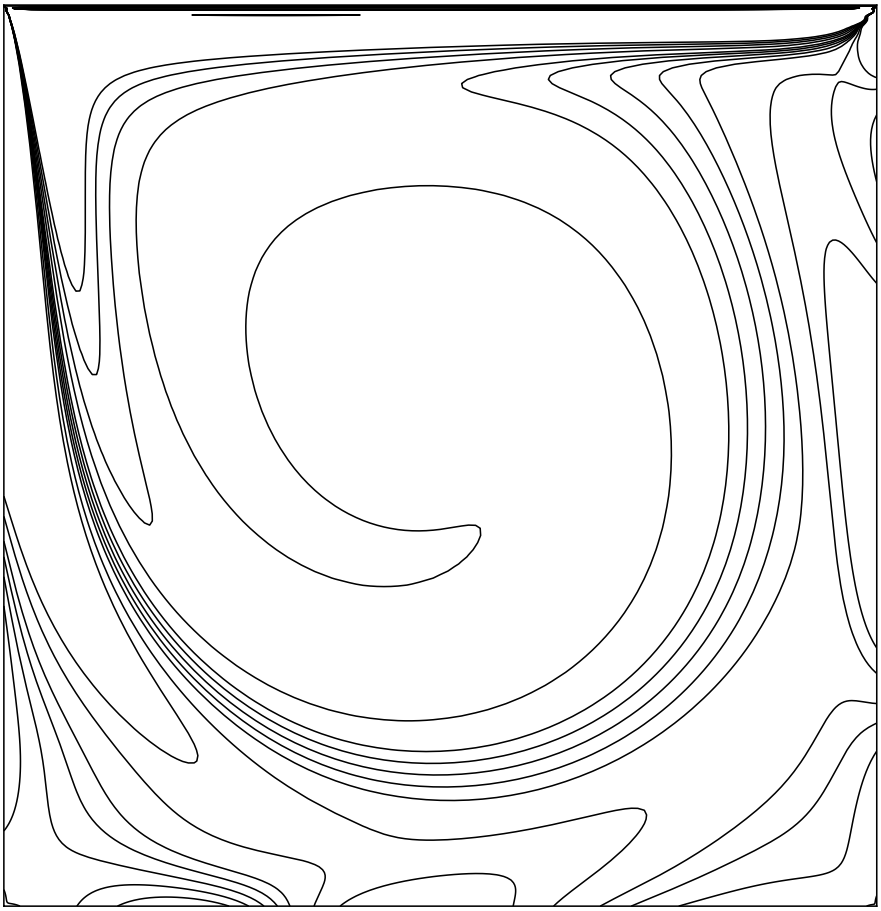
Delft University of Technology

AE4134 - Computational Fluid Dynamics I

Implementation of a Navier-Stokes Solver using Discrete Exterior Calculus



Michel Robijns (Student ID: 4088018)

April 25, 2016

Contents

1 Introduction

The vast majority of conventional numerical methods for discretizing and solving partial differential equations (for instance, the finite difference method, the finite volume method, and the finite element method) are based on the idea of approximating derivatives as a means to approximate partial differential equations. The discrete derivative either requires a small spacing between the nodal points or computationally expensive higher-order approximations to yield a small discretization error. The idea is valid, but has a number of disadvantages; truncation errors are introduced right from the start and the magnitude of the truncation error is a direct function of the spacing between the nodal points.

The objective of this report is the presentation of a novel method for solving the incompressible Navier-Stokes equations. The main aim of this report is to demonstrate that the method works and to discuss its advantages over conventional numerical methods. The method follows a geometric approach that preserves the physical relations that are inherently geometrical. This remarkable property is achieved by using tools from a vast mathematical field called discrete exterior calculus (DEC). A key ingredient in this approach is avoiding approximations of derivatives entirely by expressing all state variables as integrated quantities that "live" on associated the mesh elements. The method is therefore mathematically exact until one the last stages of the process where the spacing between the nodal points is brought into the equation.

The present method will be used to solve the incompressible Navier-Stokes equations on a two-dimensional lid-driven cavity. The lid-driven cavity flow problem is a classical test problem for the validation of Navier-Stokes codes and there is a plethora of both experimental and numerical results that can be used for the purpose of validation. In Chapter 2 we will derive the governing equations and formally present the lid-driven cavity flow problem. In Chapter 3 we will introduce the numerical framework by example and in a step-by-step manner. In Chapter 4 we will show the most important excerpt of the program code and look for optimizations of the code in terms of execution time and memory use. Finally, in Chapter 5 we will validate the results by means of a comparison with benchmark results from a 1997 paper by O. Botella and R. Peyret [1].

2 Physics

In this chapter, we will present the equations that govern the motion of viscous fluids and introduce the problem on which these governing equations are imposed.

2.1 The Navier-Stokes Equations

The Navier-Stokes equations govern the motion of viscous fluids. These equations arise from the application of conservation of mass, conservation of momentum, and conservation of energy to the motion of fluids and build upon the notion that fluids are a continuous medium rather than a set of discrete particles. The so-called convective form of the Navier-Stokes equations for an incompressible Newtonian fluid without the presence of body forces is [2]:

Continuity equation:
$$\nabla \cdot \mathbf{u} = 0 \tag{2.1a}$$

Momentum equation:
$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} \tag{2.1b}$$

where $\mathbf{u}$ is the velocity field, t is the time, ρ is the density, p is the pressure field, and ν is the kinematic viscosity. A more tangible notation of these equations is obtained by writing them in their vector component forms and by utilizing Einstein's summation convention:

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{2.2a}$$

$$\frac{\partial u_i}{\partial t} + u_j \frac{\partial u_i}{\partial x_j} = -\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \nu \frac{\partial^2 u_i}{\partial x_j^2} \tag{2.2b}$$

Equations (2.2a) and (2.2b) are dimensional. That is, each parameter like u_i, p, and ν is expressed in terms of a physical quantity. Non-dimensionalization can reduce the number of free parameters and can help to gain a greater insight into the relative size of the terms present in the equations. The dimensionless parameters are defined as follows:

$$u_i^* \equiv \frac{u_i}{U} \tag{2.3a}$$

$$x_i^* \equiv \frac{x_i}{L} \tag{2.3b}$$

$$t^* \equiv t\frac{U}{L} \tag{2.3c}$$

$$p^* \equiv \frac{p}{\rho U^2} \tag{2.3d}$$

$$\mathrm{Re} \equiv \frac{\nu}{UL} \tag{2.3e}$$

where U is the velocity scale and L is the length scale of the flow.

Multiplication of Equations (2.2a) and (2.2b) by L/U and L/U^2, respectively, gives

$$\frac{L}{U}\frac{\partial u_i}{\partial x_i} = 0 \tag{2.4a}$$

$$\frac{L}{U^2}\frac{\partial u_i}{\partial t} + \frac{L}{U^2}u_j\frac{\partial u_i}{\partial x_j} = -\frac{L}{U^2}\frac{1}{\rho}\frac{\partial p}{\partial x_i} + \frac{L}{U^2}\nu\frac{\partial^2 u_i}{\partial x_j^2} \tag{2.4b}$$

Rearrangement of Equations (2.4a) and (2.4b) yields

$$\frac{\partial \frac{u_i}{U}}{\partial \frac{x_i}{L}} = 0 \tag{2.5a}$$

$$\frac{\partial \frac{u_i}{U}}{\partial t\frac{U}{L}} + \frac{u_j}{U}\frac{\partial \frac{u_i}{U}}{\partial \frac{x_j}{L}} = -\frac{1}{\rho}\frac{\partial \frac{p}{U^2}}{\partial \frac{x_i}{L}} + \frac{\nu}{UL}\frac{\partial^2 \frac{u_i}{U}}{\partial \frac{x_j}{L}\partial \frac{x_j}{L}} \tag{2.5b}$$

Substituting Equations (2.3a)–(2.3e) into Equations (2.5a) and (2.5b), we obtain

$$\frac{\partial u_i^*}{\partial x_i^*} = 0 \tag{2.6a}$$

$$\frac{\partial u_i^*}{\partial t^*} + u_j^*\frac{\partial u_i^*}{\partial x_j^*} = -\frac{\partial p^*}{\partial x_i^*} + \frac{1}{\mathrm{Re}}\frac{\partial^2 u_i^*}{\partial x_j^*\partial x_j^*} \tag{2.6b}$$

or

$$\nabla\cdot\mathbf{u}^* = 0 \tag{2.7a}$$

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + (\mathbf{u}^*\cdot\nabla)\,\mathbf{u}^* = -\nabla p^* + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{u}^* \tag{2.7b}$$

Because it is now understood that the governing equations are dimensionless, the asterisk will be omitted in the remainder of this report. Thus,

$$\nabla\cdot\mathbf{u} = 0 \tag{2.8a}$$

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\,\mathbf{u} = -\nabla p + \frac{1}{\mathrm{Re}}\nabla^2\mathbf{u} \tag{2.8b}$$

For reasons that will be the subject of Chapter 3, it is convenient to rewrite the Navier-Stokes equations in terms of the divergence, curl, and gradient operators. Consider the following vector identities: if $\mathbf{u}$ is a three-dimensional vector field, then

$$\nabla^2\mathbf{u} = \nabla(\nabla\cdot\mathbf{u}) - \nabla\times(\nabla\times\mathbf{u}) \tag{2.9}$$

and

$$(\mathbf{u}\cdot\nabla)\,\mathbf{u} = \nabla\left(\frac{1}{2}\|\mathbf{u}\|^2\right) - \mathbf{u}\times(\nabla\times\mathbf{u}) \tag{2.10}$$

Using these vector identities, Equation (2.8b) can be written as

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla\left(\frac{1}{2}\|\mathbf{u}\|^2\right) - \mathbf{u}\times(\nabla\times\mathbf{u}) = -\nabla p + \frac{1}{\mathrm{Re}}\left(\nabla(\nabla\cdot\mathbf{u}) - \nabla\times(\nabla\times\mathbf{u})\right) \tag{2.11}$$

Recall that in a velocity field, the curl of the velocity is equal to the vorticity:

$$\xi = \nabla\times\mathbf{u} \tag{2.12}$$

Substituting Equation (2.12) into Equation (2.11), we have

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla\left(\frac{1}{2}\|\mathbf{u}\|^2\right) - \mathbf{u}\times\xi = -\nabla p + \frac{1}{\mathrm{Re}}\left(\nabla(\nabla\cdot\mathbf{u}) - \nabla\times\xi\right) \tag{2.13}$$

Some algebraic rearrangement of Equation (2.13) yields

$$\frac{\partial \mathbf{u}}{\partial t} - \mathbf{u} \times \xi + \nabla \left(p + \frac{1}{2} \|\mathbf{u}\|^2 \right) = \frac{1}{\mathrm{Re}} \left(\nabla \left(\nabla \cdot \mathbf{u} \right) - \nabla \times \xi \right) \tag{2.14}$$

In Equation (2.14), denote

$$P \equiv p + \frac{1}{2} \|\mathbf{u}\|^2 \tag{2.15}$$

and recognize that the continuity equation, $\nabla \cdot \mathbf{u} = 0$, appears in the right-hand side. Hence, Equation (2.14) is written as

$$\frac{\partial \mathbf{u}}{\partial t} - \mathbf{u} \times \xi + \nabla P + \frac{1}{\mathrm{Re}} \nabla \times \xi = 0 \tag{2.16}$$

Thus, the final system of equations is given by:

Continuity equation: $$\nabla \cdot \mathbf{u} = 0 \tag{2.17a}$$

Velocity-vorticity relation: $$\xi = \nabla \times \mathbf{u} \tag{2.17b}$$

Momentum equation: $$\frac{\partial \mathbf{u}}{\partial t} - \mathbf{u} \times \xi + \nabla P + \frac{1}{\mathrm{Re}} \nabla \times \xi = 0 \tag{2.17c}$$

2.2 The Lid-Driven Cavity Flow Problem

The incompressible dimensionless Navier-Stokes equations will be solved on the two-dimensional square domain $\Omega \equiv [0, 1] \times [0, 1]$, shown in Figure 2.1. The boundary of Ω is defined as $\partial\Omega \equiv \Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_4$.

The top boundary or "lid" of Ω, Γ_3, imposes a shear stress on the fluid by moving leftward with a velocity of unity, hence the name lid-driven cavity flow problem. Thus, the boundary conditions are $\mathbf{u}|_{\Gamma_3} = (-1, 0)$ and $\mathbf{u}|_{\Gamma_1} = \mathbf{u}|_{\Gamma_2} = \mathbf{u}|_{\Gamma_4} = (0, 0)$. At $t = 0$, the fluid is at rest. That is, $\mathbf{u}|_{t=0} \equiv 0$.

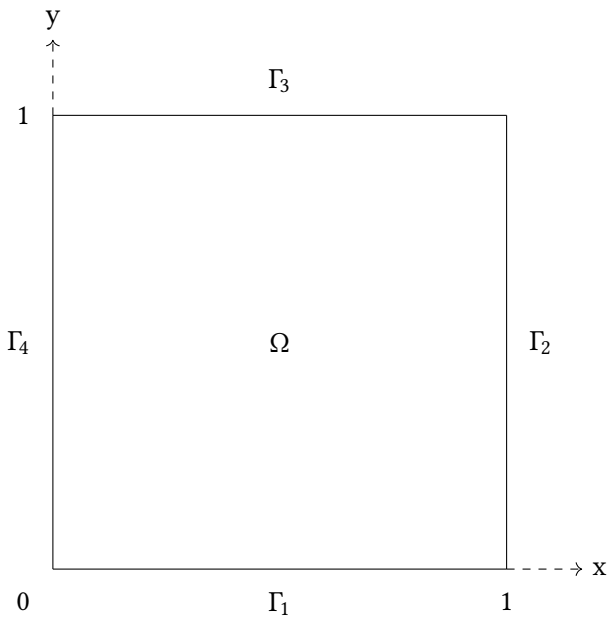


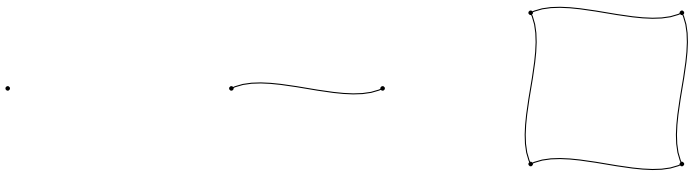
Figure 2.1: The lid-driven cavity flow problem.

3 Mathematics

In Chapter 2 we derived a system of dimensionless partial differential equations that govern the motion of viscous fluids. In this chapter, we cover the mathematical machinery necessary to express the problem into a format understandable by a computer. Instead of using classical discretization methods like the finite difference method, we take a geometric approach using tools from discrete exterior calculus (DEC), see [3] and [4]. A key ingredient in this geometric approach is the placement of physical quantities on the appropriate geometric structures. DEC is a vast mathematical field and a rigorous treatment is far beyond the scope of this assignment. The scope of this chapter is therefore limited to those particular concepts that are required to solve the lid-driven cavity flow problem.

3.1 Cells

Let us first introduce a discrete version of two-dimensional space. Two-dimensional space supports geometric structures with either zero, one, or two dimensions: points, lines, and planes. These k-dimensional geometric structures are formally called k-cells, denoted $\sigma^{(k)}$, where k represents the dimension of the cell. Three examples of k-cells are shown in Figure 3.1.



(a) 0-cell (point) (b) 1-cell (line) (c) 2-cell (plane)

Figure 3.1: Examples of k-cells in two-dimensional space.

A point is a 0-cell, a line is a 1-cell, and a plane is a 2-cell. Just like a 1-cell is a line with a length but with no position, a 2-cell can be thought of as a plane with an area, but no position. Notice that k-cells are bounded by $(k-1)$-cells and that a cell's shape is arbitrary, as indicated by the wavy lines. (*Note:* The boundary of a point is empty by definition.)

We will be using points, lines, and planes as building blocks of our mesh. A mesh is a set of cells, called a cell-complex if the boundary of each cell is also part of the set. For example, the (rather primitive) mesh in Figure 3.1c is a cell-complex because the plane is bounded by lines and the lines are in turn bounded by points, all of which are part of the mesh.

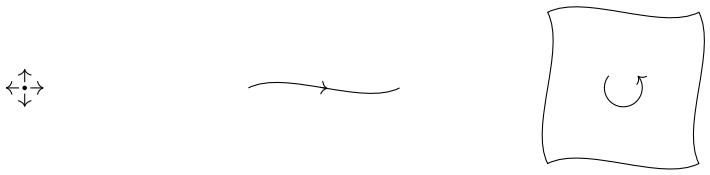
3.2 Chains and Cochains

There are two additional concepts that require some elaboration: the notion of k-chains and the notion of k-cochains. A k-chain is essentially a set of k-cells. That is, nothing more and nothing less than a set of geometric structures *of the same dimension*. For instance, a set of lines or a set of planes. k-cochains build upon chains; a k-cochain is essentially a set of k-cells tagged with a numerical value. k-cochains are in practice realized as a vector of single or double precision numbers and it may help to think of k-cochains as such. Rigorous definitions of chains and cochains can be found in [5].

3.3 Orientation

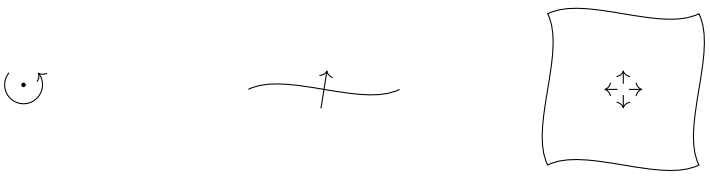
In addition to dimension, k-cells have a property called orientation. A k-cell either has a positive or a negative orientation that can be freely chosen per individual cell, as long as its orientation remains unchanged thereafter. However, in practice it may be a good idea to define an orientation that is consistent throughout all k-cells.

Figure 3.2 shows three k-cells in two-dimensional space with an intrinsic orientation that we call inner-oriented k-cells. Inner-oriented points for instance, can either represent sources or sinks and are arbitrarily chosen to be source-like (that it, an outflow is positive), as indicated by the outward pointing arrows in Figure 3.2a. Inner-oriented lines are chosen to be a positive when pointing to the right, as indicated by the arrow in Figure 3.2b. It is somewhat awkward to think of the orientation of a plane, but an inner-oriented plane is defined as positive when it is oriented counterclockwise, as indicated by the curled arrow in Figure 3.2c.



(a) 0-cell (b) 1-cell (c) 2-cell

Figure 3.2: Inner-oriented k-cells in two-dimensional space.



(a) 0-cell (b) 1-cell (c) 2-cell

Figure 3.3: Outer-oriented k-cells in two-dimensional space.

Inner-oriented k-cells have outer-oriented counterparts with an circumcentric orientation, rather than an intrinsic orientation. For example, the orientation of an

inner-oriented line is defined *on* that line whereas the orientation of an outer-oriented line is defined *through* that line. Figure 3.3 shows three outer-oriented k-cells in two-dimensional space.

The distinction between inner and outer-oriented k-cells is justified from a physical of view; certain physical quantities are naturally expressed in terms of inner-oriented cochains whereas other physical quantities are naturally expressed in terms of outer-oriented cochains.

3.4 The Exterior Derivative

The exterior derivative δ is used to compute derivatives, like gradients, divergences, or curls. The descrete exterior derivative is a linear map from a (k)-cochain to a $(k+1)$-cochain. It turns out that the standard treatment of vector calculus hides a number of important things that are applied implicitly. As shall be seen in the upcoming examples, all three of the multivariable derivatives in vector calculus (i.e. gradients, curls, and divergences) are actually *one* kind of derivative: the exterior derivative. The exterior derivative will be introduced by means of examples to provide a suitable context for the discussion of the physical interpretation of k-cochains.

3.4.1 Inner-Oriented Quantities

To capture the geometric structure of the governing equations, we define its physical quantities through integral values over the elements of the mesh. Depending on whether a given physical quantity is a point, line, or area density, its corresponding discrete representation "lives" at the associated zero, one, or two dimensional mesh elements.

We would eventually like to know the velocity field within the domain of our problem. We use circulation, i.e., the line integral of the velocity field, to encode velocity on the mesh elements. Let inner-oriented 1-cochains therefore represent circulation, defined as

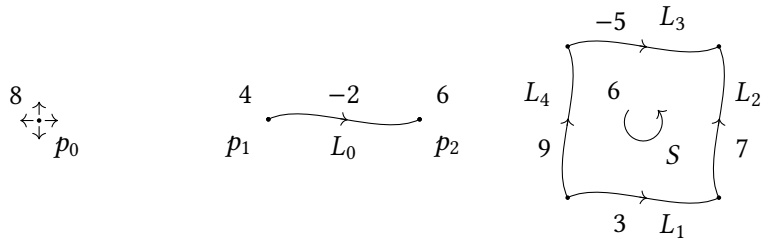
$$u_{i,j} \equiv \int_L \mathbf{u} \cdot \hat{\mathbf{t}}\, dx \tag{3.1a}$$

and

$$v_{i,j} \equiv \int_L \mathbf{u} \cdot \hat{\mathbf{t}}\, dy \tag{3.1b}$$

where $\hat{\mathbf{t}}$ denotes the tangent vector along a line and where the notations u and v are reserved for horizontal and vertical lines, respectfully. It is crucial to note that this makes circulation an *integrated*, not pointwise, quantity. Thus, the physical quantity of circulation "lives" at the one-dimensional mesh elements (i.e., lines).

Figure 3.4 shows three examples of inner-oriented k-cochains in two-dimensional space. Recall that cochains can be thought of as chains tagged with a number, hence the numerical values in Figure 3.4. Figure 3.4b shows a 0-cochain consisting of two points, p_1 and p_2, and a 1-cochain composed of a line, L_0. L_0 points from p_1 to p_2, so p_1 acts as a source and p_2 acts as a sink. The recipe for applying δ is straightforward: simply add values at points that act as a source and subtract values at points that



(a) 0-cochain (b) 1-cochain (c) 2-cochain

Figure 3.4: Examples of k-cochains in two-dimensional space.

act as a sink. Because points are source-like by default, as indicated in Figure 3.4a, application of δ results in

$$u = \delta(L_0) = 4 - 6 = -2 \tag{3.2}$$

Assuming that points represent samples of a continuous scalar field, P, it is clearly seen that the application of δ on a 0-cochain is analogous to the *integrated* gradient operator:

$$\begin{aligned} 4 - 6 &= P(p_2) - P(p_1) \\ &= \int_L \nabla\phi \cdot \mathbf{ds} \end{aligned} \tag{3.3}$$

P can be interpreted of as a dimensionless version of total pressure ($P \equiv p + \frac{1}{2}||\mathbf{u}||^2$).

Figure 3.4c shows a 1-cochain composed of four lines and a 2-cochain composed of a single plane, S. We apply the discrete exterior derivative by either adding or subtracting the line values; if the orientation of a line segment opposes the orientation of the plane, then its value must be subtracted. Thus, application of δ on S results in

$$\xi = \delta(S) = 3 + 7 - (-5) - 9 = 6 \tag{3.4}$$

Application of δ on a 1-cochain is analogous to the integrated curl operator because

$$\begin{aligned} 3 + 7 - (-5) - 9 &= u_1(L_1) + v_2(L_2) - u_2(L_3) - v_1(L_4) \\ &= \int_{L_1} \mathbf{u} \cdot \mathbf{dl} + \int_{L_2} \mathbf{u} \cdot \mathbf{dl} - \int_{L_3} \mathbf{u} \cdot \mathbf{dl} - \int_{L_4} \mathbf{u} \cdot \mathbf{dl} \\ &= \oint_{\partial S} \mathbf{u} \cdot \mathbf{dl} \\ &= \iint_S (\nabla \times \mathbf{u}) \cdot dA \end{aligned} \tag{3.5}$$

where $\partial S \equiv L_1 \cup L_2 \cup L_3 \cup L_4$. This is not the only insight to be gained here; recall that the circulation along a closed contour, denoted Γ, is defined as:

$$\Gamma \equiv \oint_C \mathbf{u} \cdot \mathbf{dl} \tag{3.6}$$

Comparing Equations (3.5) and (3.6), we conclude that $\delta(S)$ must represent the circulation around the edges of S. Circulation is related to vorticity; the circulation

around a closed contour is equal to the integrated vorticity enclosed by that contour. From Stokes' theorem:

$$\Gamma \equiv \oint_C \mathbf{u} \cdot \mathbf{dl} = \iint_A (\nabla \times \mathbf{u})\, dA = \iint_A \xi\, dA \tag{3.7}$$

Inner-oriented 2-cochains ξ can therefore be interpreted as vorticity.

Because 3-cochains are undefined in two-dimensional space, the application of δ on a 2-cochain results in the empty set by definition. In a similar vein, 0-cochains can only be the result of the application of δ on a real number [4].

3.4.2 Outer-Oriented Quantities

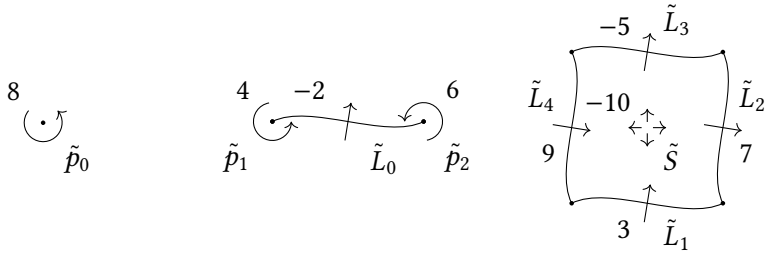
We use flux, i.e., the integral of the normal velocity along a line, as a representation of velocity on an outer-oriented mesh. Note again that this makes flux an integrated, not pointwise, quantity. Thus, let the outer-oriented 1-cochains represent flux, defined as

$$\tilde{u}_{i,j} \equiv \int_L \mathbf{u} \cdot \hat{\mathbf{n}}\, dy \tag{3.8a}$$

and

$$\tilde{v}_{i,j} \equiv \int_L \mathbf{u} \cdot \hat{\mathbf{n}}\, dx \tag{3.8b}$$

where $\hat{\mathbf{n}}$ denotes the normal vector along the line segment and where $u_{i,j}$ and $v_{i,j}$ are again reserved for horizontal and vertical lines, respectively. All outer-oriented geometric structures and quantities are marked with a tilde for easy recognition. Figure 3.5 shows three examples of outer-oriented k-cochains in two-dimensional space.



(a) 0-cochain (b) 1-cochain (c) 2-cochain

Figure 3.5: Examples of k-cochains in two-dimensional space.

Figure 3.5b shows a 0-cochain composed of two points, $\tilde{p}_1$ and $\tilde{p}_2$, and a 1-cochain composed of a line, $\tilde{L}_0$. The recipe for applying δ is again straightforward: add values at points that share a common orientation with the line and subtract values whose orientation opposes the orientation of the line. The orientation of $\tilde{p}_2$, for example, opposes the orientation of $\tilde{L}_0$ because the arrows point in opposite directions. Thus,

$$\tilde{u} = \delta(\tilde{L}_0) = 4 - 6 = -2 \tag{3.9}$$

This procedure is analogous to the integrated gradient operator because

$$\begin{aligned} 4 - 6 &= \tilde{\psi}(p_2) - \tilde{\psi}(p_1) \\ &= \int_L \nabla\tilde{\psi} \cdot \mathbf{ds} \end{aligned} \tag{3.10}$$

Let us consider the physical interpretation of 0-cochains. If lines represent flux, then points must represent the stream function, $\tilde{\psi}$. Here is why: the numerical value of the stream function is defined such that the difference $\Delta\tilde{\psi}$ between two streamlines is equal to the mass flux between the two streamlines. This fact allows us to determine the stream function up to a constant.

Figure 3.5c shows an outer-oriented 1-cochain composed of four lines and an outer-oriented 2-cochain composed of a plane, $\tilde{S}$. The plane is source-like as indicated by the outward pointing arrows. We apply the discrete exterior derivative by adding inflows and subtracting outflows. Thus, application of δ yields

$$\tilde{\xi} = \delta(\tilde{S}) = -3 + 7 - 5 - 9 = -10 \tag{3.11}$$

The application of δ on a 1-cochain is analogous to the integrated curl operator because

$$\begin{aligned} -3 + 7 + (-5) - 9 &= -\tilde{u}_1(\tilde{L}_1) + \tilde{v}_2(\tilde{L}_2) + \tilde{u}_2(\tilde{L}_3) - \tilde{v}_1(\tilde{L}_4) \\ &= -\int_{\tilde{L}_1} \mathbf{u}\cdot\mathbf{dl} + \int_{\tilde{L}_2} \mathbf{u}\cdot\mathbf{dl} + \int_{\tilde{L}_3} \mathbf{u}\cdot\mathbf{dl} - \int_{\tilde{L}_4} \mathbf{u}\cdot\mathbf{dl} \\ &= \oint_{\partial\tilde{S}} \mathbf{u}\cdot\mathbf{dl} \\ &= \iint_{\tilde{S}} (\nabla\times\mathbf{u})\cdot dA \end{aligned} \tag{3.12}$$

where $\partial\tilde{S} \equiv \tilde{L}_1 \cup \tilde{L}_2 \cup \tilde{L}_3 \cup \tilde{L}_4$.

What is the physical interpretation of $\tilde{\xi}$? The fluid is incompressible, so the velocity field must be divergence free (i.e., $\nabla\cdot\mathbf{u} = 0$). According to the generalized Stokes' theorem the integral of the divergence over a plane equals the sum of the fluxes on all its four edges. In other words, everything that gets in must also get out. The 2-cochain $\tilde{\xi}$ therefore represents the rate of mass production in the plane at which it "lives". Thus, for the law of mass conservation to hold true, $\tilde{\xi}$ must be equal to zero at each and every plane because the problem is incompressible.

3.4.3 DeRham Complex

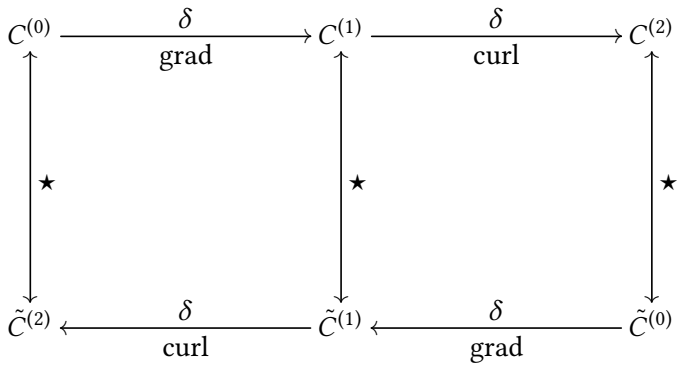
A major advantage of the discrete exterior derivative is that it satisfies the same rules and identities as its smooth counterpart. The discrete exterior derivative is mathematically exact because it acts on and produces integrated quantities. That is, application of δ does not introduce any error whatsoever. As we shall see, this property contributes to the strength of the geometric approach and it results in preservation of the physics implied in the smooth governing equations. The results of this section can be summarized in a diagram called the DeRham complex [5]:

$$\begin{aligned} C^{(0)} &\xrightarrow[\text{grad}]{\delta} C^{(1)} \xrightarrow[\text{curl}]{\delta} C^{(2)} \\ \tilde{C}^{(0)} &\xrightarrow[\text{grad}]{\delta} \tilde{C}^{(1)} \xrightarrow[\text{curl}]{\delta} \tilde{C}^{(2)} \end{aligned} \tag{3.13}$$

3.5 The Hodge-⋆ Operator

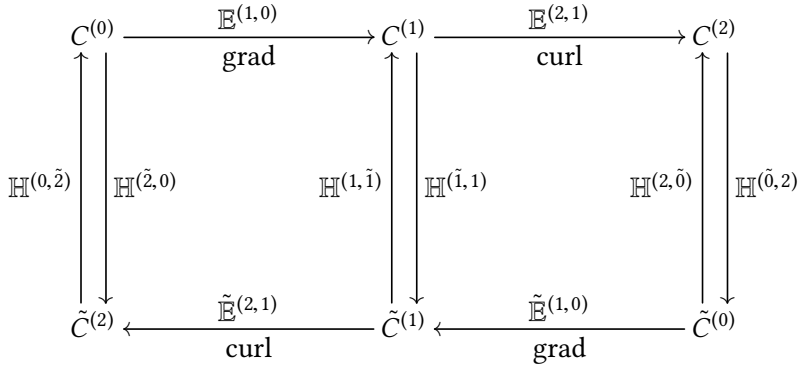
We have established that two-dimensional space supports points, lines, and planes. Let us briefly digress to three-dimensional space for the sake of familiarity; three-dimensional space supports points, lines, planes, and volumes. In three-dimensional space, we can have one linearly independent volume, three linearly independent planes, three linearly independent lines (just like vectors in $\mathbb{R}^3$), and again one linearly independent point. This 1–3–3–1 sequence suggests a pairing; there exists a vector space isomorphism between k-vectors and $(n-k)$-vectors. The Hodge-$\star$ operator is a linear map between inner-oriented k-cochains and outer-oriented $(n-k)$-cochains or vice versa, where n denotes the dimension of the vector space. It simply scales the quantities stored on the mesh cells by the size of the corresponding k-cells [6].

We have introduced the δ operator to map (k)-cochains to $(k+1)$-cochains. However, the Navier-Stokes equations equate both inner and outer oriented cochains, which requires the Hodge-$\star$ operator. The Hodge-$\star$ operator links the DeRham complices for inner and outer oriented cochains. This linked structure is called the double DeRham complex:



(3.14)

In a discrete setting, cochains are stored in vectors, the δ operator is represented by incidence matrices, denoted $\mathbb{E}$, and the Hodge-$\star$ operator is represented by Hodge matrices, denoted $\mathbb{H}$. In such a framework, application of the δ or Hodge-$\star$ operator becomes a matrix vector-multiplication. Hence, the double DeRham complex in Equation (3.14) becomes



(3.15)

3.6 Discretization of the Unit Square

It is impossible to derive the incidence and Hodge matrices without any a priori knowledge about the geometry of the grid or cell-complex. Let us therefore first discretize the unit square Ω into a grid composed of points, lines, and planes. The outer-oriented grid shown in Figure 3.6 was constructed for $n = 3$, where n denotes the number of planes in the horizontal and vertical directions. The grid has an orthogonal structure, but is not uniform. The points are distributed via cosine spacing at equal angular increments because the smallest flow features are expected to emerge in the vicinity of the walls, which justifies this particular refinement of the grid. The inner-oriented grid associated with the outer-oriented grid is constructed such that the inner-oriented points are located precisely in the center of the planes of the outer-oriented grid, see Figure 3.7. The inner-oriented grid *must* be constructed in this manner for reasons that can be found in literature [4].

3.7 Physical Representation

We briefly digress here to summarize the physical interpretation of the k-cells that compose the inner and outer oriented grids. The orientation of the outer-oriented planes $\tilde{S}_{i,j}$ is source-like, as indicated by the outward pointing arrows. That is, an outflow is considered positive and an inflow is considered negative. The outer-oriented line segments $\tilde{u}_{i,j}$ and $\tilde{v}_{i,j}$ represent flux in the form:

$$\tilde{u}_{i,j} = \int_{y_{i,j}}^{y_{i,j+1}} \mathbf{u} \cdot \hat{\mathbf{n}}\, dy \tag{3.16a}$$

and

$$\tilde{v}_{i,j} = \int_{x_{i,j}}^{x_{i+1,j}} \mathbf{u} \cdot \hat{\mathbf{n}}\, dx \tag{3.16b}$$

where $\hat{\mathbf{n}}$ denotes the normal vector along the line segment. (*Note:* it is important to remember that all physical quantities are still dimensionless.) A rightward and upward flux is considered positive, as indicated by the arrows through the line segments in Figure 3.6. If outer-oriented 1-cochains represent flux, then outer-oriented 0-cochains must represent samples of the the continuous stream function and outer-oriented 2-cochains must represent the rate of mass production within the plane on which they are defined.

The inner-oriented line segments $u_{i,j}$ and $v_{i,j}$ represent circulation:

$$u_{i,j} = \int_{x_{i,j}}^{x_{i,j+1}} \mathbf{u} \cdot \hat{\mathbf{t}}\, dx \tag{3.17a}$$

and

$$v_{i,j} = \int_{y_{i,j}}^{y_{i+1,j}} \mathbf{u} \cdot \hat{\mathbf{t}}\, dy \tag{3.17b}$$

where $\hat{\mathbf{t}}$ denotes the tangent vector along the line segment. As discussed in the preceding section, application of the integrated curl operator on a velocity field yields vorticity, assuming that the surfaces are infinitesimal. The inner-oriented 2-cochains $\xi_{(i,j)}$ do therefore represent vorticity and the inner-oriented 0-cochains $P_{i,j}$ represent total pressure. Notice that the orientation of the inner-oriented 0-cochains $P_{i,j}$ is sink-like, as indicated by the inward pointing arrows.

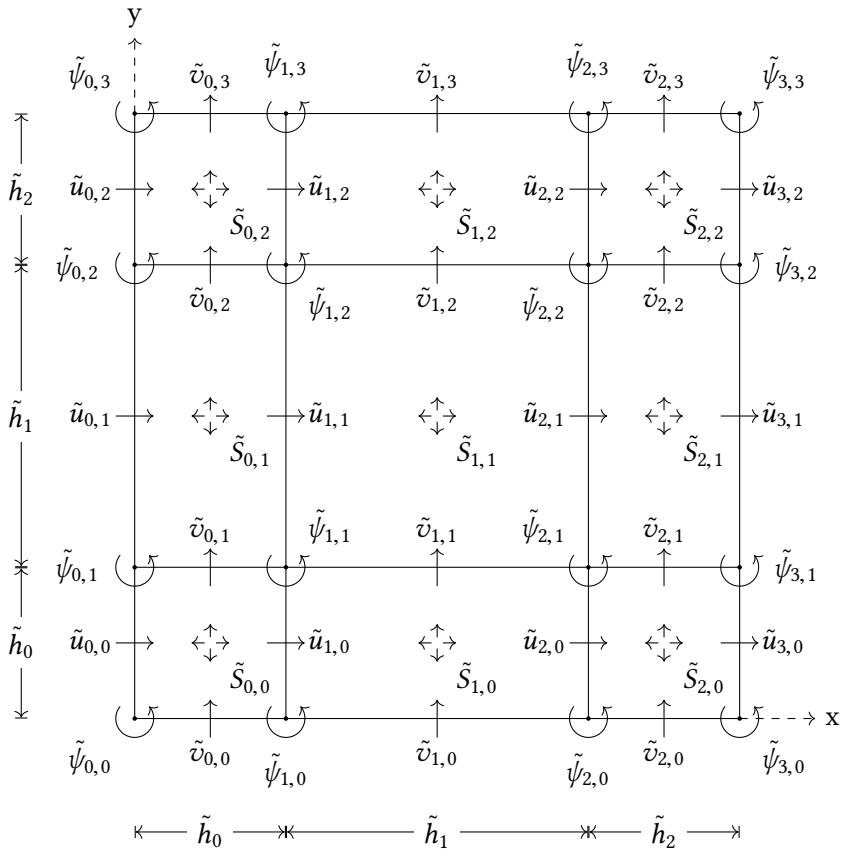


Figure 3.6: The outer-oriented grid.

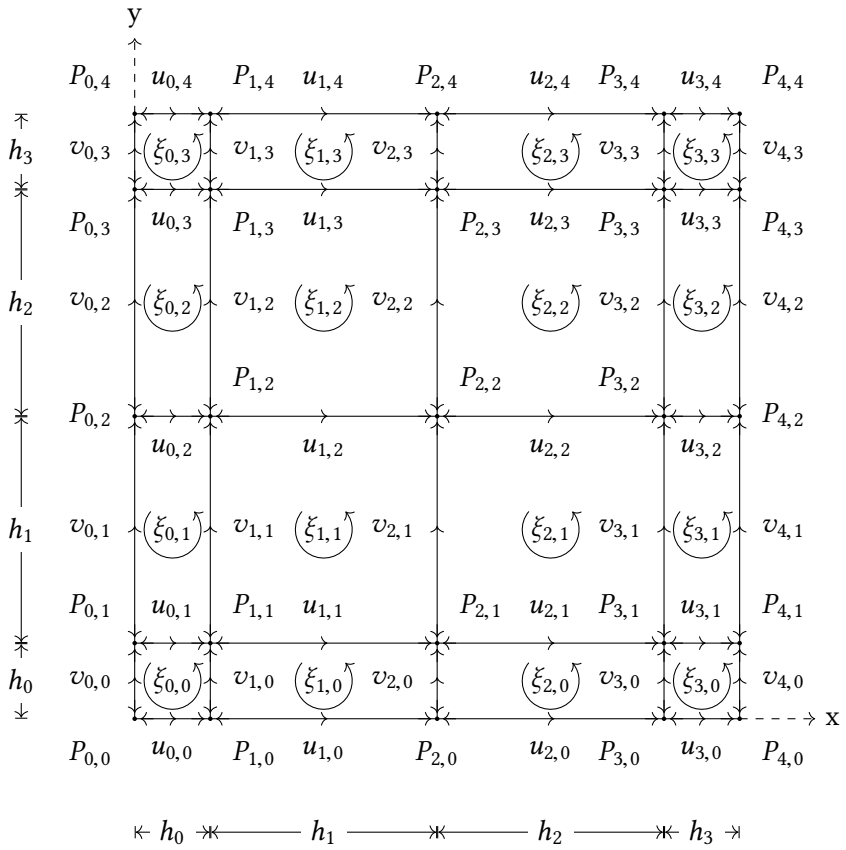
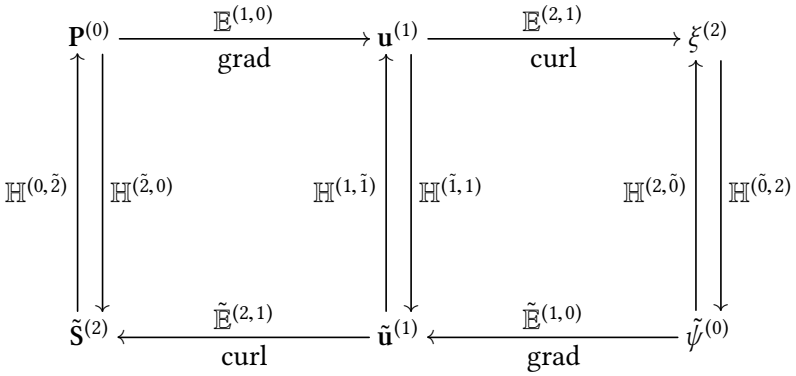


Figure 3.7: The inner-oriented grid.

We have constructed the inner and outer oriented grid and we have established what the k-cells physically represent. The double DeRham complex can be updated incorporating the physical quantities:



(3.18)

(*Note:* The variables in Equation (3.18) are written in boldface because their numerical values are typically stored in an array (i.e., a vector) on a computer. However, the physical quantities P and $\tilde{\psi}$ remain scalar quantities and should *not* be seen as vector quantities.)

3.8 Structure of the Navier-Stokes Equations

Recall the dimensionless incompressible Navier-Stokes equations that we derived in Chapter 2:

Continuity equation: $$\nabla \cdot \mathbf{u} = 0 \tag{3.19a}$$

Velocity-vorticity relation: $$\xi = \nabla \times \mathbf{u} \tag{3.19b}$$

Momentum equation: $$\frac{\partial \mathbf{u}}{\partial t} - \mathbf{u} \times \xi + \nabla P + \frac{1}{\mathrm{Re}} \nabla \times \xi = 0 \tag{3.19c}$$

These equations contain the variables that we defined on the elements of the mesh. Let us add superscripts to the variables to indicate that they are k-cochains and to indicate their dimension:

$$\nabla \cdot \mathbf{u}^{(1)} = 0 \tag{3.20a}$$

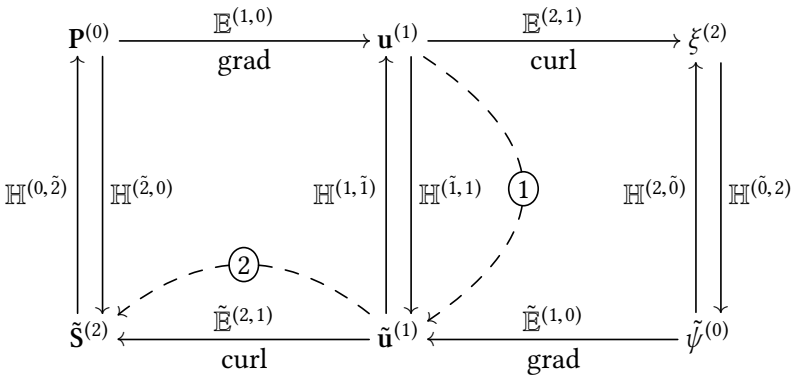
$$\xi^{(2)} = \nabla \times \mathbf{u}^{(1)} \tag{3.20b}$$

$$\frac{\partial \mathbf{u}^{(1)}}{\partial t} - \mathbf{u}^{(1)} \times \xi^{(2)} + \nabla P^{(0)} + \frac{1}{\mathrm{Re}} \nabla \times \xi^{(2)} = 0 \tag{3.20c}$$

To make these equations consistent, all terms within an equation *must* be reduced to k-cells of the same dimension *and* orientation. For example, equating 2-cochains and 1-cochains, especially of different orientations, is an invalid operation.

3.8.1 Continuity Equation

The continuity equation, Equation (3.20a), involves the divergence operator. The divergence operator is equivalent to (1) a mapping of velocity to mass flow and (2) a subsequent application of the curl operator. These two operations are annotated in the double DeRham complex below:



(3.21)

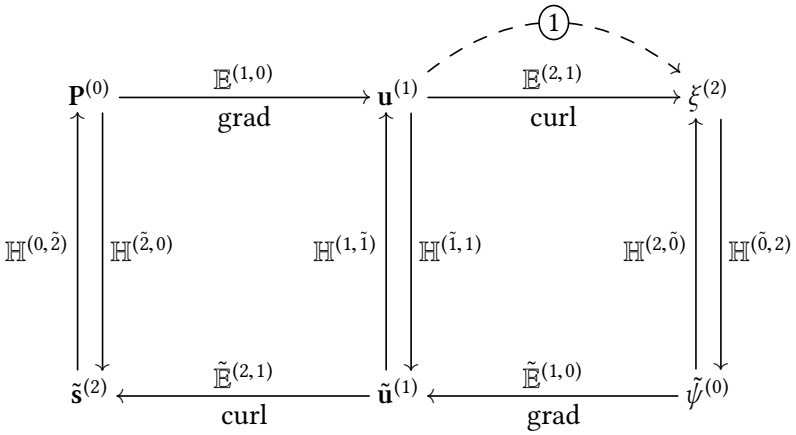
Thus, Equation (3.13a) is equivalent to

$$\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{(1)} = 0 \tag{3.22}$$

which is in turn equivalent to writing $\tilde{\mathbf{S}}^{(2)} = 0$. Recall that the physical interpretation of $\tilde{\mathbf{S}}$ is the rate of mass production within the planes that compose the mesh. Thus, $\tilde{\mathbf{S}}^{(2)} = 0$ implies that the net production of mass equals zero, which is precisely what the continuity equation represents.

3.8.2 Velocity-Vorticity relation

The velocity-vorticity relation states that vorticity, ξ, is equal to the curl of velocity, $\mathbf{u}$. This represents a mapping from an inner-oriented 1-cochain to an inner-oriented 2-cochain. This operation is represented by a single jump in the double DeRham complex, denoted (1) in the double DeRham complex below:



(3.23)

Equation 3.20b can therefore be written as

$$\xi^{(2)} = \mathbb{E}^{(2,1)}\mathbf{u}^{(1)} \tag{3.24}$$

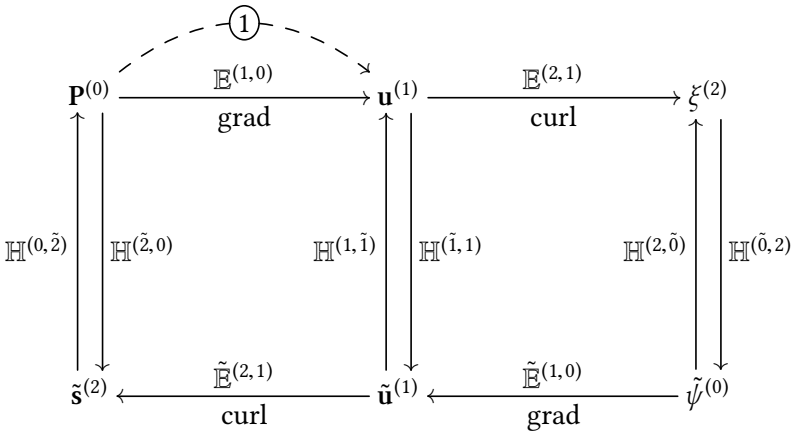
3.8.3 Momentum Equation

The momentum equation involves three terms that must be expressed in terms of incidence and Hodge matrices:

1. $\mathbf{u}^{(1)} \times \xi^{(2)}$
2. $\nabla P^{(0)}$
3. $\nabla \times \xi^{(2)}$

The cross product $\mathbf{u}^{(1)} \times \xi^{(2)}$ represents the nonlinear convective term of the Navier-Stokes equations and is a special case. We will therefore replace the nonlinear term by a generic vector named "convective" that will be derived at a later stage.

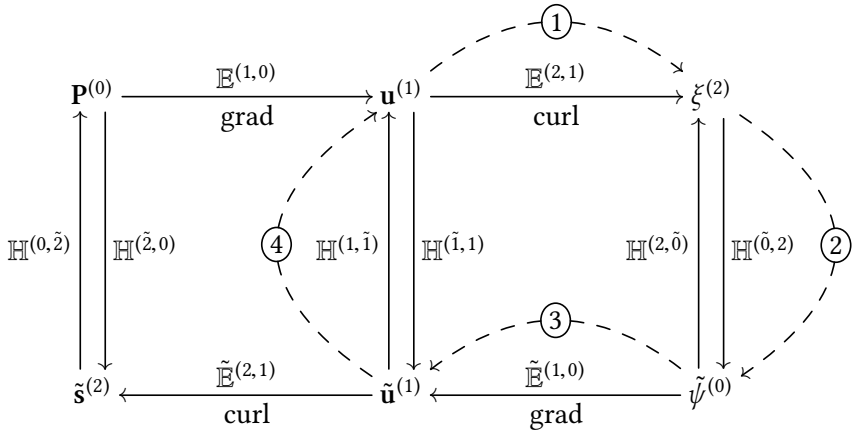
The pressure gradient $\nabla P^{(0)}$ represents a map from an inner-oriented 0-cochain to an inner-oriented 1-cochain. This mapping is denoted (1) in the double DeRham complex below:



(3.25)

The third term is not as straightforward. Both velocity and vorticity are vector fields and there would be no difference between the two in conventional vector calculus. However, within the realms of DEC, there is an important distinction between the two: velocity is an integral value associated with lines whereas vorticity is an integral value associated with surfaces.

First, apply the curl operator to velocity as usual, denoted (1). Second, map vorticity to its outer oriented counterpart, the stream function, denoted (2). Third, apply the gradient operator (which is the transpose of the curl operator $\mathbb{E}^{(2,1)}$ as we shall soon see) to obtain mass flow, denoted (3). Last, map the mass flow to velocity, denoted (4). This chain is graphically depicted in the double DeRham complex below:



(3.26)

Hence, Equation 3.20c can be written as

$$\frac{\partial \mathbf{u}^{(1)}}{\partial t} + \text{convective}^{(1)} + \mathbb{E}^{(1,0)} P^{(0)} + \frac{1}{\text{Re}} \mathbb{H}^{(1,\tilde{1})} \tilde{\mathbb{E}}^{(1,0)} \mathbb{H}^{(\tilde{0},2)} \mathbb{E}^{(2,1)} \mathbf{u}^{(1)} = 0 \tag{3.27}$$

3.8.4 The Navier-Stokes Equations in Terms of Incidence and Hodge Matrices

Thus, the Navier-Stokes equations can be rewritten in terms of incidence matrices and Hodge matrices as follows:

$$\tilde{\mathbb{E}}^{(2,1)} \mathbb{H}^{(\tilde{1},1)} \mathbf{u}^{(1)} = 0 \tag{3.28a}$$

$$\xi^{(2)} = \mathbb{E}^{(2,1)} \mathbf{u}^{(1)} \tag{3.28b}$$

$$\frac{\partial \mathbf{u}^{(1)}}{\partial t} + \text{convective}^{(1)} + \mathbb{E}^{(1,0)} P^{(0)} + \frac{1}{\text{Re}} \mathbb{H}^{(1,\tilde{1})} \tilde{\mathbb{E}}^{(1,0)} \mathbb{H}^{(\tilde{0},2)} \mathbb{E}^{(2,1)} \mathbf{u}^{(1)} = 0 \tag{3.28c}$$

3.9 The Incidence Matrices and Hodge Matrices

The incidence matrices $\mathbb{E}$ and the Hodge matrices $\mathbb{H}$ are derived using the mesh introduced in the Section 3.6. The incidence and Hodge matrices derived in this section are exclusively valid for the mesh shown in Figure 3.6 and Figure 3.7. That is, the validity is limited to the rather coarse spacing of $n = 3$. However, the format of those matrices belonging to denser grids can be correctly deduced by careful analysis of the matrix structure at $n = 3$, which is of course what we are after.

3.9.1 $\tilde{\mathbb{E}}^{(2,1)}$

The application of the incidence matrix $\tilde{\mathbb{E}}^{(2,1)}$ to the 1-cochain that represent flux, $\tilde{\mathbf{u}}^{(1)}$, yields the rate of mass production in the plane enclosed by those line segments, $\tilde{\mathbf{S}}^{(2)}$. Let us construct a linear equation for each of the planes $\tilde{s}_{i,j}$ in the mesh:

$$\begin{aligned}
\tilde{s}_{0,0} &= -\tilde{u}_{0,0} + \tilde{u}_{1,0} - \tilde{v}_{0,0} + \tilde{v}_{0,1} \\
\tilde{s}_{1,0} &= -\tilde{u}_{1,0} + \tilde{u}_{2,0} - \tilde{v}_{1,0} + \tilde{v}_{1,1} \\
&\vdots \\
\tilde{s}_{2,2} &= -\tilde{u}_{2,2} + \tilde{u}_{3,2} - \tilde{v}_{2,2} + \tilde{v}_{2,3}
\end{aligned} \tag{3.29}$$

Equation (3.29) expressed in matrix notation becomes

$$\tilde{\mathbf{s}}^{(2)} = \tilde{\mathbb{E}}^{(2,1)} \tilde{\mathbf{u}}^{(1)} \tag{3.30}$$

The mass flow rates $\tilde{u}_{i,j}$ and $\tilde{v}_{i,j}$ adjacent to the boundary of the unit square are known because the boundary conditions of the problem are known. The matrices in the right-hand side of Equation (3.30) can be split into a matrix of unknowns and into a matrix of knows. Splitting the matrix into two parts yields

$$\tilde{\mathbf{s}}^{(2)} = \tilde{\mathbb{E}}^{(2,1)} \tilde{\mathbf{u}}^{(1)} + \tilde{\mathbb{E}}^{(2,1)}_{\text{known}} \tilde{\mathbf{u}}^{(1)}_{\text{known}} \tag{3.31}$$

where

$$\tilde{\mathbb{E}}^{(2,1)} = \begin{bmatrix} 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & 1 & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 \\ \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & -1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & -1 \end{bmatrix} \tag{3.32}$$

and

$$\tilde{\mathbb{E}}^{(2,1)}_{\text{known}} = \begin{bmatrix} -1 & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & 1 \end{bmatrix} \tag{3.33}$$

(*Note:* The zero elements are replaced by dots to make the non-zero elements stand out. This turns out to be useful because it makes it far easier to identify the structure of the matrices.) It should be no surprise here that the extreme sparsity of these incidence matrices is a property worth exploiting.

Since the product $\tilde{\mathbb{E}}^{(2,1)}_{\text{known}}\tilde{\mathbf{u}}^{(1)}_{\text{known}}$ is known, Equation (3.31) can be written as

$$\tilde{\mathbf{s}}^{(2)} = \tilde{\mathbb{E}}^{(2,1)}\tilde{\mathbf{u}}^{(1)} + \tilde{\mathbf{u}}^{(1)}_{\text{norm}} \tag{3.34}$$

Because we are going to solve the lid-driven cavity problem, there is no flux into or out of the domain. The vector $\tilde{\mathbf{u}}^{(1)}_{\text{norm}}$ is therefore equal to zero and we have

$$\tilde{\mathbf{s}}^{(2)} = \tilde{\mathbb{E}}^{(2,1)}\tilde{\mathbf{u}}^{(1)} \tag{3.35}$$

3.9.2 $\mathbb{E}^{(1,0)}$

The incidence matrix $\mathbb{E}^{(1,0)}$ maps an inner-oriented 0-cochain to an inner-oriented 1-cochain. Let us create a linear equation for each of the line segments $u_{(i,j)}$ in the inner-oriented grid, considering that the points $P_{(i,j)}$ are sink-like. The equations are given by

$$\begin{aligned} u_{1,1} &= -p_{1,1} + p_{2,1} \\ u_{2,1} &= -p_{2,1} + p_{3,1} \\ &\vdots \\ u_{2,3} &= -p_{2,3} + p_{3,3} \\ v_{1,1} &= -p_{1,1} + p_{1,2} \\ v_{2,1} &= -p_{2,1} + p_{2,2} \\ &\vdots \\ v_{3,2} &= -p_{3,2} + p_{3,3} \end{aligned} \tag{3.36}$$

Writing Equation (3.36) in matrix notation, we have

$$\mathbf{u}^{(1)} = \mathbb{E}^{(1,0)}\mathbf{p}^{(0)} \tag{3.37}$$

where

$$\mathbb{E}^{(1,0)} = \begin{bmatrix} -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot \\ -1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & 1 & \cdot \end{bmatrix} \tag{3.38}$$

It is important to recognize that

$$\mathbb{E}^{(1,0)} = -\left(\tilde{\mathbb{E}}^{(2,1)}\right)^T \tag{3.39}$$

because this is a shortcut can save computational effort.

3.9.3 $\mathbb{E}^{(2,1)}$

The incidence matrix $\mathbb{E}^{(2,1)}$ maps an inner-oriented 1-cochain to an inner-oriented 2-cochain. That is, it maps circulation along line segments to vorticity in the planes enclosed by those line segments. The derivation of $\mathbb{E}^{(2,1)}$ is straightforward: add the circulation along those line segments that share a common orientation with the plane and subtract the circulation along those line segments whose orientation opposes the orientation of the plane. Executing this procedure for all planes $\xi_{i,j}$, we have

$$\begin{aligned} \xi_{0,0} &= u_{0,0} - u_{0,1} - v_{0,0} + v_{1,0} \\ \xi_{1,0} &= u_{1,0} - u_{1,1} - v_{1,0} + v_{2,0} \\ &\vdots \\ \xi_{3,3} &= u_{3,3} - u_{3,4} - v_{3,3} + v_{4,3} \end{aligned} \tag{3.40}$$

which is in accordance with the formula

$$\xi_{i,j} = u_{i,j} - u_{i,j+1} - v_{i,j} + v_{i+1,j} \tag{3.41}$$

Equation (3.40) written in matrix notation yields

$$\xi^{(2)} = \mathbb{E}^{(2,1)}\mathbf{u}^{(1)} \tag{3.42}$$

The velocities adjacent to the boundary are again known because the boundary conditions of the problem are known. Splitting the incidence matrix $\mathbb{E}^{(2,1)}$ into a matrix of unknowns and into a matrix of knows yields

$$\xi^{(2)} = \mathbb{E}^{(2,1)}\mathbf{u}^{(1)} + \mathbb{E}^{(2,1)}_{\text{known}}\mathbf{u}^{(1)}_{\text{known}} \tag{3.43}$$

where

$$\mathbb{E}^{(2,1)} = \begin{bmatrix}
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
-1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
1 & \cdot & -1 & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & -1 & 1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix} \tag{3.44}$$

and

$$\mathbb{E}^{(2,1)}_{\text{known}} = \\
\begin{bmatrix}
1 & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & 1
\end{bmatrix} \tag{3.45}$$

The product $\mathbb{E}^{(2,1)}_{\text{known}} \mathbf{u}^{(1)}_{\text{known}}$ can be evaluated because both factors are known. Equation (3.43) reduces to

$$\xi^{(2)} = \mathbb{E}^{(2,1)} \mathbf{u}^{(1)} + \mathbf{u}^{(1)}_{\text{prescribed}} \tag{3.46}$$

3.9.4 $\tilde{\mathbb{E}}^{(1,0)}$

The incidence matrix $\tilde{\mathbb{E}}^{(1,0)}$ maps an outer-oriented 0-cochain to an outer-oriented 1-cochain. That is, it maps values of the stream function located at the discrete points of the mesh to flux through the line segments. Deriving $\tilde{\mathbb{E}}^{(1,0)}$ is again straightforward; each of the line segments is bounded by two points. The flux through a line is equal to the sum of the values of the stream function at the bounding points. Add values at points whose orientation is equal to that of the line and

subtract values at points whose orientation opposes the orientation of the line. Repeating this process for each of the line segments, we have

$$\begin{aligned}
\tilde{u}_{1,0} &= -\tilde{\psi}_{1,0} + \tilde{\psi}_{1,1} \\
\tilde{u}_{2,0} &= -\tilde{\psi}_{3,0} + \tilde{\psi}_{2,1} \\
&\vdots \\
\tilde{u}_{2,2} &= -\tilde{\psi}_{2,2} + \tilde{\psi}_{2,3} \\
\tilde{v}_{0,1} &= \tilde{\psi}_{0,1} - \tilde{\psi}_{1,1} \\
\tilde{v}_{1,1} &= \tilde{\psi}_{1,1} - \tilde{\psi}_{2,1} \\
&\vdots \\
\tilde{v}_{2,2} &= \tilde{\psi}_{2,2} - \tilde{\psi}_{3,2}
\end{aligned} \tag{3.47}$$

Expressing Equation (3.47) in matrix notation yields

$$\tilde{\mathbf{u}} = \tilde{\mathbb{E}}^{(1,0)} \tilde{\psi} \tag{3.48}$$

where

$$\tilde{\mathbb{E}}^{(1,0)} = \begin{bmatrix}
\cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & -1 & \cdot & \cdot & \cdot & 1 & \cdot \\
\cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
\cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & 1 & -1 & \cdot & \cdot & \cdot & \cdot & \cdot
\end{bmatrix} \tag{3.49}$$

Also take note that

$$\tilde{\mathbb{E}}^{(1,0)} = \left(\mathbb{E}^{(2,1)}\right)^T \tag{3.50}$$

which can again save computational effort.

3.9.5 $\mathbb{H}^{(\tilde{1},1)}$ and $\mathbb{H}^{(1,\tilde{1})}$

The Hodge matrix $\mathbb{H}^{(\tilde{1},1)}$ represents a linear map between between the flux through a line segment and the circulation along a line segment. Its inverse, $\mathbb{H}^{(1,\tilde{1})}$, represents a linear map between the circulation along a line segment and the flux through a line segment. The cirulation along a line segment is equal to the velocity along that line segment times the lenght of the line segment. Or, expressed as a function of flux, we have

$$\text{ciculation along } L_a = \underbrace{\frac{\text{mass flow through } L_b}{\text{length of } L_b}}_{\text{velocity}} \cdot \underbrace{\text{length of } L_a}_{\text{length}} \tag{3.51}$$

Let us look at a specific example. The circulation along the line segment $u_{1,1}$ in Figure 3.7 is given by

$$u_{2,1} = \frac{\tilde{u}_{2,0}}{\tilde{h}_0} h_2 = \frac{h_2}{\tilde{h}_0} \tilde{u}_{2,0} \tag{3.52}$$

In general, the circulation along the line segments $u_{i,j}$ and $v_{i,j}$ can be found in accordance with the formulas

$$u_{i,j} = \frac{h_i}{\tilde{h}_{j-1}} \tilde{u}_{i,j-1} \tag{3.53a}$$

and

$$v_{i,j} = \frac{h_j}{\tilde{h}_{i-1}} \tilde{v}_{i-1,j} \tag{3.53b}$$

Equations (3.53a) and (3.53b) expressed in matrix notation yield

$$\mathbf{u}^{(1)} = \mathbb{H}^{(1,\tilde{1})} \tilde{\mathbf{u}}^{(1)} \tag{3.54}$$

The matrix $\mathbb{H}^{(1,\tilde{1})}$ is a diagonal matrix since it represents a linear operation.

3.9.6 $\mathbb{H}^{(\tilde{0},2)}$ and $\mathbb{H}^{(2,\tilde{0})}$

The Hodge matrix $\mathbb{H}^{(\tilde{0},2)}$ maps the vorticity associated with an inner-oriented 2-cochain to the stream function associated with an outer-oriented 0-cochain. This amounts to simply deviding the inner-oriented 2-cochain by its area:

$$\tilde{\psi}_{i,j} = \left(h_i h_j\right)^{-1} \xi_{i,j} \tag{3.55}$$

Equation (3.55) written in matrix notation yields

$$\tilde{\psi}^{(0)} = \mathbb{H}^{(\tilde{0},2)} \xi^{(2)} \tag{3.56}$$

where $\mathbb{H}^{(\tilde{0},2)}$ is again a diagonal matrix. Its inverse, $\mathbb{H}^{(2,\tilde{0})}$, represents a linear map between an outer-oriented 0-cochain and an inner-oriented 2-cochain.

3.9.7 The Convective Term

The derivation of the convective term, $\mathbf{u}^{(1)} \times \xi^{(2)}$, is not quite straightforward. The convective term is an exterior product of a 1-cochain and a 2-cochain. If $\xi^{(2)}$ and $\mathbf{u}^{(1)}$ are given by

$$\xi^{(2)} = \xi \, dx \, dy \tag{3.57}$$

and

$$\mathbf{u}^{(1)} = u \frac{\partial}{\partial x} + v \frac{\partial}{\partial y} \tag{3.58}$$

respectively, then the exterior product yields

$$\mathbf{u}^{(1)} \times \xi^{(2)} = u\xi \, dy - v\xi \, dx \tag{3.59}$$

As an example, let us compute the convection through the line segment $u_{1,1}$. Line segment $u_{1,1}$ is shown in Figure 3.8 along with the adjacent lines and planes that are involved in the computation.

Because $u_{1,1}$ is a strictly horizontal line segment, we do not need to consider the horizontal component of the convection through this line segment. Computing the mean vertical velocity in the plane below the line $u_{1,1}$ and multiplying it by $\tilde{\psi}_{1,0}$, we have

$$-\frac{1}{2}\left(\frac{v_{1,0}}{h_0} + \frac{v_{2,0}}{h_0}\right)\tilde{\psi}_{1,0} \tag{3.60}$$

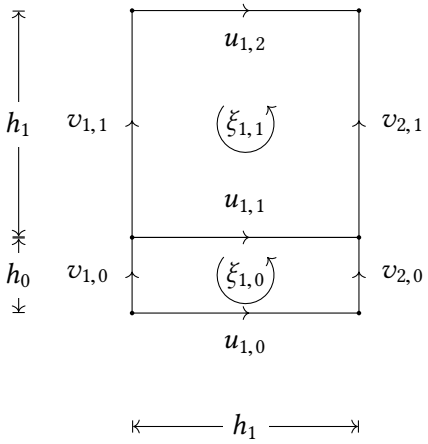


Figure 3.8: Convection through $u_{1,1}$.

For the plane above the line $u_{1,1}$, we have

$$-\frac{1}{2}\left(\frac{v_{1,1}}{h_1}+\frac{v_{2,1}}{h_1}\right)\tilde{\psi}_{1,1} \tag{3.61}$$

The average of Equations (3.60) and (3.61) multiplied by the length of $u_{1,1}$ yields the convection accross $u_{1,1}$:

$$\frac{1}{2}\left[-\frac{1}{2}\left(\frac{v_{1,0}}{h_0}+\frac{v_{2,0}}{h_0}\right)\tilde{\psi}_{1,0}-\frac{1}{2}\left(\frac{v_{1,1}}{h_1}+\frac{v_{2,1}}{h_1}\right)\tilde{\psi}_{1,1}\right]h_1 \tag{3.62}$$

or

$$-\frac{h_1}{4h_0}\left(v_{1,0}+v_{2,0}\right)\tilde{\psi}_{1,0}-\frac{h_1}{4h_1}\left(v_{1,1}+v_{2,1}\right)\tilde{\psi}_{1,1} \tag{3.63}$$

The final multipliplication by h_1, the length of the line segment $u_{1,1}$, is neseccary because the momentum equation is a 1-form equation. That is, all terms of the momentum equation must ultimately be expressed as inner-oriented 1-cochains. Repetition of the above procedure for all line segments $u_{i,j}$ and $v_{i,j}$ yields

$$\text{convection} = \begin{bmatrix}
-\frac{\tilde{h}_1}{4\tilde{h}_0}\left(v_{1,0}+v_{2,0}\right)\psi_{1,0}-\frac{\tilde{h}_1}{4\tilde{h}_1}\left(v_{1,1}+v_{2,1}\right)\psi_{1,1}\\
-\frac{\tilde{h}_2}{4\tilde{h}_0}\left(v_{2,0}+v_{3,0}\right)\psi_{2,0}-\frac{\tilde{h}_2}{4\tilde{h}_1}\left(v_{2,1}+v_{3,1}\right)\psi_{2,1}\\
-\frac{\tilde{h}_1}{4\tilde{h}_1}\left(v_{1,1}+v_{2,1}\right)\psi_{1,1}-\frac{\tilde{h}_1}{4\tilde{h}_2}\left(v_{1,2}+v_{2,2}\right)\psi_{1,2}\\
-\frac{\tilde{h}_2}{4\tilde{h}_1}\left(v_{2,1}+v_{3,1}\right)\psi_{2,1}-\frac{\tilde{h}_2}{4\tilde{h}_2}\left(v_{2,2}+v_{3,2}\right)\psi_{2,2}\\
-\frac{\tilde{h}_1}{4\tilde{h}_2}\left(v_{1,2}+v_{2,2}\right)\psi_{1,2}-\frac{\tilde{h}_1}{4\tilde{h}_3}\left(v_{1,3}+v_{2,3}\right)\psi_{1,3}\\
-\frac{\tilde{h}_2}{4\tilde{h}_2}\left(v_{2,2}+v_{3,2}\right)\psi_{2,2}-\frac{\tilde{h}_2}{4\tilde{h}_3}\left(v_{2,3}+v_{3,3}\right)\psi_{2,3}\\
\frac{\tilde{h}_0}{4\tilde{h}_0}\left(u_{0,1}+u_{0,2}\right)\psi_{0,1}+\frac{\tilde{h}_0}{4\tilde{h}_1}\left(u_{1,1}+u_{1,2}\right)\psi_{1,1}\\
\frac{\tilde{h}_1}{4\tilde{h}_1}\left(u_{1,1}+u_{1,2}\right)\psi_{1,1}+\frac{\tilde{h}_1}{4\tilde{h}_2}\left(u_{2,1}+u_{2,2}\right)\psi_{2,1}\\
\frac{\tilde{h}_2}{4\tilde{h}_2}\left(u_{2,1}+u_{2,2}\right)\psi_{2,1}+\frac{\tilde{h}_2}{4\tilde{h}_3}\left(u_{3,1}+u_{3,2}\right)\psi_{3,1}\\
\frac{\tilde{h}_0}{4\tilde{h}_0}\left(u_{0,2}+u_{0,3}\right)\psi_{0,2}+\frac{\tilde{h}_0}{4\tilde{h}_1}\left(u_{1,2}+u_{1,3}\right)\psi_{1,2}\\
\frac{\tilde{h}_1}{4\tilde{h}_1}\left(u_{1,2}+u_{1,3}\right)\psi_{1,2}+\frac{\tilde{h}_1}{4\tilde{h}_2}\left(u_{2,2}+u_{2,3}\right)\psi_{2,2}\\
\frac{\tilde{h}_2}{4\tilde{h}_2}\left(u_{2,2}+u_{2,3}\right)\psi_{2,2}+\frac{\tilde{h}_2}{4\tilde{h}_3}\left(u_{3,2}+u_{3,3}\right)\psi_{3,2}
\end{bmatrix} \tag{3.64}$$

This representation of convection is all but exact. Each time you average quantities, you introduce a certain degree of error and this representation of the convective term involves not one, but *two* averages.

3.10 Time Marching

The Navier-Stokes equations were rewritten in terms of incidence matrices and Hodge matrices, as follows:

$$\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u} + \mathbf{u}_{\text{norm}} = 0 \tag{3.65a}$$

$$\xi = \mathbb{E}^{(2,1)}\mathbf{u} \tag{3.65b}$$

$$\frac{\partial \mathbf{u}}{\partial t} + \text{convective} + \mathbb{E}^{(1,0)}P + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u} + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}} = 0 \tag{3.65c}$$

where $\mathbf{u}_{\text{norm}} = 0$ in the case of the lid-driven cavity flow problem.

We will be using the forward Euler method, an explicit method, to advance the solution in time. Suppose we have a first order differential equation given by

$$\frac{d\mathbf{u}}{dt} + f(\mathbf{u}) = 0 \tag{3.66}$$

Replacing the time derivative by the forward Euler scheme, we have

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + f(\mathbf{u}^n) = 0 \tag{3.67}$$

where n denotes a certain point in time and Δt denotes the timestep. Equation (3.67) can be rearranged such that the solution at time $n+1$ is expressed as a function of the solution at time n:

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t\ f(\mathbf{u}^n) \tag{3.68}$$

This method is a first-order method because it produces an error of $O(\Delta t)$. Time-stepping using the forward Euler method is as simple as it gets, but its drawback is that it requires extremely small values of Δt to be numerially stable. In spite of this major disadvantage, let us go ahead and discretize the time derivative in Equation (3.65c) using a forward Euler scheme. It will be trivial to replace the time-stepping method by a higher-order method at a later stage. Replacing the time derivative in Equation (3.65c) by a forward Euler scheme yields

$$\frac{\mathbf{u}^{n+1} - \mathbf{u}^n}{\Delta t} + \text{convective}^n + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^n + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}} = 0 \tag{3.69}$$

Why does P have $n+1$ in its superscript and not just n? The pressure at $t = 0$ is not required because P^0 does not appear in the equations. This means that P must be associated with the next timestep. Multiplication of Equation (3.69) by Δt gives

$$\mathbf{u}^{n+1} - \mathbf{u}^n + \Delta t\left(\text{convective}^n + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^n + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\right) = 0 \tag{3.70}$$

Multiplying all terms in Equation (3.70) by $\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}$, we have

$$\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n+1} - \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^n + \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\Delta t\left(\text{convective}^n + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^n + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\right) = 0 \tag{3.71}$$

Equating Equation (3.71) to the continuity equation yields

$$\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n+1} - \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n} + \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\Delta t\Bigg(\text{convective}^{n} + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^{n} + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\Bigg) = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n+1} + \mathbf{u}_{\text{norm}} \tag{3.72}$$

(*Note:* It is perfectly acceptable to write the continuity equation as $\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n+1}$. The superscript of $\mathbf{u}$ in the continuity equation can be chosen freely because the continuity equation holds true at *any* given time.) The $\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n+1}$ term can now be removed from both sides of Equation (3.72). Doing so gives

$$-\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbf{u}^{n} + \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\Delta t\Bigg(\text{convective}^{n} + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^{n} + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\Bigg) = \mathbf{u}_{\text{norm}} \tag{3.73}$$

After some simple algebraic rearragenement of Equation (3.73), we obtain

$$\tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbb{E}^{(1,0)}P^{n+1} = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\Bigg(\frac{\mathbf{u}^{n}}{\Delta t} - \text{convective}^{n} - \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^{n} - \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\Bigg) + \frac{\mathbf{u}_{\text{norm}}}{\Delta t} \tag{3.74}$$

which is equivalant to

$$A\mathbf{P}^{n+1} = f \tag{3.75}$$

where

$$A = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbb{E}^{(1,0)} \tag{3.76}$$

and

$$f = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\Bigg(\frac{\mathbf{u}^{n}}{\Delta t} - \text{convective}^{n} - \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^{n} - \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\Bigg) + \frac{\mathbf{u}_{\text{norm}}}{\Delta t} \tag{3.77}$$

(*Note:* $\mathbf{P}^{n+1}$ in Equation (3.75) is written in boldface because the numerical values of P are stored in an array (i.e., a vector). However, the physical quantity P remains a scalar quantity and is *not* a vector quantity.) Once the system $A\mathbf{P}^{n+1} = f$ has been solved, P^{n+1} can be substituted into

$$\mathbf{u}^{n+1} = \mathbf{u}^{n} - \Delta t\Bigg(\text{convective}^{n} + \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^{n} + \frac{1}{\text{Re}}\mathbf{u}_{\text{prescribed}}\Bigg) \tag{3.78}$$

to compute the solution at time $n + 1$. Note that Equation (3.78) is the result of a simple algebraic rearrangement of Equation (3.70).

4 Code

In Chapter 2 we derived a system of dimensionless partial differential equations that govern the motion of viscous fluids and in Chapter 3 we presented a framework that enables a computer to solve these equations. In this chapter, we will look at the implementation of this framework in `Python` and `C`. As is the case with most numerical simulations, the program can take a long time to run if reasonable accuracy is desired. Thankfully, there is vast potential for speedup if common sense and simple optimization techniques are applied. For instance, simply rearranging parts of the code results in a speedup factor of more than 10 without a loss of accuracy.

4.1 The Simulation Loop

After generating the mesh and all incidence and hodge matrices, an implementation of the procedure described in the preceding chapter is in essence a matter of looping the following five steps:

1. Generate the convective term.
2. Construct the system $A\mathbf{P}^{n+1} = f$ where

$$A = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\mathbb{E}^{(1,0)} \tag{4.1}$$

   and

$$f = \tilde{\mathbb{E}}^{(2,1)}\mathbb{H}^{(\tilde{1},1)}\left(\frac{\mathbf{u}^n}{\Delta t} - \text{convective}^n - \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^n - \frac{1}{\text{Re}}\mathbf{u}_{\text{pres}}\right) + \frac{\mathbf{u}_{\text{norm}}}{\Delta t} \tag{4.2}$$

3. Solve the system for $\mathbf{P}^{n+1}$.
4. Advance the solution in time using

$$\mathbf{u}^{n+1} = \mathbf{u}^n - \Delta t\left(\text{convective}^n - \mathbb{E}^{(1,0)}P^{n+1} + \frac{1}{\text{Re}}\mathbb{H}^{(1,\tilde{1})}\tilde{\mathbb{E}}^{(1,0)}\mathbb{H}^{(\tilde{0},2)}\mathbb{E}^{(2,1)}\mathbf{u}^n + \frac{1}{\text{Re}}\mathbf{u}_{\text{pres}}\right) \tag{4.3}$$

5. Check whether or not the solution has converged to within a satisfactory tolerance and if not, then go back to the first step. Continue this loop until the solution has converged.

4.2 Baseline Implementation

Listing 4.1 shows a naive implementation of the above five steps. It is essentially a one-to-one translation of the mathematical equations without any sort of consideration for performance. Note that this excerpt of the program is neither a working program nor written in a real programming language. It is simply a description of the simulation loop using some elements of `Python` and `Matlab` for easy comprehension. The vast majority of the remainder of the code (that is, all 500 lines that are not shown here) consists of helper functions to generate the incidence and hodge matrices.

Listing 4.1: Code excerpt of a naive implementation.

```
while diff > tol:
    xi = Ht02 * E21 * u
    convective = generate_convective(xi)

    A = tE21 * Ht11 * E10
    f = tE21 * Ht11 * (u/dt - H1t1 * tE10 * Ht02 *
       E21 * u/Re - u_pres/Re - convective)
    P = solve(A, f)

    u_old = u
    u = u - dt * (E10 * P + H1t1 * tE10 * Ht02 *
       E21 * u/Re + u_pres/Re + convective)

    diff = max(abs(u - u_old)) / dt
```

The code in Listing 4.1 produces correct results but it is called a *naive* implementation for a good reason: it is painfully slow.

4.3 Optimized Implementation

The code in Listing 4.1 can be optimized to run faster and consume less memory. We will explore some of these optimizations in this section. Let us begin by assuming that all matrices are *dense* matrices. That is, all zero elements are stored in memory. The use of sparse matrix storage schemes is one of the optimizations that we will discuss.

4.3.1 Common Sense Optimizations

It may sound a bit strange, but one of the most elementary optimization techniques is to simply do *less work*. The code in Listing 4.1 contains ten matrix-matrix multiplications of $O(N^3)$, five matrix-vector multiplications of $O(N^2)$, and one matrix solve of $O(N^3)$. The code can be rearranged in such a way that the loop only contains four matrix-vector multiplications of $O(N^2)$ and one matrix solve of $O(N^3)$. That is, *all* matrix-matrix multiplications, which are by far the most expensive operations, can be avoided in their entirety.

It should not come as a surprise that such an optimized loop is *much* faster than the naive implementation above. To be specific, the factor of speedup with respect to the naive implementation for $N = 16$ and $\Delta t = 0.05$ is around 12. That means, in practical terms, that a simulation of an hour reduces to merely five minutes. Let us look at where the code can be rearranged to yield better performance:

1. The pressure matrix `A` does not change between loop iterations. The code on line 5, `A = tE21 * Ht11 * E10`, can therefore be moved out of the loop, saving two matrix-matrix multiplications per loop iteration.
2. In a similar vein, the matrix-matrix multiplication `Ht02 * E21` on line 2 can be taken out of the loop and its result can be stored in a constant `C0`. This saves one matrix-matrix multiplication per loop iteration.
3. The matrix-matrix multiplication `tE21 * Ht11` on line 6 can be taken out of the loop and its result can be stored in a constant `C1`. This again saves one matrix-matrix multiplication per loop iteration.
4. The product `H1t1 * tE10 * Ht02 * E21` occurs on line 6 and line 10 and can be moved out of the loop and can be stored in a constant `C2`. This saves six matrix-matrix multiplication per loop iteration.
5. The devision `u_pres/Re` occurs on line 6 and line 10 and can be moved out of the loop and its result can be stored in a constant `C3`. The savings of this step are negligible, but it is good practice to do it anyway.
6. The preceding five rearrangements result in the code `C2 * u/Re + C3 + convective` occurring on line 6 and line 10. Instead of having this code occur twice, it can be stored in a variable `C4` that is updated once per loop iteration. This saves one matrix-vector multiplication per loop iteration.

The six rearrangement suggested in the above list result in the simulation loop shown in Listing 4.2.

Listing 4.2: Code excerpt rearranged for improved performance.

```
1  C0 = Ht02 * E21
2  C1 = tE21 * Ht11
3  C2 = H1t1 * tE10 * C0
4  C3 = u_pres/Re
5
6  A = C1 * E10
7
8  while diff > tol:
9      xi = C0 * u
10     convective = generate_convective(xi)
11
12     C4 = C2 * (u/Re) + C3 + convective
13     f = C1 * (u/dt - C4)
14     P = solve(A, f)
15
16     u_old = u
17     u = u - dt * (E10 * P + C4)
18
19     diff = max(abs(u - u_old)) / dt
```

4.3.2 Advanced Optimizations

The code in listing 4.2 is still of $O(N^3)$ because it contains a matrix solve of $O(N^3)$. That is, assuming the underlying implementation of the `solve(A, f)` function uses a direct solution method like LU factorization. Without going into too much detail, let us take a quick look at LU factorization. The first step of LU factorization is factorizing the matrix of coefficients A into the product of a lower triangular matrix L and an upper triangular matrix U, hence the name LU factorization. The system $A\mathbf{P} = \mathbf{f}$ then becomes

$$LU\mathbf{P} = \mathbf{b} \tag{4.4}$$

The system in Equation (4.4) can then be solved in two stages; first solving

$$L\mathbf{x} = \mathbf{f} \tag{4.5}$$

and then solving

$$U\mathbf{P} = \mathbf{x} \tag{4.6}$$

Solving the systems in Equations (4.5) and (4.6) is trivial because L and U are triangular matrices. Factorizing the system $A\mathbf{P} = \mathbf{f}$ into Equation (4.4) is called the factorization stage and is an operation of order $O(N^3)$. Solving the systems in Equations (4.5) and (4.6) is called the solve stage and is an operation of order $O(N^2)$.

Just like it does not make sense to recompute the pressure matrix A in every iteration of the simulation loop, it does also not make much sense to redo the factorization stage in every iteration of the loop. After all, if the matrix A does not change, then the matrices L and U cannot change either. It would be more efficient to move the factorization state out of the loop and to keep the solve stage inside the loop. Doing so reduces the order of the loop from $O(N^3)$ to $O(N^2)$ because all there is left inside the loop are matrix-vector multiplications and the solve stage, both of which are of $O(N^2)$. Listing 4.3 shows the simulation loop incorporating this improvement.

Listing 4.3: Code excerpt after moving the factorization stage.

```
1  C0 = Ht02 * E21
2  C1 = tE21 * Ht11
3  C2 = H1t1 * tE10 * C0
4  C3 = u_pres / Re
5
6  A = C1 * E10
7  LU = lu_factor(A)
8
9  while diff > tol:
10     xi = C0 * u
11     convective = generate_convective(xi)
12
13     C4 = C2 * (u/Re) + C3 + convective
14     f = C1 * (u/dt - C4)
15     P = lu_solve(LU, f)
16
17     u_old = u
18     u = u - dt * (E10 * P + C4)
19
```

```
20      diff = max(abs(u - u_old)) / dt
```

Further optimizing the code starts to get substantially harder from here. There are several things that I have tried, with varying degrees of success.

Using ATLAS as a Replacement for BLAS

The numpy library is essentially a wrapper around the BLAS (Basic Linear Algebra Subprograms) and LAPACK libraries. Most implementations of BLAS are open source. Therefore, BLAS comes in many flavors and some are more performant than others. I replaced my particular BLAS library with ATLAS (Automatically Tuned Linear Algebra Software), which is fully multithreaded and must be compiled from source so that it can automatically tune itself for the system that it is being compiled on. Using ATLAS over the default BLAS library resulted in a large performance increase.

A Native C Implementation

Using numpy is fast, most of the time just as fast as calling BLAS routines natively from C, because most of its functions call compiled routines. However, certain functions like `generate_convective()` are still inherently slow because they are implemented in the Python scripting language.

Calling ATLAS subroutines natively from C allows you to combine mathematical operations that you could otherwise not combine when using numpy. For instance, the level 2 BLAS function `SGEMV()` computes

$$\mathbf{y} \coloneqq \alpha A\mathbf{x} + \beta\mathbf{y} \tag{4.7}$$

with one function call. This allows you to replace line 13 in Listing 4.3 with one call of `SGEMV()`, where $\alpha = 1/\mathrm{Re}$, $A = \mathtt{C2}$, $\mathbf{x} = \mathbf{u}$, $\mathbf{y} = \mathtt{C3} + \mathrm{convective}$, and $\beta = 1$.

I decided to write a second implementation of the simulation in C for two reasons: first, to have compiled and therefore inherently fast versions of all functions, including the function `generate_convective()`, and second, because it allows you to call ATLAS routines directly and use it to combine certain mathematical operations like matrix-vector multiplications and vector-vector additions.

Computing in Single-Precision Instead of Double-Precision

Using single-precision floating-point numbers instead of double-precision floating-point numbers allows the CPU to store more numbers in its cache, thereby improving the cache hit ratio and reuse ratio which in turn improves overal performance. Most of the errors are introduced by using first-order time-stepping and a low fidelity approximation of the convective term. I therefore deemed it safe to use single-precision floating-point numbers instead of double-precision floating-point numbers.

Sparse Matrix-Vector Multiplication

One of the advantages of this method based on DEC is that the incidence and hodge matrices are highly sparse. Sparse matrix-vector multiplication is *very* fast, unlike sparse matrix-matrix multiplication which is heavily memory bound. I converted the incidence and hodge matrices in the `Python` implementation to a sparse format to see what improvements it would bring. The improvements were marginal because it turned out that sparse-matrix vector multiplication only used a single CPU core. There does not seem to be a trivial way to use multithreaded sparse matrix operations, neither in `Python` nor in `C`. I discarded the idea of using sparse matrix operations because it is not worth the time to include an exotic library, let alone implement such a library myself, to speedup an already reasonably fast code. However, I am convinced that sparse matrix operations are absolutely necessary when writing a 3D or high fidelity solver using DEC.

Higher-Order Time-Stepping Methods

In addition to making the code run faster, it is also not a bad idea to use mathematics to make the problem converge faster. One obvious way to increase the rate of convergence is to replace the first-order time-stepping method by a higher-order time-stepping method. I replaced the forward Euler method of $O(\Delta t)$ by the modified Euler method, which is a Runge-Kutta method of $O(\Delta t^2)$. This method resulted in much faster convergence during the initial 100 iterations, but ended up producing literally the same results in subsequent iterations. So the net effect of using the modified Euler method was nil. This implies that there must be some other source of error producing an error larger than $O(\Delta t)$, most likely the convective term.

5 Results

The aim of this chapter is to present a comparison of the present method with the benchmark results from [1]. We will present contour plots of the stream function, vorticity, and static pressure and discuss any differences with the benchmark results. Additionally, the factors of speedup resulting from the optimizations suggested in Chapter 4 will be presented and discussed.

5.1 Convergence, Timestep, and Tolerance

The program requires four arguments:

1. The stopping criterion.
2. The Reynolds number.
3. The timestep Δt.
4. The number of planes in the horizontal and vertical direction N.

Each of these four arguments has a major impact on the outcome of the simulation and must therefore be carefully considered.

The simulation will halt when

$$\frac{\max\left(\left|\mathbf{u}^n - \mathbf{u}^{n-1}\right|\right)}{\Delta t} < \text{tol} \tag{5.1}$$

where tol is an abbreviation for tolerance. A tolerance of $1 \cdot 10^{-5}$ turned out to be a reasonable compromise between accuracy and execution time. The Reynolds number was fixed at 1000 to match the benchmark results. The number of planes in the horizontal and vertical direction N was increased from 16 to 64 with three intermediate steps. The timestep Δt is highly dependent on N; in certain numerical methods, it is crucial to choose the timestep such that the flow does not cover a distance larger than the smallest mesh spacing within Δt. This is known as the Courant–Friedrichs–Lewy (CFL) condition:

$$C = \frac{u\Delta t}{\Delta x} \leq C_{\max} \tag{5.2}$$

where $C_{\max}$ is typically equal to 1, so

$$\Delta t \leq \frac{C_{\max}\Delta x}{u} \tag{5.3}$$

While the CFL condition does not nessecarily in this case, it can at least serve as a guideline. The CFL condition turned out to hold within an order of magnitude and the final value of Δt was determined by means of trial and error. A timestep that is too large causes the solution to diverge and a timestep that is too small results in unnecessarily slow convergence. The optimal timestep is a value just below the

#	N	Δt	tolerance	Re	iterations	iterations $\cdot \Delta t$
1	16	0.05	$1 \cdot 10^{-5}$	1000	1379	68.95
2	32	0.004	$1 \cdot 10^{-5}$	1000	12002	48.01
3	48	0.0007	$1 \cdot 10^{-5}$	1000	56577	39.60
4	56	0.0004	$1 \cdot 10^{-5}$	1000	93111	37.24
5	64	0.0002	$1 \cdot 10^{-5}$	1000	?	?

Table 5.1: Settings for different runs of the simulation.

value that would cause the solution to diverge. The parameters used in the creation of the final results are tabulated in Table 5.1.

It is interesting that the actual simulated time iterations $\cdot \Delta t$ (see the rightmost column of Table 5.1) converges to around 35 (omitting the word "seconds" here because the time is dimensionless). The more detailed your simulation, the closer the approximation of transport of momentum in the fluid. The simulation converges once the flow field is rotates steadily, which requires transport of momentum from the boundaries to the center of the domain.

5.2 Stream Function

The stream function $\tilde{\psi}$ is defined such that the difference $\Delta\tilde{\psi}$ between two streamlines is equal to the mass flux between two streamlines. This fact allows us to compute the stream function based on the values of flux along the line segments. The outer-oriented line segments $\tilde{u}_{i,j}$ and $\tilde{v}_{i,j}$ represent flux in the form:

$$\tilde{u}_{i,j} = \int_{y_{i,j}}^{y_{i,j+1}} \mathbf{u} \cdot \hat{\mathbf{n}}\, dy \tag{5.4a}$$

and

$$\tilde{v}_{i,j} = \int_{x_{i,j}}^{x_{i+1,j}} \mathbf{u} \cdot \hat{\mathbf{n}}\, dx \tag{5.4b}$$

where $\hat{\mathbf{n}}$ denotes the normal vector along the line segment. Thus, computing the stream function is a simply matter of assuming that the stream function has a value of zero at the boundaries and using either Equation (5.4a) or Equation (5.4b) to compute the unknowns in the remainder of the domain.

Figure 5.1 shows an interpolated contour plot of the stream function for $N = 64$, Re $= 1000$, $\Delta t = 0.0002$, and tol $= 1 \cdot 10^{-5}$. Figure 5.2 shows the benchmark result from [1]. The contour levels associated with the labels a–j in Figure 5.2 can be found in Table 7 of [1]. The two figures are, for all intents and purposes, identical. Figure 5.7 shows plots of the stream function for $N = 16$, $N = 32$, $N = 48$, $N = 56$, and $N = 64$ without interpolation as well as the benchmark result.

5.3 Vorticity

Plotting the vorticity requires no post-processing. Figure 5.3 shows a contour plot of the stream function for $N = 64$, Re $= 1000$, $\Delta t = 0.0002$, and tol $= 1 \cdot 10^{-5}$. Figure 5.4 shows the benchmark result from [1]. The contour levels associated with the labels a–k in Figure 5.4 can be found in Table 8 of [1].

The swirl in the center of the velocity plot (line *d* in the benchmark result) appears to be different from the benchmark. Aside of the center swirl, the two images are practically identical. The difference is most likely caused by differences in mesh density and tolerance. For example, the benchmarks were created with $N = 128$ and the tolerance may have been set to a smaller number than during the creation of these results. Figure 5.8 shows vorticity plots for $N = 16$, $N = 32$, $N = 48$, $N = 56$, and $N = 64$ without interpolation as well as the benchmark result. These plots support the argument that the difference in mesh density N is likely the reason for the difference with the benchmark results; the center swirl is affected most when adjusting N.

5.4 Pressure

Computing the pressure does require some additional processing. We are interested in the static pressure p rather than the total pressure P. Thus,

$$p = P - \frac{1}{2} \|\mathbf{u}\|^2 \tag{5.5}$$

The velocities at the line segments can be found by dividing the circulation by the length of the respective line segments and taking the average of the velocities around $P_{i,j}$. We therefore have

$$\|\mathbf{u}\|_{i,j} = \sqrt{\left[\frac{1}{2}\left(\frac{u_{i-1,j}}{h_{i-1}} + \frac{u_{i,j}}{h_i}\right)\right]^2 + \left[\frac{1}{2}\left(\frac{v_{i,j-1}}{h_{j-1}} + \frac{v_{i,j}}{h_j}\right)\right]^2} \tag{5.6}$$

and the static pressure p can be determined up to a constant. The pressure can only be determined up to a constant because the flow is governed by differences in *static* pressure; the total pressure is irrelevant. To reproduce the plot in [1], the static pressure must be scaled such that it equals zero at the center of the domain.

Figure 5.5 shows a contour plot of the stream function for $N = 64$, $\mathrm{Re} = 1000$, $\Delta t = 0.0002$, and $\mathrm{tol} = 1 \cdot 10^{-5}$. Figure 5.6 shows the benchmark result from [1]. The contour levels associated with the labels *a*–*j* in Figure 5.6 can be found in Table 8 of [1]. The static pressure plot shows the same overal pattern as the benchmark results, but the lines are slightly displaced. The difference is causes by the fact that the mesh densities are different ($N = 128$ versus $N = 64$) and the constant that is subtracted from the static pressure may have been different. Figure 5.9 shows plots of the stream function for $N = 16$, $N = 32$, $N = 48$, $N = 56$, and $N = 64$ without interpolation as well as the benchmark result. The match with the benchmark results is rather poor for low values of N.

5.5 Code Optimizations

The results of the optimizations suggested in Chapter 4 are tabulated in Table 5.2. The baseline version of the program is the skeleton code supplied with this assignment. The largest single cumulative factor of speedup with respect to the baseline version is 343.89. To put that into perspective, a speedup of 343.89 makes a simulation that takes one week to run finish in just under half an hour.

The largest single relative factor of speedup is 23.42. This is quite a remarkable factor of speedup given that it is merely the result of simply rearranging parts of the code. This proves yet again that the principle of "do less work" along with using common sense goes a long way in optimizing slow pieces of code.

#	Version	Execution time	Speedup (rel.)	Speedup (cum.)
1	Baseline	3 h 37 m 48 s	1.00	1.00
2	Rearranged code	9 m 18 s	23.42	23.42
3	No factorization stage	3 m 11 s	2.92	68.42
4	Sparse matrix storage	1 m 55 s	1.66	113.63
5	Double-precision in `C`	1 m 7 s	1.72	195.04
6	Single-precision in `C`	38 s	1.76	343.89

Table 5.2: Execution time and factors of speedup for $N = 32$, $\mathrm{Re} = 1000$, $\Delta t = 0.004$, and $\mathrm{tol} = 1 \cdot 10^{-5}$ on an Intel Core i5-2400 with four CPU cores clocked at 3100 MHz (CPU throttling and turbo disabled).

5.6 Integrated Vorticity

The integrated vorticity is more or less constant, regardless of the value of N used to run the simulation. The integrated viscosity grows quadratically as a function of the Reynolds number. Here is why: for a general fluid, vorticity is twice the mean angular velocity and is therefore related to angular momentum. Angular momentum can neither be created nor destroyed and must be constant in a *closed* system. However, the lid-driven cavity is *not* a closed system because energy is continuously added to the system by the moving boundary. At $t = 0$, the integrated angular momentum of the system is zero. The moving boundary adds energy (and therefore angular momentum) to the system until the viscous dissipation is able to dissipate the energy that is added to the system by the boundary. This is when an equilibrium settles in and the system rotates steadily (aside of unsteady fluctuations). A high Reynolds number implies that the fluid is viscous and that viscous dissipation occurs at small length scales. As a result, a high Reynolds number allows the system to build up more angular momentum before the viscous dissipation can provide a sink for the energy that is added to the system.

5.7 Turbulence

One question I have been asking myself ever since starting this assignment is: why does the simulation converge? After all, we are simulating the unsteady viscous Navier-Stokes equations and not some time-averaged equations. The simulation converges because the Reynolds number is very low and the flow is entirely laminar. A low Reynolds number implies that the fluid is relatively "sluggish", that the viscous dissipation takes place at relatively large scales, and that small unsteady vortices are suppressed. Trial runs with a far larger Reynolds number show that the rate of convergence drops dramatically (that is, the solution does not converge) and starts fluctuating wildly. The wild fluctuation can be attributed to the large and small vortices that are continuously being created and destroyed.

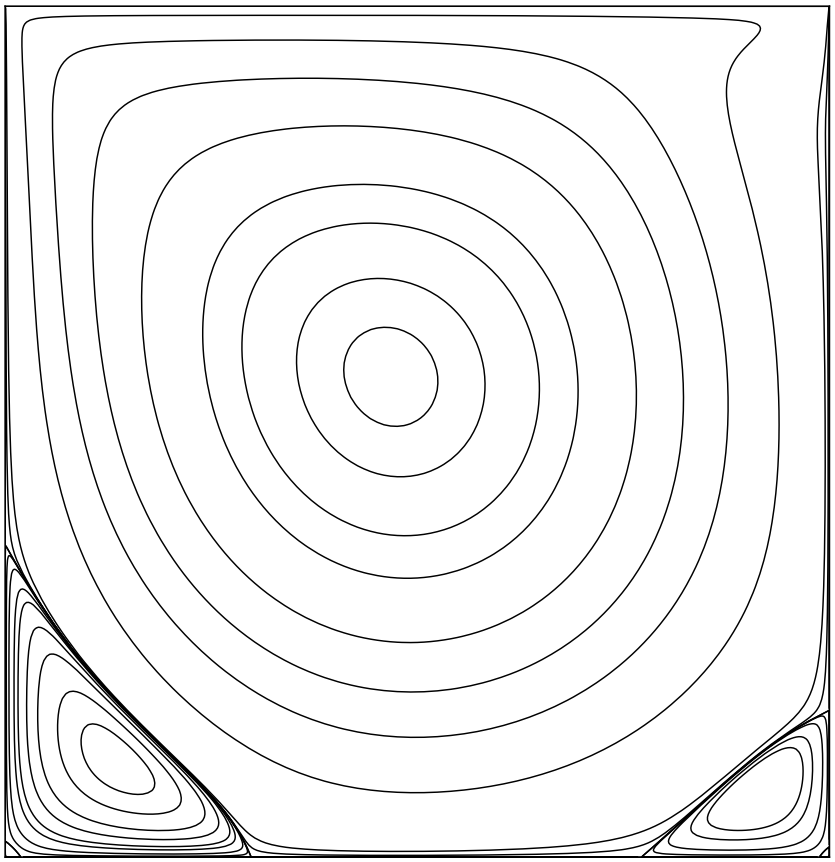


Figure 5.1: Stream function for $N = 64$, Re $= 1000$, $\Delta t = 0.0002$, and tol $= 1 \cdot 10^{-5}$.

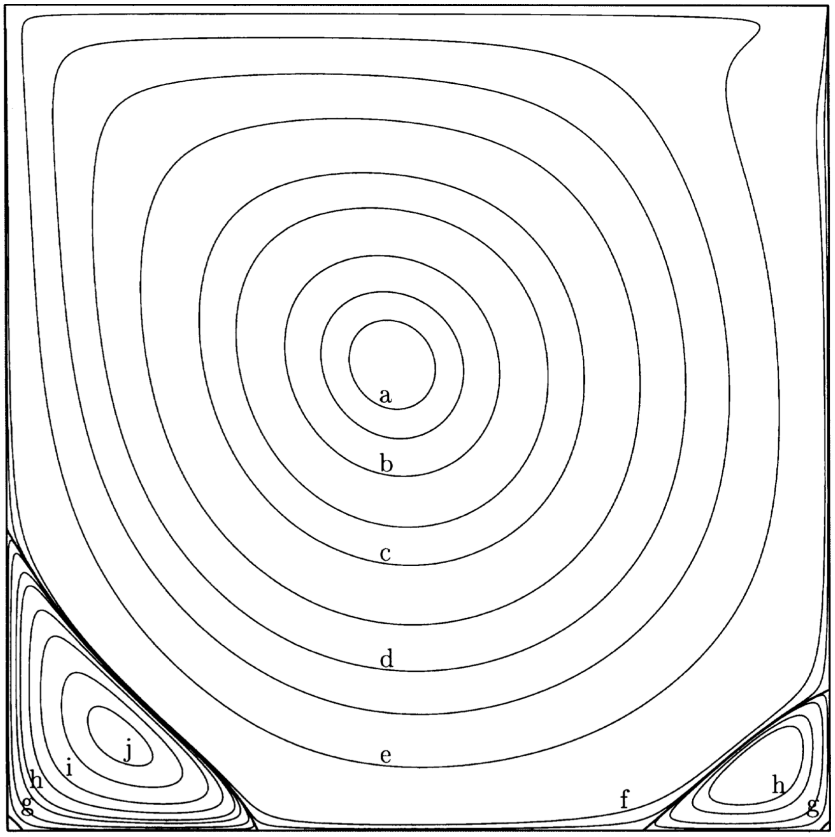


Figure 5.2: Benchmark stream function for Re $= 1000$ [1].

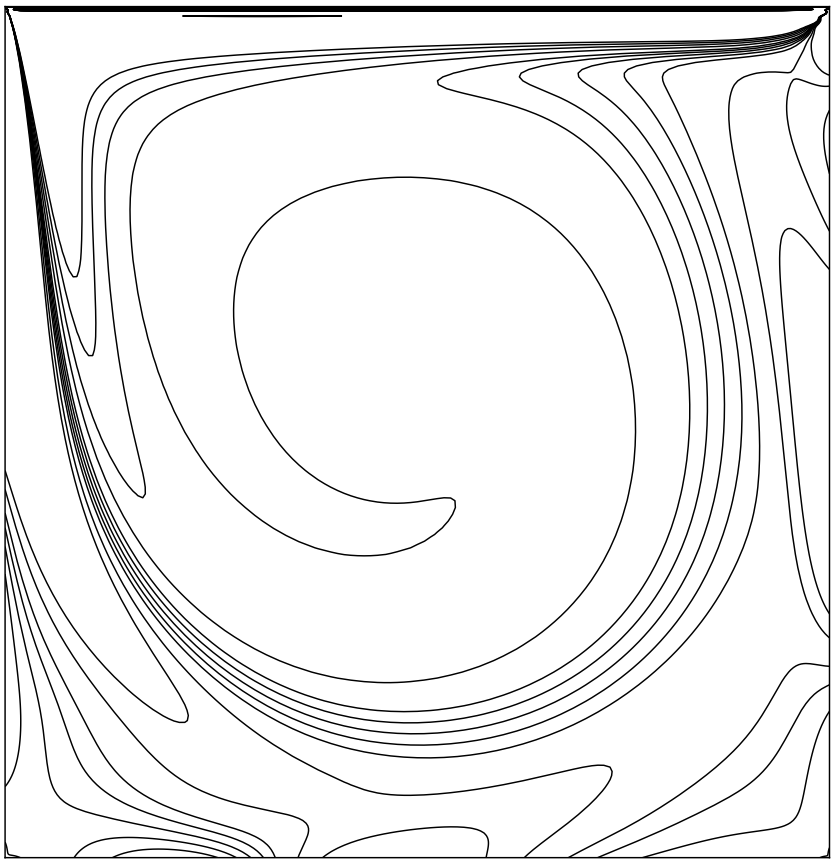


Figure 5.3: Vorticity for $N = 64$, Re $= 1000$, $\Delta t = 0.0002$, and tol $= 1 \cdot 10^{-5}$.

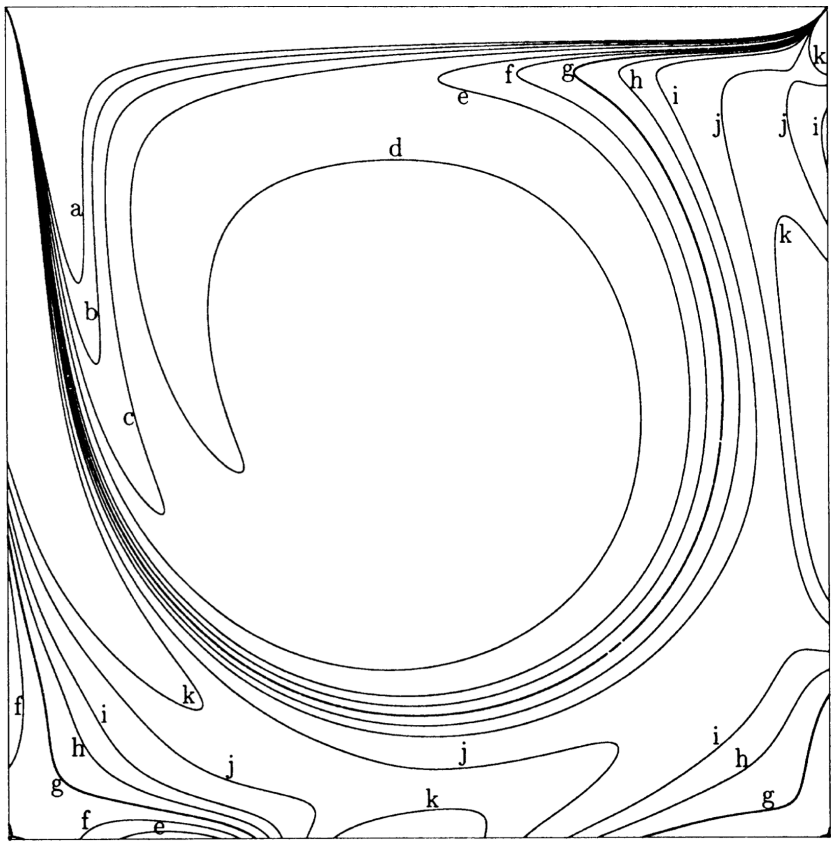


Figure 5.4: Benchmark vorticity for Re $= 1000$ [1].

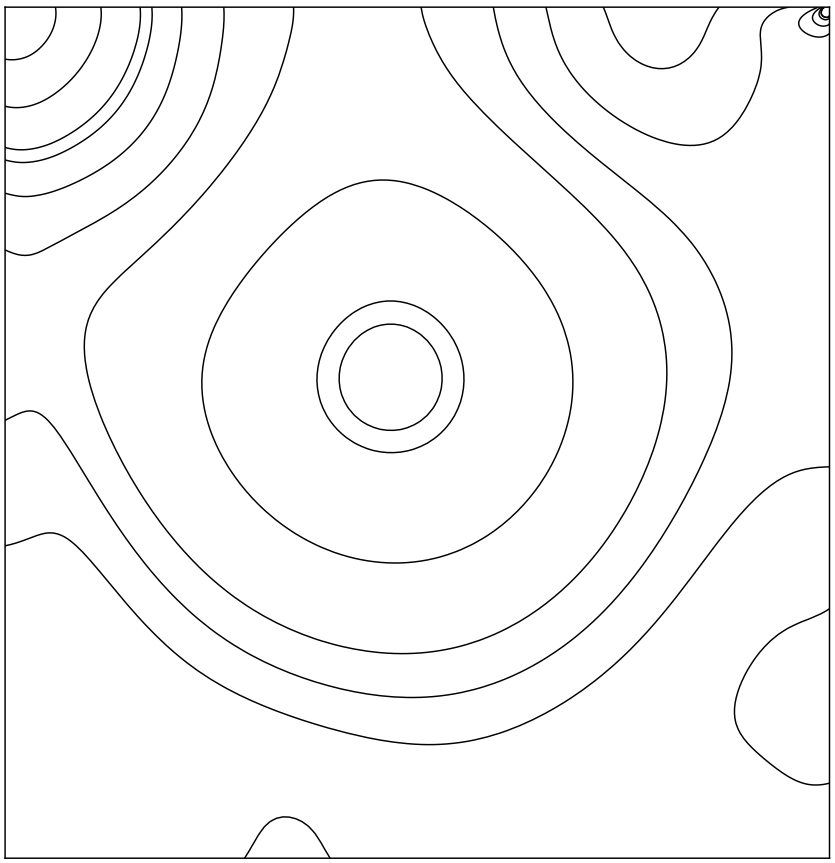


Figure 5.5: Pressure for $N = 64$, Re $= 1000$, $\Delta t = 0.0002$, and tol $= 1 \cdot 10^{-5}$.

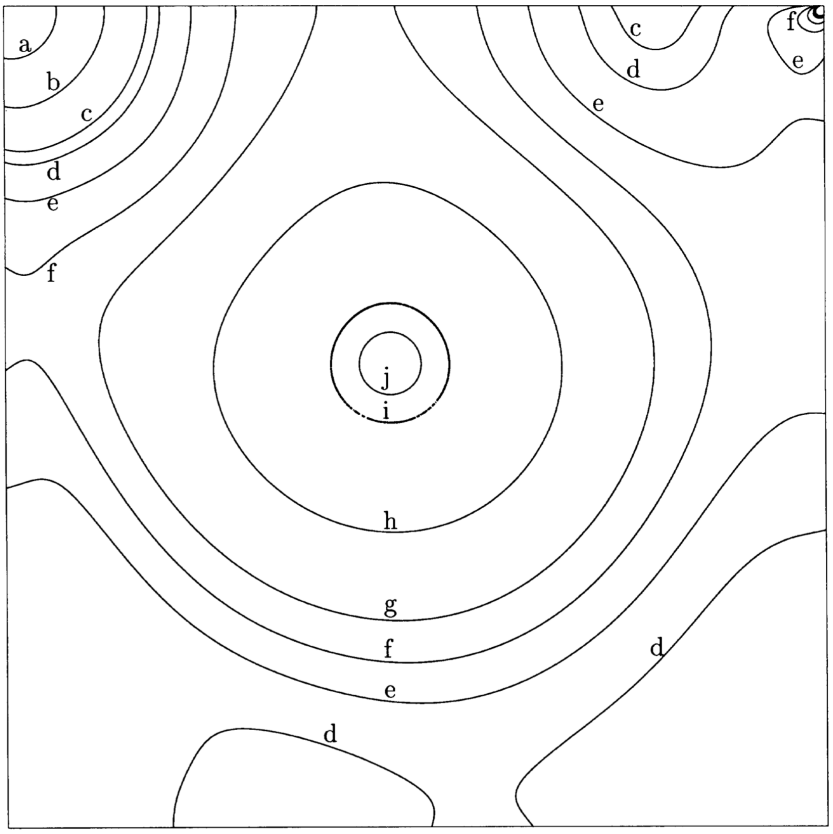
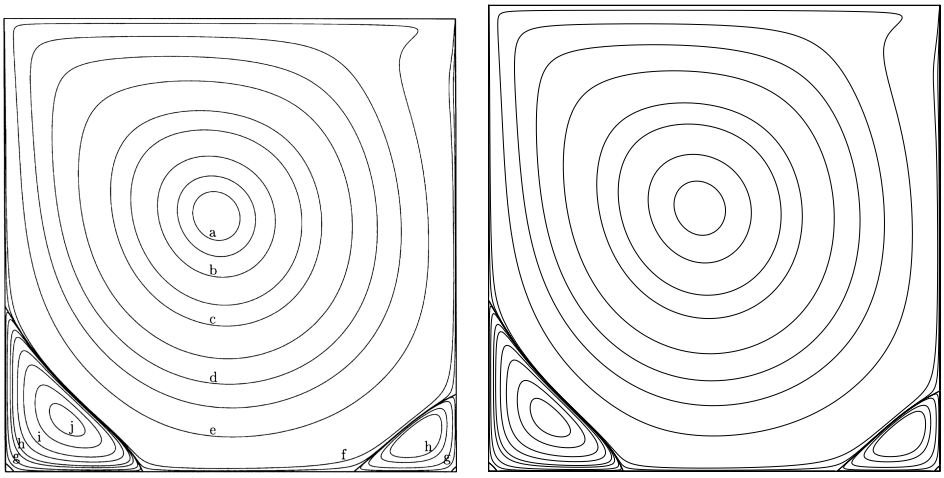
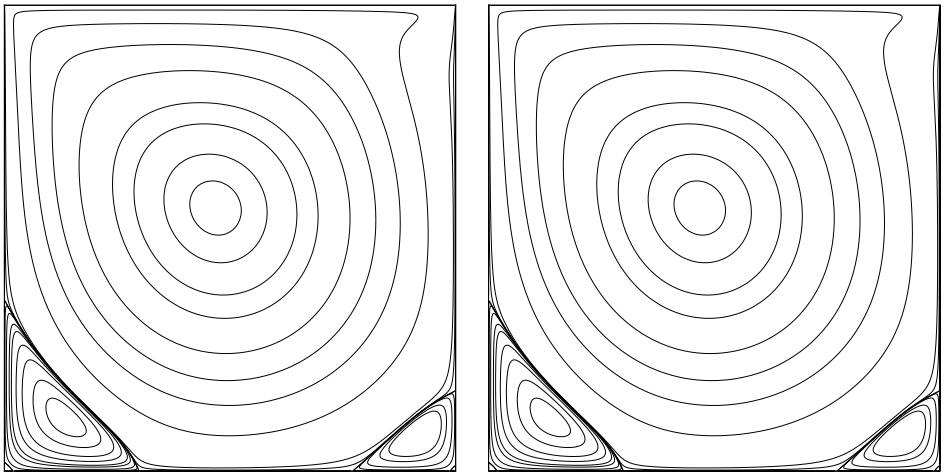


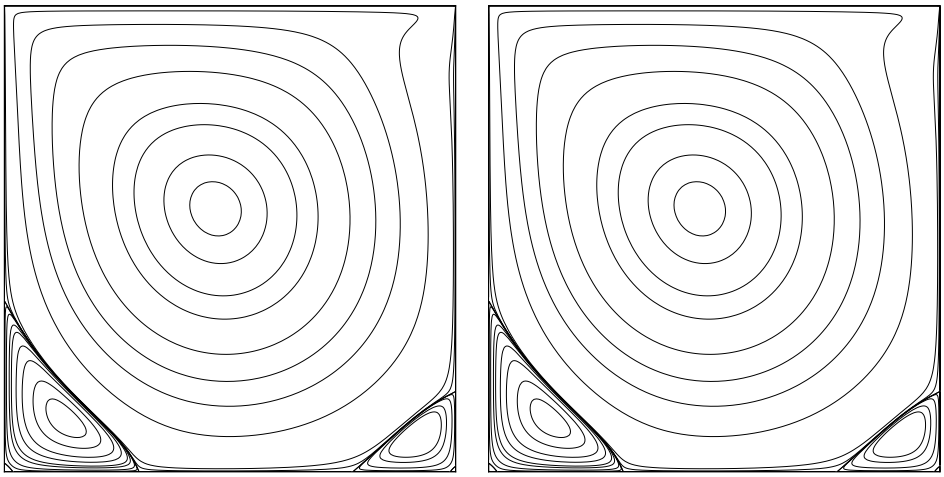
Figure 5.6: Benchmark pressure for Re $= 1000$ [1].



(a) Benchmark (b) $N = 64$

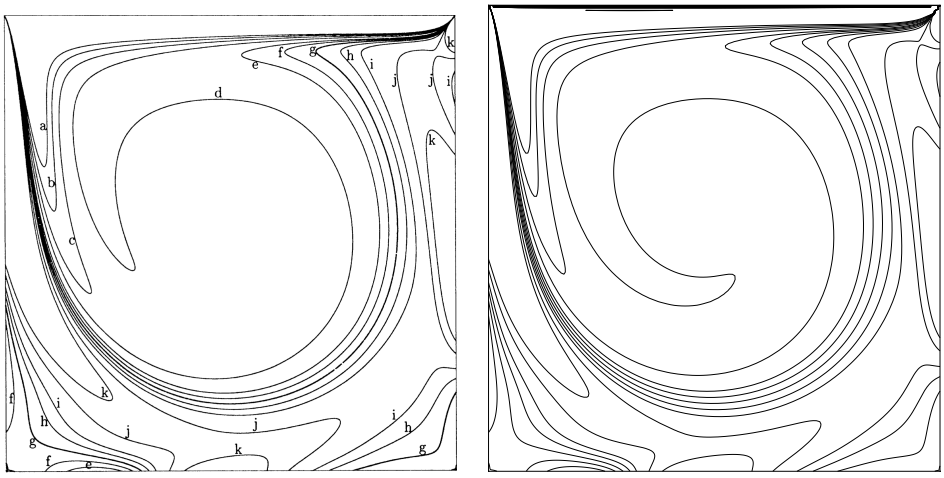


(c) $N = 56$ (d) $N = 48$

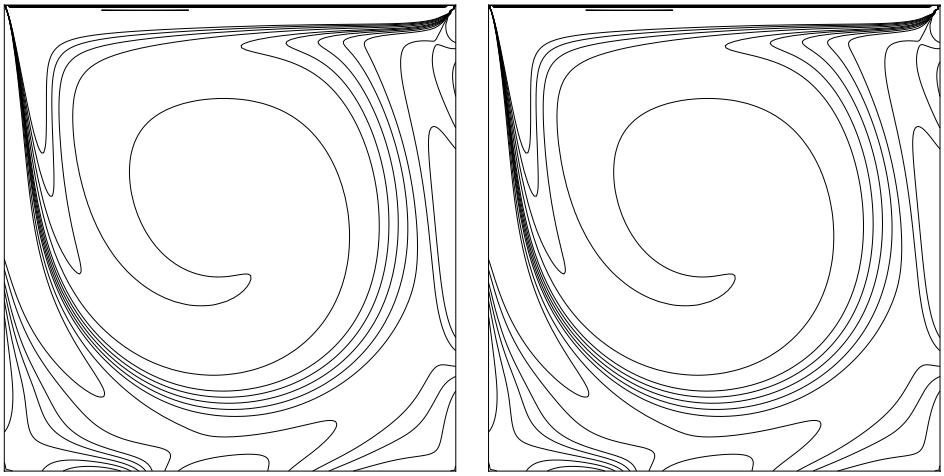


(e) $N = 32$ (f) $N = 16$

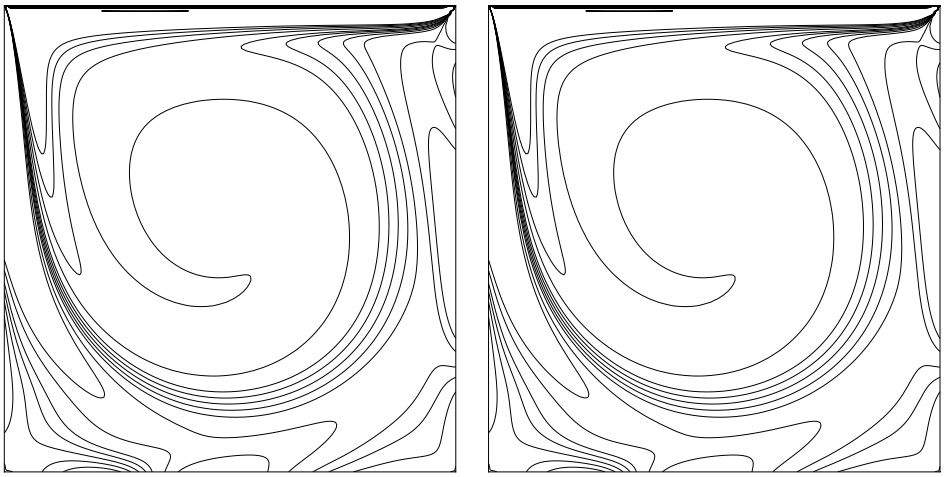
Figure 5.7: Stream function for Re = 1000, $\Delta t = 0.0002$, tol = $1 \cdot 10^{-5}$, and various values of N as well as the benchmark result



(a) Benchmark (b) $N = 64$

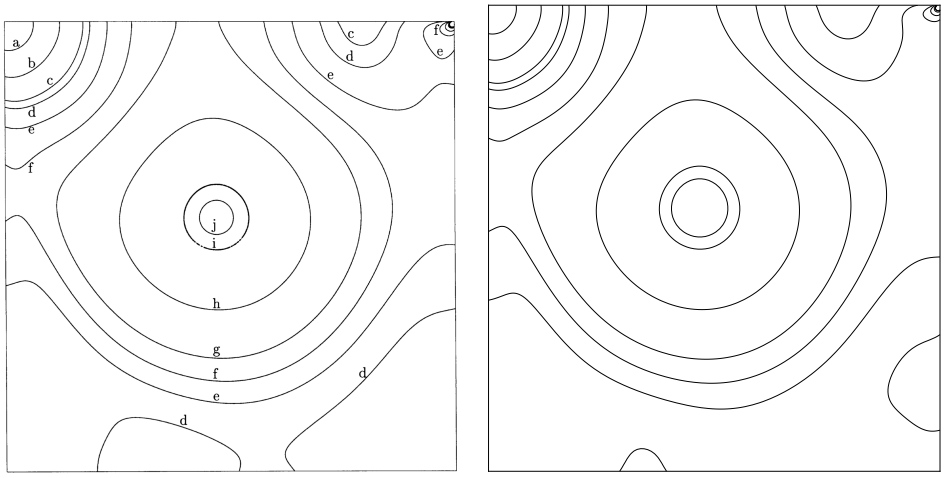


(c) $N = 56$ (d) $N = 48$

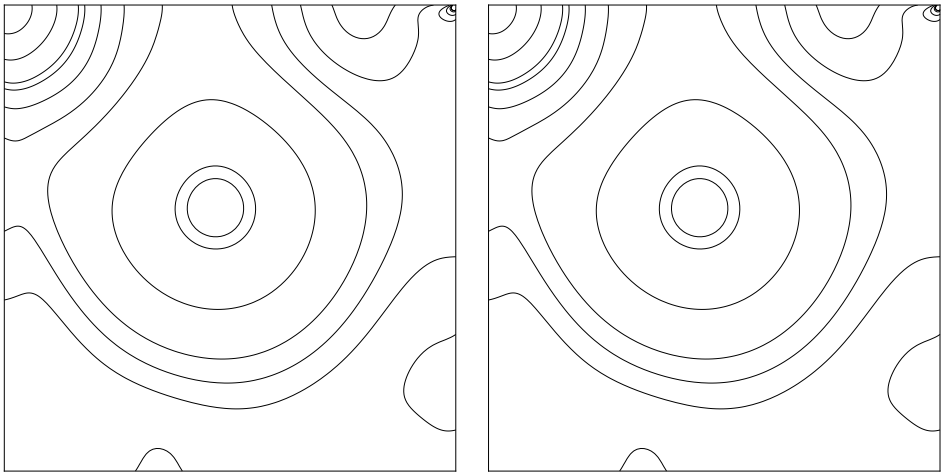


(e) $N = 32$ (f) $N = 16$

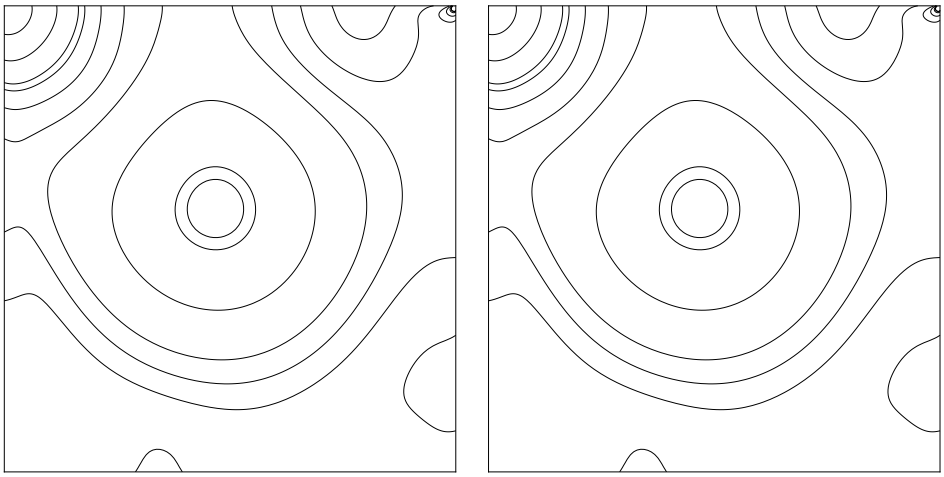
Figure 5.8: Vorticity for Re = 1000, $\Delta t = 0.0002$, tol = $1 \cdot 10^{-5}$, and various values of N as well as the benchmark result.



(a) Benchmark (b) $N = 64$



(c) $N = 56$ (d) $N = 48$



(e) $N = 32$ (f) $N = 16$

Figure 5.9: Static pressure for Re $= 1000$, $\Delta t = 0.0002$, tol $= 1 \cdot 10^{-5}$, and various values of N as well as the benchmark result.

6 Conclusion

We have presented a novel numerical method for solving the incompressible Navier-Stokes equations. The present method is proposed as an alternative to conventional numerical methods such as the finite-difference method, finite-volume method, and finite-element method. The present method was used to solve the incompressible Navier-Stokes equations on a two-dimensional lid-driven cavity, a classical test problem for the validation of Navier-Stokes codes. After comparing its results (i.e., the stream function, vorticity, and static pressure) with benchmark results in [1], it can be concluded that the novel method works and produces correct results.

To capture the geometric structure of the governing equations, we defined its physical quantities through integral values over the elements of the mesh. Depending on whether a given physical quantity was a point, line, or area density, its corresponding discrete representation "lived" at the associated zero, one, or two dimensional mesh elements. This is both an elegant and a natural approach and its advantages are manyfold. Numerical errors are only introduced when switching between the inner and outer-oriented grids, when computing the convection term, and when advancing the solution in time. The identification of error sources is trivial as the introduction of error is limited to only a small number of stages of the solution process. The present method results in highly sparse matrices that can be exploited to write computationally efficient code.

Future implementations of the present method should explore improved approximations of the convective term as the poor approximation of the convective term turns out to be the bottleneck of accuracy. Once the poor approximation of the convective term is improved, it may be benefition to incorporate multithreaded sparse matrix operations and higher-order time-stepping methods to improve execution time, memory use, and the rate of convergence. Aspiring developers of DEC based codes are also strongly advised to consider the method described in [6], whose authors use an implicit time-stepping scheme that is inherently independent of the timestep Δt.

References

[1] R. Peyret O. Botella. "Benchmark Spectral Results on The Lid-Driven Cavity Flow". In: *Computers and Fluids* 27.4 (1998), pp. 421–433.

[2] John D. Anderson. *Fundamentals of Aerodynamics*. 5th ed. McGraw-Hill Series in Aeronautical and Aerospace Engineering. McGraw-Hill, 2011. ISBN: 978-0-07-339810-5.

[3] Theodore Frankel. *The Geometry of Physics*. 3rd ed. McGraw-Hill Series in Aeronautical and Aerospace Engineering. Cambridge University Press, 2012. ISBN: 978-1-107-60260-1.

[4] Anil N. Hirani. "Discrete Exterior Calculus". PhD thesis. California Institute of Technology, 2003.

[5] Marc Gerritsma. *Computational Fluid Dynamics*. Tech. rep. Delft University of Technology, 2014.

[6] Sharif Elcott. "Discrete, Circulation-Preserving, and Stable Simplicial Fluids". MA thesis. California Institute of Technology, 2005.