The Mathematica® Journal

Developing an Understanding of the Steps Involved in Solving Navier–Stokes Equations

Desmond Adair
Martin Jaeger

This article describes how *Mathematica* can be used to develop an understanding of the basic steps involved in solving Navier–Stokes equations using a finite-volume approach for incompressible steady-state flow. The main aim is to let students follow from a mathematical description of a given problem through to the method of solution in a transparent way. The well-known "driven cavity" problem is used as the problem for testing the coding, and the Navier–Stokes equations are solved in vorticity-streamfunction form. Building on what the students were familiar with from a previous course, the solution algorithm for the vorticity-streamfunction equations chosen was a relaxation procedure. However, this approach converges very slowly, so another method using matrix and linear algebra concepts was also introduced to emphasize the need for efficient and optimized code.

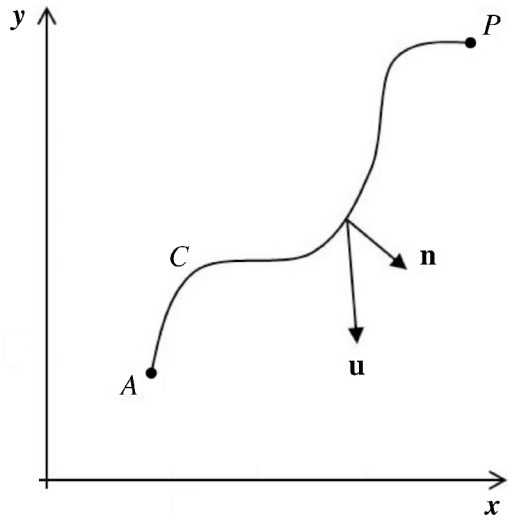
Introduction

Mathematica is used to help with an initial understanding of the process of solving Navier–Stokes equations. For 2D incompressible flows, it is possible to recast the Navier–Stokes equations in an alternative form in terms of the streamfunction and the vorticity. In many applications, the vorticity-streamfunction form of the Navier–Stokes equations provides better insight into the physical mechanisms driving the flow than the "primitive variable" formulation in terms of the mean velocities u, v, and pressure p. The streamfunction and vorticity formulation is also useful for numerical work, since it avoids some problems resulting from the discretization of the continuity equation.

The streamfunction is defined as

$$\psi_A(P) = \int_A^P \mathbf{u} \cdot \mathbf{n}\, ds, \tag{1}$$

where the integral has to be evaluated along a curve C from the arbitrary but fixed point A to point P, $\mathbf{u}$ is the velocity vector, and $\mathbf{n}$ is the unit normal on the curve from A to P; see Figure 1. We regard $\psi_A(P)$ as a function of the location of point P.



▲ **Figure 1.** Sketch illustrating the definitions of a streamfunction.

Figure 1 shows that $\mathbf{u} \cdot \mathbf{n}$ is equal to the component of the velocity $\mathbf{u}$ that crosses C. Therefore $\psi_A(P)$ represents the volume flux (per unit depth in the z direction) through C. Evaluating $\psi_A(P)$ along two different paths and invoking the integral form of the incompressibility constraint shows that $\psi_A(P)$ is path independent; that is, its value only depends on the locations of the points A and P. Changing the position of point A only changes $\psi_A(P)$ by a constant. It turns out that for all applications such changes are irrelevant. It is therefore common to suppress the explicit reference to A. Hence, we regard $\psi_A(P)$ as a function of the spatial coordinates only; that is, $\psi_A(P) = \psi(P) = \psi(x, y)$. Streamlines are lines that are everywhere tangential to the velocity field, that is, $\mathbf{u} \cdot \mathbf{n}$, where $\mathbf{n}$ is the unit normal to the streamline. Hence the streamfunction ψ is constant along streamlines. Note that stationary impermeable boundaries are also characterized by $\mathbf{u} \cdot \mathbf{n} = 0$, where $\mathbf{n}$ is the unit normal on the boundary. Therefore, ψ is also constant along such boundaries. Invoking the integral incompressibility constraint for an infinitesimally small triangle shows that ψ is related to the two Cartesian velocity components u and v via

$$u = \frac{\partial \psi}{\partial y};\ v = -\frac{\partial \psi}{\partial x}. \tag{2}$$

Flows that are specified by a streamfunction automatically satisfy the continuity equation, since

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=\frac{\partial}{\partial x}\left(\frac{\partial \psi}{\partial y}\right)-\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial x}\right)=0. \tag{3}$$

For 2D flows, the vorticity vector $\omega = \nabla \times \mathbf{u}$ only has one nonzero component (in the z direction); that is, $\omega = \omega\, e_z$, where

$$\omega=\frac{\partial v}{\partial x}-\frac{\partial u}{\partial y}. \tag{4}$$

Using the definition of the velocities in terms of the streamfunction shows that

$$\omega=\frac{\partial}{\partial x}\left(-\frac{\partial \psi}{\partial x}\right)-\frac{\partial}{\partial y}\left(\frac{\partial \psi}{\partial y}\right)=-\nabla^2\psi, \tag{5}$$

where $\nabla^2 = \partial^2 / \partial x^2 + \partial^2 / \partial y^2$ is the 2D Laplace operator.

The understanding of the steps involved in solving the Navier–Stokes equations using the vorticity-streamfunction form is one of the topics used in a third-year undergraduate course on computational fluid mechanics, solely for students majoring in mechanical engineering. The use of *Mathematica* makes the assumption that the students are familiar with the package, as it generally takes a good deal of exposure to *Mathematica* to become comfortable using it at the level required here [1]. The students taking the computational fluid mechanics course are indeed very familiar with *Mathematica*, as the computer algebra system is used during year one in the modules Calculus and Applications and Vector Calculus, and during year two in the module Numerical Methods for Engineering. In addition, there are many notes, explanations, and examples on the in-house Moodle open-source learning platform. Similar work to this can be found in Fearn [2], while background reading on computational fluid dynamics and vorticity may be found in Ferziger and Perić [3] and Chorin [4], respectively.

The aim here is to give a continuous and comprehensive process involving the mathematics of one formulation of the Navier–Stokes equations and a solution using *Mathematica*. In this way, the students can actually see the development of the mathematical description linked to a programming environment solution process so often hidden in commercial code used for training CFD students.

Navier–Stokes Equations in Vorticity-Streamfunction Form

The Navier–Stokes equations for incompressible steady-state flow in vorticity-streamfunction form are

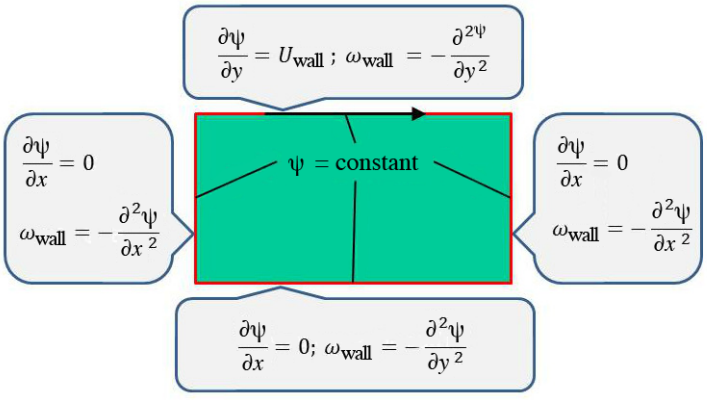
Advection-Diffusion Equation

$$\frac{\partial \psi}{\partial x}\frac{\partial \omega}{\partial y} - \frac{\partial \psi}{\partial y}\frac{\partial \omega}{\partial x} = \frac{1}{\mathbb{R}\mathrm{e}}\left(\frac{\partial^2 \omega}{\partial x^2} + \frac{\partial^2 \omega}{\partial y^2}\right); \tag{6}$$

Elliptic Equation

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega, \tag{7}$$

where $\mathbb{R}\mathrm{e}$ is the Reynolds number. For a given problem, boundary conditions need to be specified to solve equations (6) and (7). The problem chosen here introduces the student to a classical computational fluid dynamics (CFD) case, namely the lid-driven cavity flow [5]. This flow is commonly used to test, for example, a novel method of discretization of the equations or new computer programming, as the resulting flow is well known from experiments. Consider a rectangular box as shown in Figure 2, where a lid is allowed to move in the horizontal plane from left to right. When the lid is not moving, the fluid in the box is stationary, whereas when the lid is moving, the fluid circulates inside the box. Here the boundary conditions needed for solution are summarized in terms of vorticity and streamfunction. As all four walls of the cavity touch, the streamfuction must be equal for all four walls, as indicated by the ψ = constant in Figure 2. The streamfuction is constant at the walls, as its gradient is velocity, which is zero relative to a given wall (no-slip condition), as indicated in the outer boxes also shown in Figure 2. The vorticity boundary conditions for each wall are also shown in each of these outer boxes and were derived from the streamfunction.



▲ **Figure 2.** Summary of boundary conditions in terms of vorticity and streamfunction for cavity with moving lid.

Equations in Dimensionless Form

It is convenient from a numerical point of view to make equations (6) and (7) nondimensional. If a and b are the height and width of the cavity, respectively, and V is a reference velocity, then

$$\tilde{x} = \frac{x}{b};\ \tilde{y} = \frac{y}{a};\ \tilde{\psi} = \frac{\psi}{Vb};\ \tilde{\omega} = \frac{\omega\, b}{V};\ \gamma = \frac{b}{a};\ \mathbb{R}\mathrm{e} = \frac{Vb}{\nu}.$$

Equations (6) and (7) become

$$\frac{\partial^2 \tilde{\omega}}{\partial \tilde{x}^2} + \gamma^2 \frac{\partial^2 \tilde{\omega}}{\partial \tilde{y}^2} = \mathbb{R}\mathrm{e}\,\gamma\left(\frac{\partial \tilde{\psi}}{\partial \tilde{y}}\frac{\partial \tilde{\omega}}{\partial \tilde{x}} - \frac{\partial \tilde{\psi}}{\partial \tilde{x}}\frac{\partial \tilde{\omega}}{\partial \tilde{y}}\right), \tag{8}$$

$$\frac{\partial^2 \tilde{\psi}}{\partial \tilde{x}^2} + \gamma^2 \frac{\partial^2 \tilde{\psi}}{\partial \tilde{y}^2} = -\tilde{\omega}, \tag{9}$$

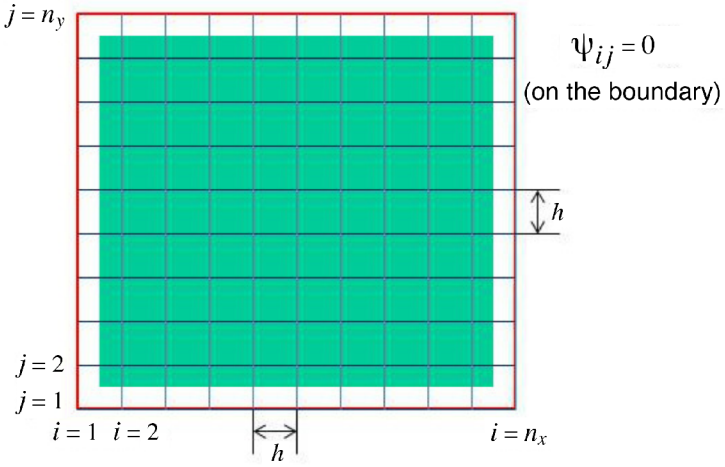
defined on the domain $0 \le \tilde{x} \le 1, 0 \le \tilde{y} \le 1$. The boundary equations are defined by

$$\begin{aligned}
&\tilde{\psi}(\tilde{x}, 0) = \psi(\tilde{x}, 1) = \tilde{\psi}(0, \tilde{y}) = \tilde{\psi}(1, \tilde{y}) = 0,\\
&\tilde{\omega}(\tilde{x}, 0) = -\gamma^2\left(\frac{\partial^2 \tilde{\psi}}{\partial \tilde{y}^2}\right)_{\tilde{y}=0}, \tilde{\omega}(\tilde{x}, 1) = -\gamma^2\left(\frac{\partial^2 \tilde{\psi}}{\partial \tilde{y}^2}\right)_{\tilde{y}=1},\\
&\tilde{\omega}(0, \tilde{y}) = -\left(\frac{\partial^2 \tilde{\psi}}{\partial \tilde{x}^2}\right)_{\tilde{x}=0}, \tilde{\omega}(1, \tilde{y}) = -\left(\frac{\partial^2 \tilde{\psi}}{\partial \tilde{x}^2}\right)_{\tilde{x}=1}.
\end{aligned} \tag{10}$$

Discretization and Solution Algorithm

Discretization of Transport Equations

Here is an iterative solution for solving the incompressible steady 2D Navier–Stokes equations. The method is based on Burggraf's proposal [6]. So as to include the boundary nodes of the cavity, central differencing is used to discretize equations (8) and (9). The number of nodes in the x and y directions is set at n_x and n_y, respectively. The mesh is a regular Cartesian grid with the nodes equally spaced at a distance h, as illustrated in Figure 3.



▲ **Figure 3.** Typical Cartesian mesh used for the lid-driven cavity flow.

To discretize equations (8) and (9), the following general finite central-difference approximations were introduced for spatial dimensions, obtained by adding or subtracting one Taylor series from another and invoking the intermediate value theorem. As can be seen from equation (11), the order of accuracy for both the first and second derivatives is $O(h^2)$.

$$\frac{\partial f(x)}{\partial x} = \frac{f(x+h) - f(x-h)}{2h} + O\left(h^2\right),$$
$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O\left(h^2\right). \tag{11}$$

Using the approximations from equation (11), the discretized advection-diffusion equation (equation (8)) at the (i, j) node is

$$\frac{\omega_{i+1,j} - 2\omega_{ij} + \omega_{i-1,j}}{h^2} + \gamma^2 \frac{\omega_{ij+1} - 2\omega_{ij} + \omega_{ij-1}}{h^2} =$$
$$\mathbb{R}\mathrm{e}\,\gamma \left(\frac{\psi_{ij+1} - \psi_{ij-1}}{2h} \frac{\omega_{i,+1j} - \omega_{i-1,j}}{2h} - \frac{\psi_{i+1,j} - \psi_{i-1j}}{2h} \frac{\omega_{i,j+1} - \omega_{ij-1}}{2h} \right). \tag{12}$$

The elliptic equation (equation (9)) at the (i, j) node is

$$\frac{\psi_{i+1,j} - 2\psi_{ij} + \psi_{i-1,j}}{h^2} + \gamma^2 \frac{\psi_{ij+1} - 2\psi_{ij} + \psi_{ij-1}}{h^2} = -\omega(x_i, y_j). \tag{13}$$

The equations are applied to the internal nodes of the Cartesian mesh; that is, $2 \le i \le n_x - 1$ and $2 \le j \le n_y - 1$.

Solution Algorithm

The solution method uses residual functions; that is, if the values of ψ_{ij} and ω_{ij} are exact on the nodes spanned by the residual functions $\mathcal{R}_{ij}$ and $\mathcal{L}_{ij}$, then

$$\mathcal{R}_{ij}^{k+1} = \mathcal{L}_{ij}^{k+1} = 0, \tag{14}$$

where

$$\mathcal{R}_{ij}^{k+1} = \frac{1}{2\,(1+\gamma^2)}\left(\psi_{i+1,j}^{k} + \psi_{i-1,j}^{k+1} + \gamma^2\left(\psi_{ij+1}^{k+1} + \psi_{ij-1}^{k+1}\right)\right) + h^2\,\omega_{ij}^{k} - \psi_{ij}^{k}, \tag{15}$$

$$\begin{aligned}\mathcal{L}_{ij}^{k+1} = {} & \\ & \frac{1}{2\,(1+\gamma^2)}\left(\left(\omega_{i+1,j}^{k} + \omega_{i-1,j}^{k} + \gamma^2\left(\omega_{ij+1}^{k} + \omega_{ij-1}^{k}\right)\right) - \frac{\mathbb{R}\mathrm{e}}{4}\gamma\left(\left(\psi_{ij+1}^{k} - \psi_{ij-1}^{k+1}\right)\left(\omega_{i+1,j}^{k} - \right.\right.\right. \\ & \left.\left.\left.\omega_{i-1,j}^{k+1}\right) - \left(\psi_{i+1,j}^{k} - \psi_{i-1,j}^{k+1}\right)\left(\omega_{ij+1}^{k} - \omega_{ij-1}^{k+1}\right)\right)\right) - \omega_{ij}^{k}.\end{aligned} \tag{16}$$

The following fixed-point iterative procedure, based on the Gauss–Seidel scheme, is then constructed:

$$\begin{aligned}\psi_{ij}^{k+1} &= \mathcal{F}^{k}\left(\psi^{(k)}, \psi^{(k+1)}, \omega^{(k)}, \omega^{(k+1)}\right) \equiv \psi_{ij}^{k} + p\,\mathcal{R}_{ij}^{k}, \\ \omega_{ij}^{k+1} &= \mathcal{G}^{k}\left(\psi^{(k)}, \psi^{(k+1)}, \omega^{(k)}, \omega^{(k+1)}\right) \equiv \omega_{ij}^{k} + p\,\mathcal{L}_{ij}^{k},\end{aligned} \tag{17}$$

where p is a relaxation parameter lying in the range $0 < p \le 1$, and $k+1$ and k refer to the respective iterations. In this article, the actual value of p depends on $\mathbb{R}\mathrm{e}$ and can be obtained by numerical experimentation. The use of this relaxation parameter is to improve the stability of a computation, particularly in solving steady-state problems. It works by limiting, when necessary, the amount that a variable changes from one iteration to the next. The optimum choice of p is one that is small enough to ensure stable computation but large enough to move the iterative process forward quickly.

The boundary conditions now need to be determined. For the streamfunction from equation (11),

$$\begin{aligned}&\psi_{i,1} = 0 \text{ for } i = 1, \ldots, n_x, \\ &\psi_{n_x,j} = 0 \text{ for } j = 1, \ldots, n_y, \\ &\psi_{i,n_y} = 0 \text{ for } i = 1, \ldots, n_x, \\ &\psi_{1,j} = 0 \text{ for } j = 1, \ldots, n_y.\end{aligned} \tag{18}$$

For vorticity, the following needs to be considered for, say, the left wall of the cavity with one node outside the computational domain:

$$\omega_{ij} = -\frac{1}{h^2}\left(\psi_{2,j} - 2\,\psi_{1,j} + \psi_{0,j} + \gamma^2(\psi_{1,j+1} - 2\,\psi_{1,j} + \psi_{1,j-1})\right). \tag{19}$$

The value of $\psi_{0,j}$ can be accounted for using the no-slip boundary condition on the cavity wall; that is,

$$v = -\frac{\partial \psi}{\partial x} = 0 \text{ at } x = 0. \tag{20}$$

Using central differences, this condition can be written as

$$-\left(\frac{\partial \psi}{\partial x}\right)_{1,j} = 0 \Longrightarrow -\left(\frac{\psi_{2,j} - \psi_{0,j}}{2\,h}\right) = 0 \Longrightarrow \psi_{0,j} = \psi_{2,j}, \tag{21}$$

where again the order of accuracy is $O(h^2)$.

This finding, together with $\psi_{1,j} = \psi_{1,j+1} = \psi_{1,j-1} = 0$, gives

$$\omega_{1,j} = -\frac{2}{h^2}(\psi_{2,j}) \text{ for } j = 2, n_y - 1. \tag{22}$$

Similarly, for the other walls of the cavity the boundary conditions for vorticity can be deduced from

$$\omega(x_i, y_i) = -\left(\frac{\psi_{i+1,j} - 2\,\psi_{ij} + \psi_{i-1,j}}{h^2} + \gamma^2\,\frac{\psi_{ij+1} - 2\,\psi_{ij} + \psi_{ij-1}}{h^2}\right) \tag{23}$$

to give

$$\begin{aligned}
\omega_{i,1} &= -\frac{2}{h^2}\gamma^2(\psi_{i,2}) \text{ for } i = 1, \ldots, n_x, \\
\omega_{n_x j} &= -\frac{2}{h^2}(\psi_{n_x-1,j}) \text{ for } j = 2, \ldots, n_y - 1, \\
\omega_{in_y} &= -\frac{2}{h^2}\gamma^2\left(\psi_{in_y-1} + \frac{h}{\gamma}\right) i = 1, \ldots, n_x,
\end{aligned} \tag{24}$$

where n_x and n_y are defined in Figure 3. Note that the final boundary condition shown in equation (24) is for the moving lid, and the extra term is due to the tangential velocity being nonzero.

Use of *Mathematica*

This section starts with the definition of mesh size and grid spacing; for this problem the mesh spacing is set equal in the x and y directions. The parameters `Nx` and `Ny` denote the mesh size. Initially, ω and ψ are set to zero for all nodes except at the top wall, where vorticity is not zero. The Reynolds number is set at 100, the relaxation parameter `p` is set to 1, and the aspect ratio γ is set at 1. A maximum allowable residual value `e` is set, and nested loops are used to execute the iterative algorithm. At every iteration step `k`, if the maximum value of the absolute value of $\mathcal{L}$`[i, j]` is less than `e`, the calculations are halted. The iterative loops are wrapped with the function `Timing` to give an estimate of the time taken to do the calculations.

■ *Geometry, Mesh Parameters, Initial and Boundary Conditions*

```
a = 1; b = 1; Nx = 41; Ny = 41; ℝe = 100; p = 1.0; γ = 1.;
h = a / (Nx - 1) // N;
Do[ω[i, j] = 0, {i, Nx}, {j, Ny - 1}];
Do[ψ[i, j] = 0, {i, Nx}, {j, Ny}];
Do[ω[i, Ny] = -2 γ^2 (ψ[i, Ny - 1] + h/γ) / h^2, {i, Nx}];
```

■ *Transport Equations*

```
ℛ[i_, j_] :=
  1/(2 (1 + γ^2)) (ψ[i + 1, j] + ψ[i - 1, j] + γ^2 (ψ[i, j - 1] + ψ[i, j + 1]) +
      h^2 ω[i, j]) - ψ[i, j]
ℒ[i_, j_] :=
  1/(2 (1 + γ^2)) ((ω[i + 1, j] + ω[i - 1, j] + γ^2 (ω[i, j - 1] + ω[i, j + 1])) -
      ℝe/4 γ ((ψ[i, j + 1] - ψ[i, j - 1]) (ω[i + 1, j] - ω[i - 1, j]) -
          (ψ[i + 1, j] - ψ[i - 1, j]) (ω[i, j + 1] - ω[i, j - 1]))) -
  ω[i, j]
```

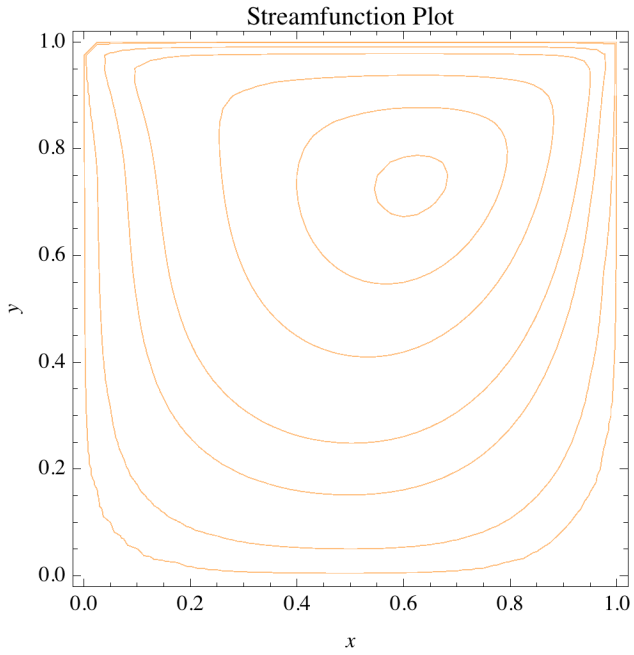
■ Iterative Algorithm

```
With[{e = 0.001},
 Timing[
  For[
   k = 1,
   Max[Abs[Table[ℒ[i, j], {i, 2, Nx - 1}, {j, 2, Ny - 1}]]] > e,
   k++,
   For[
    i = 2, i < Nx, i++,
    For[
     j = 2, j < Ny, j++,
     ψnew[i, j] = ψ[i, j] + p ℛ[i, j];
     ωnew[i, j] = ω[i, j] + p ℒ[i, j];
     ψ[i, j] = ψnew[i, j];
     ω[i, j] = ωnew[i, j];
     ω[1, j] = -2 ψ[2, j] / h^2;
     ω[i, 1] = -2 γ^2 ψ[i, 2] / h^2;
     ω[Nx, j] = -2 ψ[Nx - 1, j] / h^2;
     ω[i, Ny] = -2 γ^2 (ψ[i, Ny - 1] + h/γ) / h^2
    ]
   ]
  ]
 ]
]
```

```
{56.7269, Null}
```

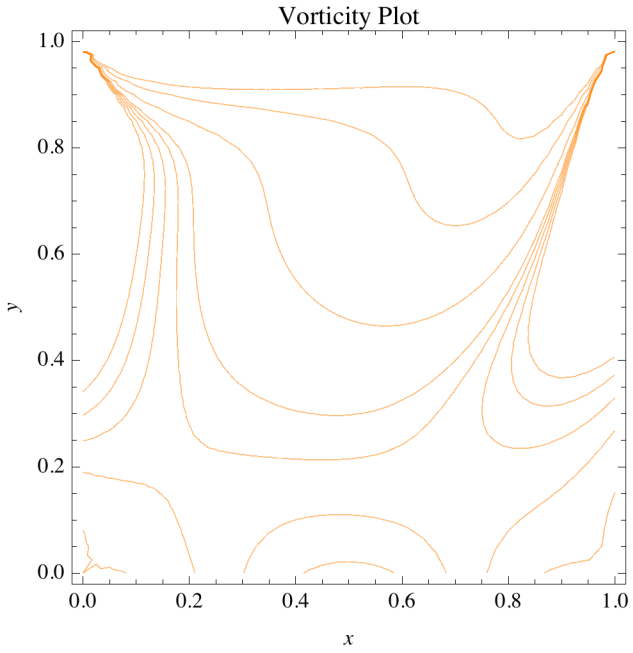
■ *Streamfunction Results*

```
ListContourPlot[
 Transpose[
  Partition[Flatten[Table[ψ[i, j], {i, Nx}, {j, Ny}]], Ny]],
 Contours → {-0.00005, -0.008, -0.1, -0.08, -0.05,
   -0.02, -0.001},
 ContourShading → False,
 AspectRatio → Automatic,
 ContourStyle → Orange,
 DataRange → {{0, 1}, {0, 1}},
 FrameLabel → {x, y},
 ImageSize → 300,
 PlotLabel → "Streamfunction Plot"
]
```



■ Vorticity Results

```
ListContourPlot[
 Transpose[
  Partition[Flatten[Table[ω[i, j], {i, Nx}, {j, Ny}]], Ny]],
 Contours → {-5, -3, -1, 0, 0.25, 0.5, 0.75, 1},
 ContourShading → False,
 ContourStyle → Orange,
 DataRange → {{0, 1}, {0, 1}},
 FrameLabel → {x, y},
 ImageSize → 300,
 PlotLabel → "Vorticity Plot"
]
```



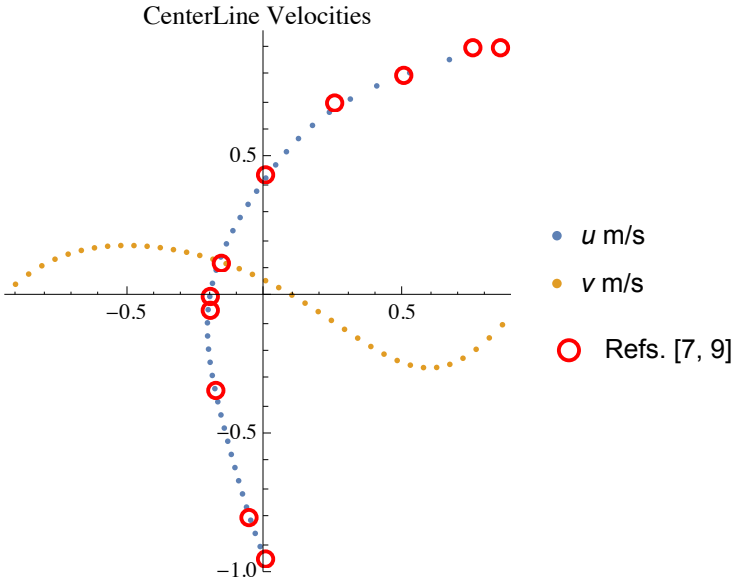
Extraction of the centerline velocities is also instructive. First of all, students are given more experience viewing velocity profiles, and second, there are ample calculations and measurements in the literature [7, 8, 9] for comparison with the results obtained here. The centerline velocities u (in the x direction) and v (in the y direction) were derived from the streamfunction values using the equations

$$u_{i,j} = \frac{\partial \psi}{\partial y} = \frac{\psi_{i,j+1} - \psi_{i,j-1}}{2\,h} \; j = 2, \ldots, n_y, \tag{25}$$

$$v_{i,j} = -\frac{\partial \psi}{\partial y} = \frac{\psi_{i,+1\,j} - \psi_{i-1,j}}{2\,h} \; i = 2, \ldots, n_x. \tag{26}$$

The velocity distributions for u and v in the following figure are drawn along the vertical and horizontal centerlines, respectively.

```
Module[
 {Nxm = (Nx + 1) / 2, Nym = (Ny + 1) / 2, i, j, u, v, xx},
 For[j = 2, j < Ny - 1, j++,
  u[Nxm, j] = (ψ[Nxm, j + 1] - ψ[Nxm, j - 1]) / (2 h)];
 For[j = 2, j < Ny - 1, j++, u[Nxm, j] = -u[Nxm, j]];
 For[i = 2, i < Nx - 1, i++,
  v[i, Nym] = -(ψ[i + 1, Nym] - ψ[i - 1, Nym]) / (2 h)];
 For[j = 2, j < Ny - 1, j++, xx[j] = (j - Nym) / Nym];
 Show[
  ListPlot[
   {
    Table[{-u[Nxm, j], xx[j]}, {j, 2, Ny - 1}],
    Table[{xx[i], v[i, Nxm]}, {i, 2, Nx - 1}]
   },
   ImageSize → 300,
   AspectRatio → Automatic,
   PlotLegends → {Row[{Style["u", Italic], " m/s"}],
     Row[{Style["v", Italic], " m/s"}]},
   PlotLabel → "CenterLine Velocities"
  ],
  ListPlot[
   {
    {0.0, -0.95}, {-0.06, -0.8}, {-0.18, -0.34},
    {-0.2, -0.05}, {-0.2, 0.0}, {-0.16, 0.12}, {0, 0.44},
    {0.25, 0.7}, {0.5, 0.8}, {0.75, 0.9}, {0.85, 0.9}
   },
   PlotMarkers → Graphics[{Red, Circle[]}, ImageSize → 8],
   PlotLegends → {"Refs. [7, 9]"}]
 ]
]
```



Importantly, the students were made aware that the profiles they calculated must be compared, preferably with experimental measurements, to test the validity of the calculation technique. As can be seen from the preceding figure, the calculations for the horizontal centerline velocities compare very favorably with those reported in the literature [7, 9] for $\mathbb{R}e = 100$.

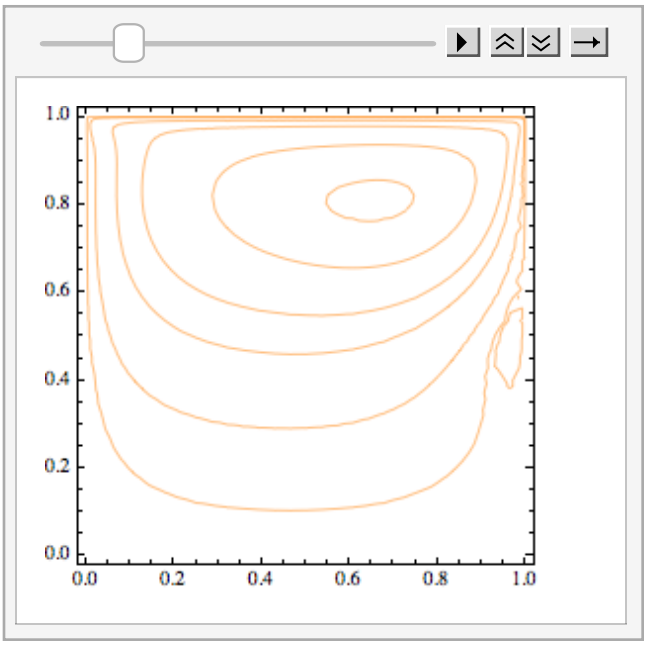
A More Efficient and Optimized Solution Method

It can be clearly seen that the relaxation procedure, though reasonably easy for students to follow and therefore educationally productive, is indeed very slow to converge. However, students must be aware that code should be written efficiently and optimized, making use of all available concepts and procedures. Therefore, as part of their understanding of solving Navier–Stokes equations in the best way, they were also introduced to the following *Mathematica* code. The preceding exercise was a small-scale problem using a fairly primitive method of solution, and even for this simple problem, the convergence time is prohibitive. Therefore, for large numerical computations and those with more complex geometry, it is important to use a solution method that will run more efficiently. This can be achieved using linear algebra and matrices and highly optimized functionalities built into *Mathematica*, namely `NDSolve` and `LinearSolve`. The following program solves the same problem as that described in Figure 2, except the Reynolds number is set at 400 and the computational grid is slightly more dense. The time to solve this same problem was reduced by over an order of magnitude. It should be understood that what the code is solving is the unsteady 2D Navier–Stokes equations. Each iteration is a time step from impulse-starting conditions to steady state, and for the lid-driven cavity flow, the unsteady solution converges to the steady state. The animation shown is for the streamfunction with a Reynolds number of 400.

■ *Optimized Coding*

```
Module[
 (* global: dx, dy, sf, grid *)
 {
  ℝe = 400, γ = 1.,
  nx = ny = N[Range[0, 50] / 50],
  dh, dt = 0.02,
  eye, d2x, d2y,
  biLeft, biRight, biBottom, biTop,
  bi,
  vbiLeft, vbiRight, vbiBottom, vbiTop,
  LaplacianSV,
  vort, sfSol, sfPlots
 },
 dh = nx[[2]] - nx[[1]];
 {eye, dx, d2x, dy, d2y} =
  NDSolve`FiniteDifferenceDerivative[#, {nx, ny},
       "DifferenceOrder" → 2]["DifferentiationMatrix"] & /@
    {{0, 0}, {1, 0}, {2, 0}, {0, 1}, {0, 2}};
 grid = Flatten[Outer[List, nx, ny], 1];
 {biLeft, biRight, biBottom, biTop} =
Flatten[Position[grid, #]] & /@
   {{0., y_}, {1., y_}, {x_, 0.}, {x_, 1.}} ;
 bi = DeleteDuplicates[
   Flatten[{biLeft, biRight, biBottom, biTop}]];
 {vbiLeft, vbiRight, vbiBottom, vbiTop} =
  Flatten[Position[grid, #]] & /@
   {{dh, y_}, {1. - dh, y_}, {x_, dh}, {x_, 1. - dh}};
 LaplacianSV = eye - dt / ℝe (d2x + γ^2 d2y);
 LaplacianSV[[bi]] = eye[[bi]];
 LUMat = LinearSolve[LaplacianSV];
 vort = sf = ConstantArray[0., Length[grid]];
 sfSol = Last@Reap[
    Do[
     rhs = vort - dt γ (dy.sf dx.vort - dx.sf dy.vort) ;
     rhs[[biLeft]] = -2 / dh^2 sf[[vbiLeft]];
     rhs[[biRight]] = -2 / dh^2 sf[[vbiRight]];
     rhs[[biBottom]] = -2 / dh^2 γ^2 sf[[vbiBottom]];
     rhs[[biTop]] = -2 / dh^2 γ^2 (sf[[vbiTop]] + dh / γ);
     vort = LUMat[rhs];
     rhs = dt / ℝe vort + sf;
     rhs[[bi]] = 0.;
     sf = LUMat[rhs];
     If[
      Mod[timeStep - 1, 100] == 0,
      Sow[Interpolation[Transpose[{grid, sf}]]]
     ],
     {timeStep, 3000}
```

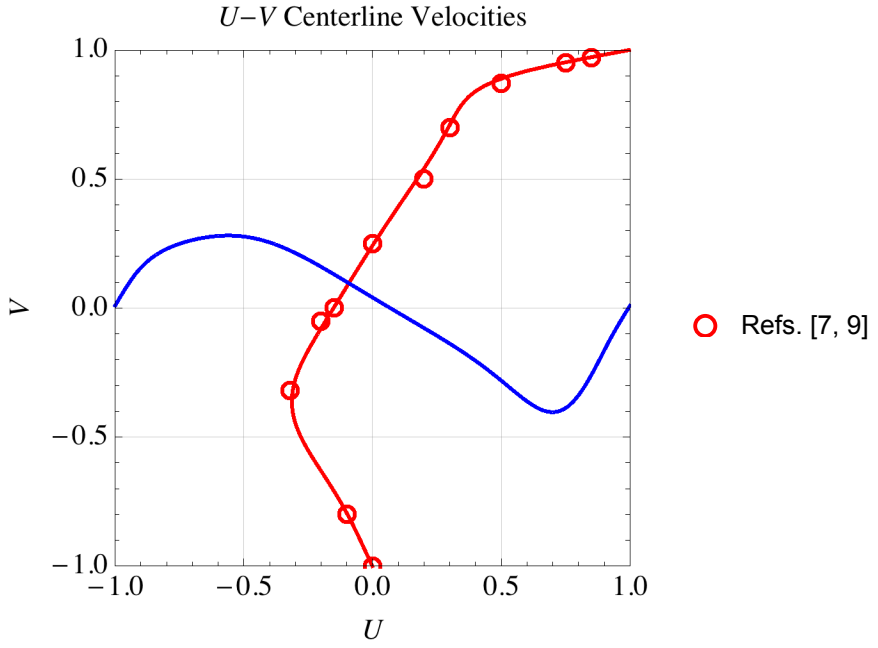
```
    ]
   ];
 sfPlots = Rasterize@ContourPlot[
      #[x, y], {x, 0, 1}, {y, 0, 1},
      Contours → {-0.00005, -0.008, -0.1, -0.08,
        -0.05, -0.02, -0.001, 0.001, 0.0001, 0.0005,
        0.00001},
      ContourShading → False,
      ContourStyle → Orange,
      ImageSize → 250] & /@ First[sfSol];
ListAnimate[sfPlots, AnimationRunning → False]
]
```




```
U = Interpolation[Transpose[{grid, dy.sf}]];
V = Interpolation[Transpose[{grid, -dx.sf}]];

Show[
 ListLinePlot[
  {
   Table[{U[0.5, y], 2 y - 1}, {y, 0, 1, 1 / 100.}],
   Table[{2 x - 1, V[x, 0.5]}, {x, 0, 1, 1 / 100.}]
  },
  PlotRange → {{1, -1}, {1, -1}},
  AspectRatio → 1,
  Axes → False,
  Frame → True,
  GridLines → Automatic,
  FrameLabel → {Style["U", Italic], Style["V", Italic]},
```

```
  PlotStyle → {Red, Blue}, FrameStyle → Directive[Black, 12],
  PlotLabel →
   Text[
    Style[Row[{Style["U", Italic], "-", Style["V", Italic],
       " Centerline Velocities"}], 12]]
 ],
 ListPlot[
  {{0.0, -1.0}, {-0.1, -0.8}, {-0.32, -0.32},
   {-0.2, -0.05}, {-0.15, 0.0}, {0.0, 0.25}, {0.2, 0.5},
   {0.3, 0.7}, {0.5, 0.87}, {0.75, 0.95}, {0.85, 0.97}},
  PlotMarkers → Graphics[{Red, Circle[]}, ImageSize → 8],
  PlotLegends → {"Refs. [7, 9]"}
 ]
]
```



Having established the method of solution, the student would then experiment by varying the grid size and seeking a solution independent of the grid, which is very important in CFD calculations. Results would also be obtained for different aspect ratios of the container. Again, where possible, calculations must be compared with experimental measurements to establish validity. As can be seen, the calculated horizontal velocity along the vertical centerline for $\mathbb{R}\mathrm{e} = 400$ was in good agreement with experimental results [7, 9].

Summary

This article outlines a well-defined sequence of steps needed to solve the Navier–Stokes equations cast in the vorticity-streamfunction form. The sequence is integrated with the use of *Mathematica* at appropriate stages to take care of the tedious computations, and hence to allow the students to concentrate on the overall details of the solution process. In addition, it is now important to integrate computer technology so as to complete lectures and theory. This has the advantages of helping with the computations, aiding presentation for reports and analysis, and motivating students. Incorporating *Mathematica* also takes away the "black-box" approach so often being used by students with full CFD commercial codes, which give no real understanding of the numerics involved. The idea of efficient and optimized coding was also introduced.

References

[1] S. Pomeranz, "Using a Computer Algebra System to Teach the Finite Element Method," *International Journal of Engineering Education*, **16**(4), 2000 pp. 362–368. www.ijee.ie/articles/Vol16-4/IJEE1162.pdf.

[2] R. L. Fearn, "Airfoil Aerodynamics Using Panel Methods," *The Mathematica Journal*, **10**(4), 2008 pp. 725–739. www.mathematica-journal.com/2008/11/airfoil-aerodynamics-using-panel-methods.

[3] J. H. Ferziger and M. Perić, *Computational Methods for Fluid Dynamics*, 3rd ed., Berlin: Springer, 2002.

[4] A. J. Chorin, *Vorticity and Turbulence* (*Applied Mathematical Sciences*, Vol. 103), New York: Springer-Verlag, 1994.

[5] J. D. Bozeman and C. Dalton, "Numerical Study of Viscous Flow in a Cavity," *Journal of Computational Physics*, **12**(3), 1973 pp. 348–363. doi:10.1016/0021-9991(73)90157-5.

[6] O. R. Burggraf, "Analytical and Numerical Studies of the Structure of Steady Separated Flows," *Journal of Fluid Mechanics*, **24**(1), 1966 pp. 113–151. doi:10.1017/S0022112066000545.

[7] U. Gia, K. N. Ghia, and C. T. Shin, "High-Re Solutions for Incompressible Flow Using the Navier–Stokes Equations and a Multigrid Method," *Journal of Computational Physics*, **48**(3), 1982 pp. 387–411. doi:10.1016/0021-9991(82)90058-4.

[8] W. F. Spotz, "Accuracy and Performance of Numerical Wall Boundary Conditions for Steady, 2D, Incompressible Streamfunction Vorticity," *International Journal for Numerical Methods in Fluids*, **28**(4), 1998 pp. 737–757. onlinelibrary.wiley.com/doi/10.1002/%28SICI%291097-0363%2819980930%2928:4%3C737::AID-FLD744%3E3.0.CO;2-L/abstract.

[9] D. C. Wan, Y. C. Zhou, and G. W. Wei, "Numerical Solution of Incompressible Flows by Discrete Singular Convolution," *International Journal for Numerical Methods in Fluids*, **38**(8), 2002 pp. 789–810. doi:10.1002/fld.253.

D. Adair and M. Jaeger, "Developing an Understanding of the Steps Involved in Solving Navier–Stokes Equations," *The Mathematica Journal*, 2015. dx.doi.org/doi:10.3888/tmj.17-8.

About the Authors

Desmond Adair is a professor of mechanical engineering in the School of Engineering, Nazarbayev University, Astana, Republic of Kazakhstan. His recent research interests include investigations of airborne pollution for both passive and reacting flows, and developing engineering mathematics by the incorporation of computer algebra systems.

Martin Jaeger is an associate professor of civil engineering and head of department in the School of Engineering, Australian College of Kuwait, Misref, Kuwait. His recent research interests include construction management and total quality, as well as developing strategies for engineering education.

Desmond Adair
School of Engineering
Nazarbayev University
53 Kabanbay batyr Ave.
Astana, 010000, Republic of Kazakhstan
dadair@nu.edu.kz

Martin Jaeger
School of Engineering
Australian College of Kuwait
Al Aqsa Mosque Street
Misref, Kuwait City, Kuwait
m.jaeger@ack.edu.kw