]}_{i,j,k+1} + p^{[m+1]}_{i,j,k-1}}{h_z^2} - R_{i,j,k}\right), \tag{31}$$

where we have replaced the superscript index k with m to avoid confusion with the grid indices i, j, k. We also note that we must perform an extra loop over all k values. This will considerably increase the required computation time, and thus the algorithm will necessitate parallelisation. This is subsequently described before presenting results of the evolution of a designed initial field.

PARALLELISATION OF CODE

Now that the Navier Stokes equations are ready to be solved in three dimensions, it is important to parallelise the code. We start by the parallelisation of the solution to the Poisson equation in two dimensions only, before including the initialisation of the F, G and R fields. Once both of these have been accomplished, attention will be diverted to the extension into three dimensions. Our chosen method of concurrent computation is to use the Message Passing Interface (MPI) [8], since this is widely compatible with the cluster on which the programme will eventually be executed. We have adapted code written by the Florida State University [9], which is distributed under the GNU Lesser General Public License.

In order to parallelise the code, the domain is divided into 'strips'. Each processor is then assigned a section of grid on which it is to operate. However, there are two changes that need to be made:

1. The SOR algorithm requires knowledge of the contents of the neighbours of each node. Therefore, each process must be able to access not only its assigned nodes, but a single layer of ghost cells containing information about neighbouring nodes in the original domain. Since all processes are concurrent, the only way to achieve this is to let each process know which process neighbours it. The current process sends information about its edge nodes, which are received by the correct process and saved into that process' ghost cells. The real domain must also have a single layer of ghost cells to accommodate this, so that the true grid is offset by one node in each direction.

2. Before now, we used 'pointers to pointers to doubles' to represent the required arrays. However, this approach does not ensure a contiguous layout in memory, which is needed for ease of MPI implementation. In order to enforce this criterion, we refine each two dimensional array as a one dimensional array (single pointers to doubles), and define the macro: $\mathrm{INDEX(i,j)}$ as $(\mathrm{N\_i} + 2) \times \mathrm{i} + \mathrm{j}$.

The parallel version of the code is outlined in Algorithm 5, which uses the parallel version of the SOR method summarised in Algorithm 4.

Algorithm 4 Parallelised version of the SOR method

function PARASOR
  Update top and bottom ghost layers
  Update left and right ghost layers using non-blocking send and receive
  Apply Algorithm 1 to all internal nodes
end function

Algorithm 5 Parallelised version of the code

function MAIN
  Initialise $t = 0$ and set initial conditions for $U^{(0)}$, $V^{(0)}$ and $p^{(0)}$
  Divide the domain in strips of height $(N_j + 2)$
  while $t < 0$ **do**
    Distribute P, P_new, U and V from the root process to all slave processes
    Each Process sets $F^{(n)}$, $G^{(n)}$ and $R^{(n)}$
    while (not converged) **do**
      Call PARASOR( ) from each process, on their subdomain
      Swap P and P_new
    end while
    Send each process' converged section back to main
    Update $U^{(n+1)}$ and $V^{(n+1)}$ from main
    $t = t + h_t$
  end while
end function

Convergence

It was desired that the Poisson equation be solved within a particular degree of error, hence a technique was employed to estimate the error after each iteration of the SOR algorithm. This is outlined in Algorithm 6. The convergence criterion in Algorithm 5 is simply that $\mathrm{change} \leq \epsilon$, where ϵ is the desired fractional accuracy.

It was decided to test the effect of changing the value of ϵ, on the accuracy of the solution. The centre point was chosen as a suitable reference point since the magnitude of the velocity field should remain zero here. FIG. 6 shows the deviation from zero velocity magnitude at the centre of the domain as a function of time, for a variety of values for ϵ. In both the serial and parallel cases, the deviation decreases monotonically as a function of time, although the gradient decreases between $\sim$15 s and $\sim$30 s. For reasonably large ϵ (eg, 10^{-1}), the deviation oscillates whilst conforming to the same averaged curve. With dual processing, this oscillation is larger, but in both cases it diminishes with time. Once ϵ is smaller than $\sim 10^{-3}$, the oscillations are no longer noticeable, hence this is might be deemed as a minimum convergence criterion for the current conditions.

In Three Dimensions

The task of translating the two dimensional version of the parallel code into the three dimensional version, as outlined previously, is relatively straightforward, since many features in the two dimensional case are repeated. The main difference is that

Algorithm 6 Estimating the error in the SOR algorithm

function ESTIMATEERROR
  Set change, n, $\mathrm{my\_change}$ and $\mathrm{my\_n}$ to zero
  for (Each node in this process, excluding ghost cells) **do**
    if $(\mathrm{P\_new} \neq 0)$ **then**
      $\mathrm{my\_change} = \mathrm{my\_change} + |(1 - \mathrm{P_{i,j}})/\mathrm{P\_new_{i,j}}|$
      $\mathrm{my\_n}$ ++
    end if
  end for
  Collect $\mathrm{my\_change}$ and $\mathrm{my\_n}$ as change and n respectively, in each process, using $\mathrm{MPI\_Allreduce}$.
  if $(\mathrm{n} \neq 0)$ **then**
    $\mathrm{change} = \mathrm{change}/\mathrm{n}$
  end if
end function

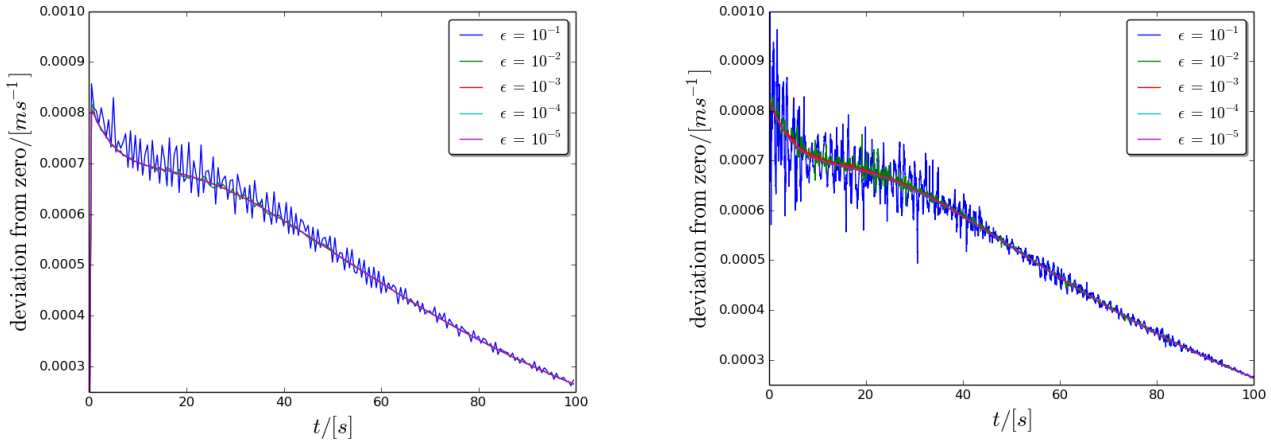


FIG. 6: The deviation of the magnitude of the velocity field from zero at the centre of the domain as a function of time (up to 100 s). The results from running the code in both serial (left) parallel (right) are presented.

the domain is divided into columns rather than rectangles, although the layout of the $x-y$ face of each column is calculated in the exact same manner as before: The columns extend into the whole of the y and z directions of the domain, and the division is done in terms of the x axis. The layer of ghost cells is still enforced, with the difference residing in the fact that *faces* must now be transported into them, rather than lines of cells. For this reason, care must be taken to use the correct face for transportation due to the memory layout. For example, if we use the function INDEX(i, j, k) to return $((N_i+2)i+j)(N_j+2)+k$ as the on dimensional array index, then it is the $y-z$ face that we must use for updating the ghost cells. The other faces ($x-y$ and $x-z$) are connected 'top to bottom' and 'front to back' within each individual process, respectively.

The Initial 3D field

With the three dimensional parallel code prepared, we require a solution for testing. In the two dimensional case, it was sufficient to rotate the gradient of a Gaussian function by $\pi/2$ so that the result was a divergence free field. In three dimension, this rotation is non-trivial, and does not necessarily a guarantee a non-divergent field. One option is to use the identity $\nabla \cdot (\nabla \times \boldsymbol{A}) = 0$, where $\boldsymbol{A}$ is a vector field and the desired field is given by $\boldsymbol{v}^{(0)}_{l,m,n} = \nabla \times \boldsymbol{A}$. Alternately, we may be more daring and suggest a divergentless field from scratch. This option might be preferred since we seek to design a field whose time evolution will retain its 'shape' while decaying in amplitude. An example of such a field is

$$\boldsymbol{v}^{(0)}_{l,m,n} := A \begin{pmatrix} -2\left(\frac{y-m}{b}\right)\left(\frac{z-n}{c}\right)\exp\left(-\frac{(x-l)^2}{a}-\frac{(y-m)^2}{b}-\frac{(z-n)^2}{c}\right) \\ \left(\frac{x-l}{a}\right)\left(\frac{z-n}{c}\right)\exp\left(-\frac{(x-l)^2}{a}-\frac{(y-m)^2}{b}-\frac{(z-n)^2}{c}\right) \\ \left(\frac{y-m}{b}\right)\left(\frac{x-l}{a}\right)\exp\left(-\frac{(x-l)^2}{a}-\frac{(y-m)^2}{b}-\frac{(z-n)^2}{c}\right) \end{pmatrix} \tag{32}$$

where n is another integer. By its design, this field is clearly divergent free. These fields may be summed over an appropriate number of unit cells to give the total initial velocity field:

$$\boldsymbol{v}^{(0)}_{\text{tot}} = \sum_{l=-l_{\max}}^{l_{\max}} \sum_{m=-m_{\max}}^{m_{\max}} \sum_{n=-n_{\max}}^{n_{\max}} \boldsymbol{v}^{(0)}_{l,m,n}. \tag{33}$$

It was deemed that it was sufficient to set $l_{\max} = m_{\max} = n_{\max} = 4$, due to the quick decay of the exponential function. Vector plots of (33) are presented in FIG. 7, and the magitude of the field is represented by the figures in FIG. 8.

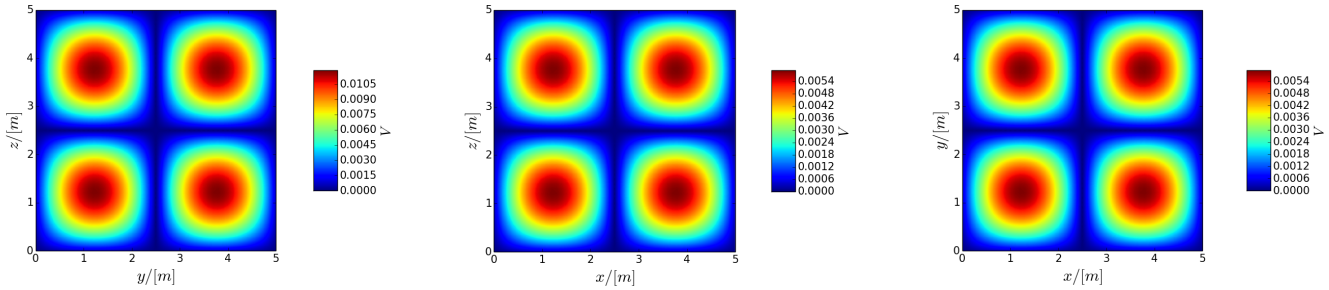
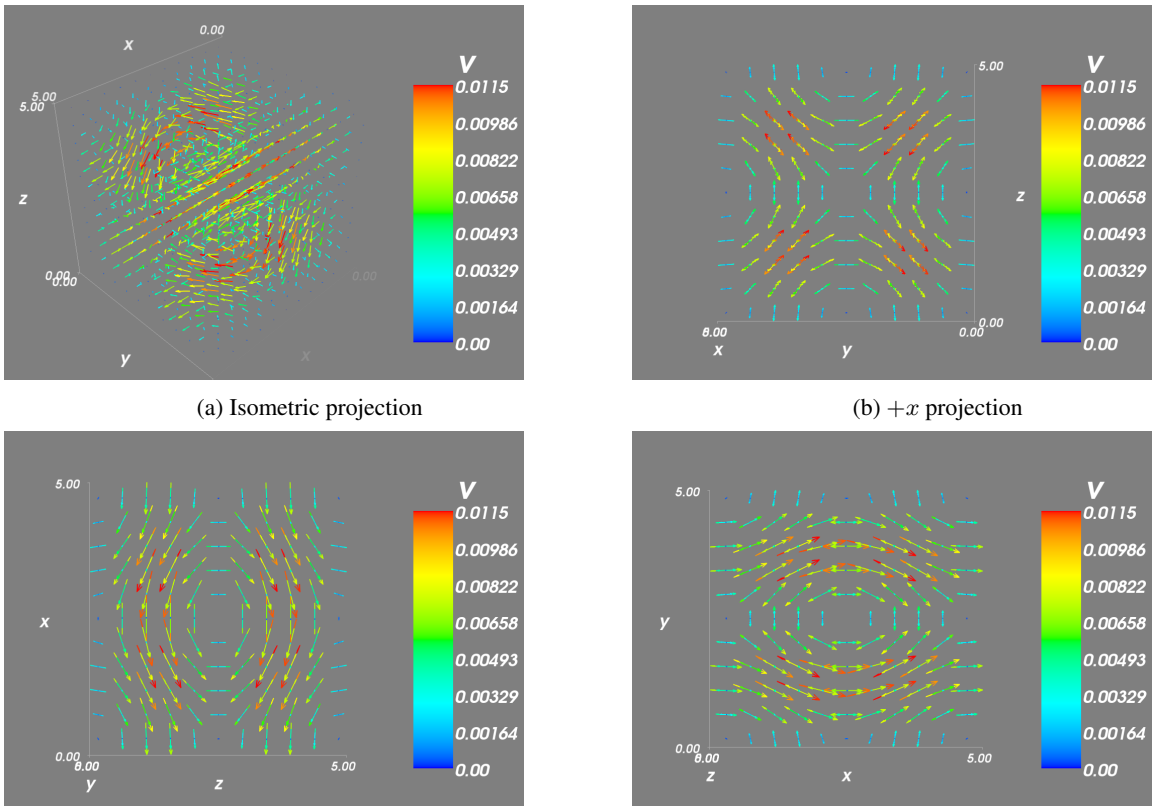


FIG. 8: The magnitude of the initial field projected along the x, y and z directions (left to right), at the midpoint of each axis. The symmetry of the field is evident, although the magnitude of the field in the x projected figure has a maximum of approximately twice that of the other projections.



(a) Isometric projection

(b) $+x$ projection

(c) $+y$ projection

(d) $+z$ projection

FIG. 7: Clockwise from top left: The velocity field projected isometrically, and along the $+x$, $+y$ and $+z$ directions.

Upon time evolution, we postulate that an initial field such as in (33) will retain regions of zero magnitude such as in FIG. 8, while retaining its symmetry. The simulation was run for only 30 s, since at this stage we used a dual core laptop. FIG. 9 demonstrates the time evolution of the magnitude of the field shown FIG. in 7. At this point, it was realised that the initialisation of the velocity field took longer to complete than the subsequent iterative calculations. Henceforth, it was decided to parallelise the initialisation in addition to the allocation of F, G and H, and the solving of the Poisson equation.

Comparing FIG. 9 with FIG. 7 and the first of FIG. 8, we observe that the time evolution appears as expected: The shape of the field remains the same while the amplitude decays. In order to test the accuracy of the solution, it was decided to measure the deviation from zero velocity at the centre of the domain as a function of time, as was done in the two dimensional case (FIG. 6). This is presented in FIG. 10. It was decided to temporarily withhold parallel execution of the 3D code while the programme was run on a laptop, hence only serial data is shown. We notice immediately three main differences with the plot obtained in the two dimensional case:

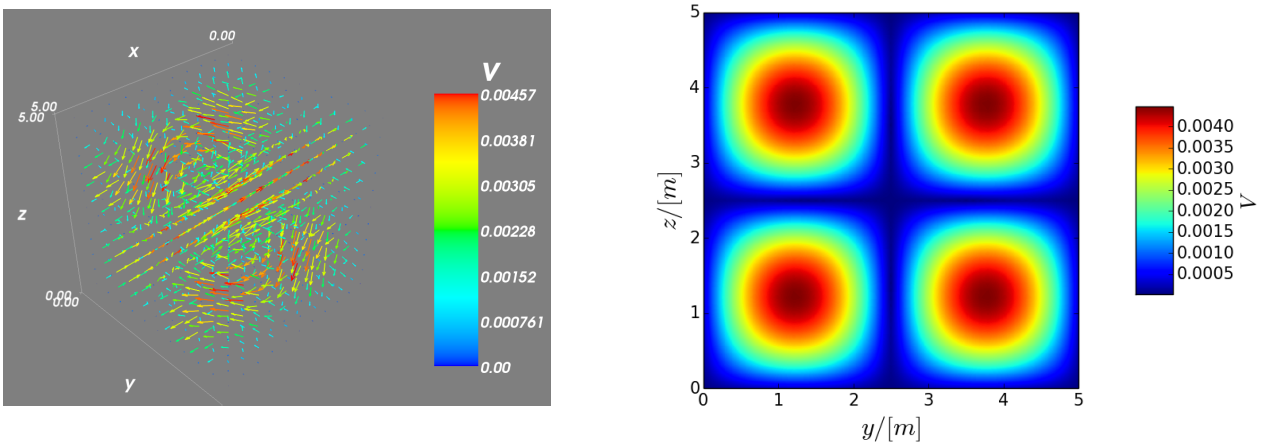


FIG. 9: The time evolution of the magnitude of the velocity field after thirty seconds. Left: The vector field projected isometrically. Right: The magnitude of the field observed in the $+x$ direction, positioned half way along the axis.

i) The deviation from zero velocity at the centre of the three dimensional domain as a function of time seems to be independent of the convergence criterion used (within less than $\sim 0.2\ \mu\text{ms}^{-1}$).

ii) The deviation is not monotonically decreasing, as was the case in two dimensions. Rather, there is a maximum at $t \simeq 12$ s.

iii) The maximum deviation is approximately an order of magnitude smaller than that in the two dimensional case.

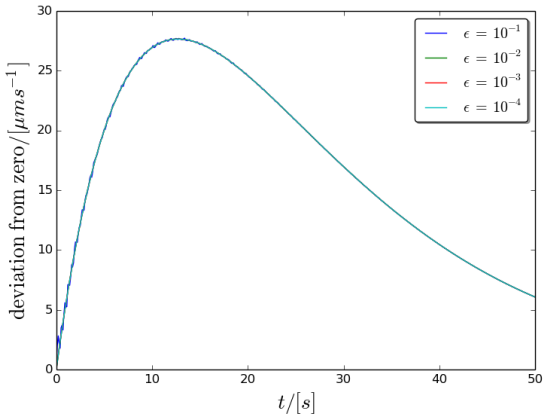


FIG. 10: The deviation of the magnitude of the velocity field from zero at the centre of the domain as a function of time (up to 50 s). The results are from serial execution of the programme.

CONCLUSION

We have successfully implemented software to solve the Navier-Stokes equations in both two and three dimensions, assuming imcompressibility and periodic boundary conditions. This was confirmed by designing initial velocity fields which are supposed to retain their shape upon time evolution, while diminishing in amplitude. The solutions were tested up to times of 100 s, and found to be consistent with expectations. The effect that changing the convergence criterion had on the deviation of the velocity vector at the centre of the domain as a function of time was investigated. In the two dimensional case, the error decreased monotonically. As ϵ was increased, oscillations in this error were reduced. In the three dimensional case, the error reached a maximum at approximately 10 s (thereafter decreasing), and the oscillations were much smaller. In both cases, the overall trend

was independent of ϵ. The programmes for both the two and three dimensional solutions were successfully parallelised for use on a high performance cluster.

FUTURE WORK

The simulation is not currently compatible with turbulent flow. Although the Navier-Stokes equations generally describe turbulent flow in analytic situations with a reasonable degree of accuracy, computational simulations are often susceptible to numerical instability. In order to successfully simulate turbulent flow, it is possible to use the 'Reynolds-Averaged Navier Stokes' equations, which are time-averaged equations for fluid flow. This might be a possible route of improvement for future development.

It was mentioned that in order to simulate convective weather phenomenon, it is necessary to include additional equations which must be satisfied, which account for thermodynamic effects. This might be a more straightforward addition to the code than incorporating turbulence, and it might be possible to approximate a convective weather phenomenon as a purely laminar system, whilst sustaining convection.

ACKNOWLEDGMENTS

I would like to thank Professor John Hannay for his continued interest in this project, and for all the helpful advice he was able to give me.

[1] Clay Mathematics Institute, "Millinnium problems." `http://www.claymath.org/millenium-problems/navier-stokes-equation`. Accessed: 19/09/14.
[2] A. J. Chorin, "Numerical solution of the navier-stokes equations," *Mathematics of Computation*, vol. 22, no. 104, 1968.
[3] M. Fritzsche, *Parallel Numerical Simulation of Navier-Stokes Equations on GPUs*. PhD thesis, Friedrich-Schiller-Universität Jena, 2009.
[4] L. D. Landau and E. M. Lifshitz, *Fluid Mechanics*, vol. 6. London: Pergamon Press Ltd, 1959.
[5] E. F. Toro, *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Berlin: Springer, third ed., 2010.
[6] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes*. Cambridge: Cambridge University Press, third ed., 2007.
[7] A. Paterson, *A first course in fluid dynamics*. Cambridge: Cambridge University Press, 1983.
[8] MPICH, "Mpich." `http://http://www.mpich.org/`. Accessed: 28/08/14.
[9] F. S. University, "Parallel 2d poisson equation solver using mpi." `http://http://people.sc.fsu.edu/~jburkardt/c_src/poisson_mpi/poisson_mpi.html`. Accessed: 28/08/14.