8 Finite-difference methods

In previous chapters, we have discussed the equations governing the structure of a steady flow and the evolution of an unsteady flow, and derived selected solutions for elementary flow configurations by analytical and simple numerical methods. To generate solutions for arbitrary flow conditions and boundary geometries, it is necessary to develop general-purpose numerical methods. In this chapter, we discuss the choice of governing equations whose solution is to be found, and the implementation of finite-difference methods for incompressible Newtonian flow. The discourse will reveal a set of conceptual and practical challenges encountered in the broader context of computational fluid dynamics (CFD).

8.1 Choice of governing equations

General-purpose methods for computing the flow of an incompressible Newtonian fluid can be classified into two categories distinguished by the choice of governing equations.

In the first class of methods, the flow is described in terms of primary variables, including the velocity and the pressure. The structure of the velocity and pressure fields in a steady flow, and the evolution of the velocity and pressure fields in an unsteady flow are computed by solving the Navier–Stokes equation and the continuity equation, subject to appropriate boundary conditions, initial conditions, and possibly supplemental constraints.

In the second class of methods, the flow is computed based on the vorticity transport equation. The numerical procedure involves two stages: first, the structure or evolution of the vorticity field is computed based on the vorticity transport equation discussed in Section 6.6; second, the simultaneous structure or evolution of the velocity field is obtained

C. Pozrikidis, *Fluid Dynamics*, DOI 10.1007/978-1-4899-7991-9_8

by inverting the equation defining the vorticity as the curl of the velocity,

$$\boldsymbol{\omega} = \boldsymbol{\nabla} \times \mathbf{u}, \tag{8.1.1}$$

subject to constraints imposed by the continuity equation and boundary conditions. Inverting (8.1.1) involves solving for $\mathbf{u}$ in terms of $\boldsymbol{\omega}$. Descendant methods are distinguished by the particular procedure used to recover the velocity field from a specified vorticity distribution.

The strengths and weaknesses of the aforementioned two classes of methods will become apparent as we describe their implementation. One appealing feature of the second class of methods based on the vorticity transport equation is the lack of need to solve for the pressure. Bypassing the computation of the pressure is desirable when boundary conditions for the pressure are not directly available but must be derived from the governing equations. Disadvantages include the need to derive boundary conditions for the vorticity.

Problem

8.1.1 *Inversion of the vorticity*

Show that, if $\mathbf{u}$ is a solenoidal velocity field corresponding to a certain vorticity field $\boldsymbol{\omega}$, that is, $\boldsymbol{\nabla} \cdot \mathbf{u} = 0$, then the velocity field

$$\mathbf{v} = \mathbf{u} + \boldsymbol{\nabla} f \tag{8.1.2}$$

corresponds to the same vorticity field, where f is an arbitrary smooth scalar function. Explain why, for the velocity field $\mathbf{v}$ to remain solenoidal, $\boldsymbol{\nabla} \cdot \mathbf{v} = 0$, the function f must be harmonic, that is, it must satisfy Laplace's equation, $\nabla^2 f = 0$.

8.2 Unidirectional flow; velocity/pressure formulation

We begin developing finite-difference methods by discussing the velocity/pressure formulation for unidirectional flow in a channel confined between two parallel walls located at $y = 0$ and $y = h$, as illustrated in Figure 8.2.1. The lower and upper walls translate parallel to themselves along the x axis with generally time-dependent velocities, $V_1(t)$ and $V_2(t)$.

In practice, channel flow occurs under two complementary conditions reflecting the physical mechanism driving the flow, as follows:

- In the first case, the flow rate along the channel $Q(t)$ is prescribed and the streamwise pressure gradient $\partial p(t)/\partial x$ is computed as part of the solution.
- In the second case, the pressure gradient is specified, and the flow rate is computed as part of the solution.

In this section and in Section 8.3 we consider the case of flow driven to a specified and possibly time-dependent pressure gradient. In Section 8.4, we consider the complementary case of flow subject to a specified flow rate.

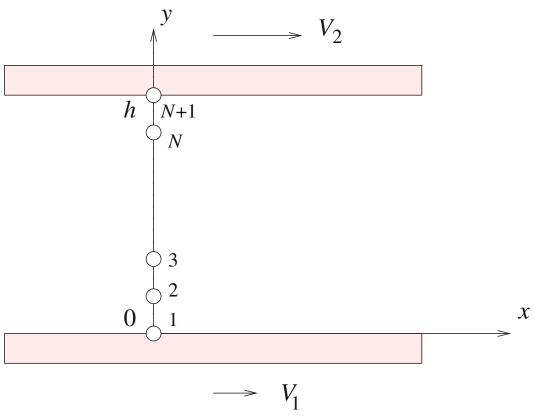


Figure 8.2.1 A one-dimensional finite-difference grid is used to compute the velocity profile in unidirectional channel flow.

8.2.1 Governing equations

To set up the mathematical formulation, we consider the x component of the equation of motion. In the case of unidirectional flow, we obtain the simplified form

$$\frac{\partial u_x}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\,\frac{\partial^2 u_x}{\partial y^2} + g_x, \tag{8.2.1}$$

where ρ is the fluid density, ν is the kinematic viscosity, and g_x is the x component of the acceleration of gravity. The partial differential equation (8.2.1) is to be solved subject to a specified initial condition and to the possibly time-dependent velocity boundary conditions

$$u_x(y=0) = V_1(t), \qquad u_x(y=h) = V_2(t), \tag{8.2.2}$$

enforcing no-slip at the walls.

8.2.2 Explicit finite-difference method

To implement the finite-difference method, we divide the cross-section of the channel extending over $0 \le y \le h$ into N intervals defined by $N+1$ grid points, as shown in Figure 8.2.1. For convenience, the x component of the velocity at the i grid point is denoted as

$$u_i(t) \equiv u_x(y_i, t). \tag{8.2.3}$$

Next, we evaluate both sides of (8.2.1) at the ith interior grid point at time t for $i = 2, \ldots, N$, and approximate the time derivative on the left-hand side with a first-order forward finite difference and the second derivative on the right-hand side with a second-order centered finite difference. The result is a finite-difference equation (FDE),

$$\frac{u_i(t+\Delta t) - u_i(t)}{\Delta t} = -\frac{1}{\rho}\frac{\partial p}{\partial x}(t) + \nu\,\frac{u_{i-1}(t) - 2\,u_i(t) + u_{i+1}(t)}{\Delta y^2} + g_x. \tag{8.2.4}$$

Solving for $u_i(t+\Delta t)$ on the left-hand side, we obtain

$$u_i(t+\Delta t) = \alpha\, u_{i-1}(t) + (1-2\,\alpha)\, u_i(t) + \alpha\, u_{i+1}(t) + \Delta t\,\Big(-\frac{1}{\rho}\frac{\partial p}{\partial x}(t) + g_x \Big) \tag{8.2.5}$$

for $i = 2, \ldots, N$. We have introduced the dimensionless ratio

$$\alpha \equiv \frac{\nu \Delta t}{\Delta y^2}, \tag{8.2.6}$$

called the *numerical diffusion number.*

Equation (8.2.5) allows us to update the velocity at the interior grid points explicitly, starting from the specified initial condition, subject to the prescribed boundary conditions

$$u_1(t) = V_1(t), \qquad\qquad u_{N+1}(t) = V_2(t). \tag{8.2.7}$$

The following MATLAB code entitled *channel_ftcs*, located in directory *channel* inside directory *11_fdm* of FDLIB, performs the animation of the evolving velocity profile:

```
%----
% parameters
%----

h = 1.0;
mu = 0.6; rho = 0.5;
N = 32;
dpdx = -2.0; gx = 0.4;
V1 = 0.0; V2 = 0.0;
al = 0.51;     % alpha
nstep = 20000;     % number of steps

%---
% prepare
%---

nu = mu/rho;     % kinematic viscosity
Dy = h/N;
Dt = al*Dy*Dy/nu;

%---
% grid and initial condition
%---

for i=1:N+1
  y(i) = (i-1)*Dy;
  u(i) = 0;
end
u(1) = V1;
u(N+1) = V2;
```

```
t = 0.0;

%---
% time stepping
%---

for step=1:nstep

  t = t + Dt;

  unew(1) = V1;
  for i=2:N
    unew(i) = al*u(i-1) + (1-2*al)*u(i) + al*u(i+1) ...
      + Dt*(-dpdx/rho+gx);
  end
  unew(N+1) = V2;
  u = unew;

  if(step==1)
    Handle1 = plot(u,y,'o-');
    set(Handle1, 'erasemode', 'xor');
    set(gca,'fontsize',15)
    axis([0 0.5 0 h])
    xlabel('u','fontsize',15)
    ylabel('y','fontsize',15)
  else
    set(Handle1,'XData',u,'YData',y);
    pause(0.02)
    drawnow
  end

end     % of time stepping
```

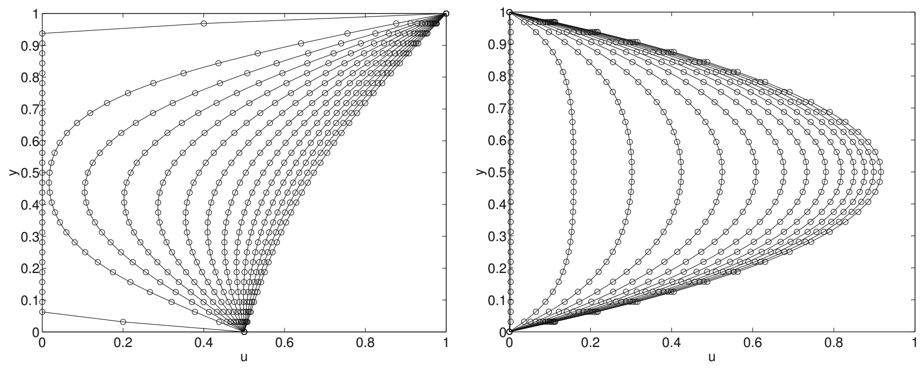
Evolving profiles are shown in Figure 8.2.2 for two values of the dimensionless numerical parameter α.

Numerical stability

Numerical experimentation reveals, and theoretical analysis confirms, that the explicit method of updating the velocity based on equation (8.2.5) is free of oscillations only when the time step, Δt, is small enough so that the dimensionless numerical diffusion number α defined in (8.2.6) is less than $\frac{1}{2}$.[1] For larger time steps, the velocity profile develops unphysical growing numerical oscillations unrelated to the physics of the motion, as illustrated in Figure 8.2.2(*b*) for $\alpha = 0.51$. We say that the explicit finite-difference method is *conditionally stable.*

[1] Pozrikidis, C. (2008) *Numerical Computation in Science and Engineering.* Second Edition, Oxford University Press.

(a)



(b)

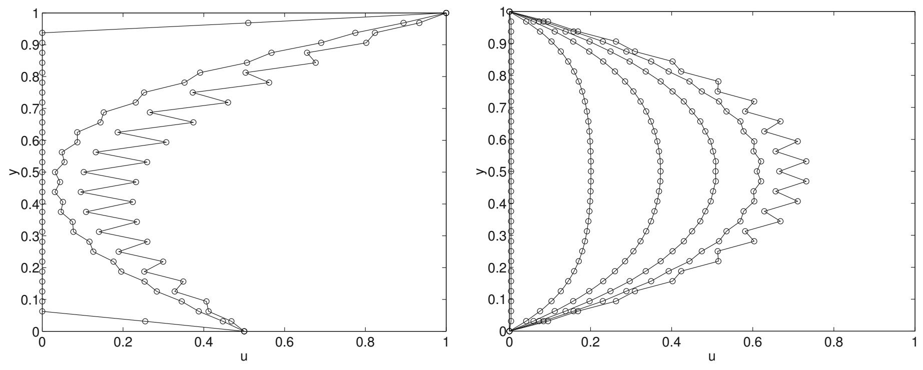


Figure 8.2.2 Evolving profiles of unidirectional flow in a channel computed by an explicit finite-difference method for numerical diffusion number (a) $\alpha = 0.40$ and (b) 0.51. The scary oscillations in the second case are a manifestation of numerical instability.

8.2.3 Implicit finite-difference method

To avoid the restriction on the time step for numerical stability, we implement an *implicit* finite-difference method. Evaluating equation (8.2.1) at the ith interior grid point at time $t + \Delta t$ for $i = 2, \ldots, N$, and then approximating the time derivative on the left-hand side with a first-order backward finite difference and the second derivative on the right-hand side with a second-order centered finite difference, we obtain the difference equation

$$\frac{u_i(t+\Delta t) - u_i(t)}{\Delta t} = -\frac{1}{\rho}\,\frac{\partial p}{\partial x}(t+\Delta t) + \nu\,\frac{u_{i+1}(t+\Delta t) - 2\,u_i(t+\Delta t) + u_{i-1}(t+\Delta t)}{\Delta y^2} + g_x. \tag{8.2.8}$$

Rearranging, we obtain

$$-\alpha\, u_{i-1}(t+\Delta t) + (1+2\,\alpha)\, u_i(t+\Delta t) - \alpha\, u_{i+1}(t+\Delta t) \\ = u_i(t) - \frac{\Delta t}{\rho}\frac{\partial p}{\partial x}(t+\Delta t) + \Delta t\, g_x, \qquad (8.2.9)$$

where α is the numerical diffusion number defined in (8.2.6), $\alpha \equiv \nu \Delta t/\Delta y^2$.

Equation (8.2.9) allows us to compute the velocity at the interior grid points at the time level $t + \Delta t$ in an implicit fashion, which means that we solve simultaneously for all unknown grid values, subject to the prescribed boundary conditions ,

$$u_1(t+\Delta t) = V_1(t+\Delta t), \qquad u_{N+1}(t+\Delta t) = V_2(t+\Delta t). \qquad (8.2.10)$$

To formalize the implicit solution algorithm, we write equation (8.2.9) for $i = 2, \ldots, N$ and enforce the boundary conditions to obtain a system of $N-1$ linear equations for the velocity at the $N-1$ interior grid points at time $t + \Delta t$,

$$\mathbf{A} \cdot \mathbf{u}(t+\Delta t) = \mathbf{u}(t) + \mathbf{b}. \qquad (8.2.11)$$

We have introduced the tridiagonal coefficient matrix

$$\mathbf{A} = \begin{bmatrix} 1+2\alpha & -\alpha & 0 & \cdots & 0 & 0 & 0 \\ -\alpha & 1+2\alpha & -\alpha & \cdots & 0 & 0 & 0 \\ 0 & -\alpha & 1+2\alpha & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1+2\alpha & -\alpha & 0 \\ 0 & 0 & 0 & \cdots & -\alpha & 1+2\alpha & -\alpha \\ 0 & 0 & 0 & \cdots & 0 & -\alpha & 1+2\alpha \end{bmatrix}, \qquad (8.2.12)$$

the vector of unknown velocities

$$\mathbf{u}(t+\Delta t) = \begin{bmatrix} u_2(t+\Delta t) \\ u_3(t+\Delta t) \\ \vdots \\ u_{N-1}(t+\Delta t) \\ u_N(t+\Delta t) \end{bmatrix}, \qquad (8.2.13)$$

and the known vectors

$$\mathbf{u}(t) = \begin{bmatrix} u_2(t) \\ u_3(t) \\ \vdots \\ u_{N-1}(t) \\ u_N(t) \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} -\frac{\Delta t}{\rho}\frac{\partial p}{\partial x}(t+\Delta t) + \Delta t\, g_x + \alpha\, V_1(t+\Delta t) \\ -\frac{\Delta t}{\rho}\frac{\partial p}{\partial x}(t+\Delta t) + \Delta t\, g_x \\ \vdots \\ -\frac{\Delta t}{\rho}\frac{\partial p}{\partial x}(t+\Delta t) + \Delta t\, g_x \\ -\frac{\Delta t}{\rho}\frac{\partial p}{\partial x}(t+\Delta t) + \Delta t\, g_x + \alpha\, V_2(t+\Delta t) \end{bmatrix}. \qquad (8.2.14)$$

The numerical method involves solving the linear system (8.2.11) at the current time instant, t, to obtain the velocity profile at the next time instant, $t + \Delta t$, beginning from a specified initial state. The tridiagonal structure of the matrix $\mathbf{A}$ displayed in (8.2.12) allows us to compute the solution efficiently using the legendary Thomas algorithm discussed in Section 8.2.4.

Finite-difference code

The following MATLAB code entitled *channel_btcs,* located in directory *channel* inside directory *11_fdm* of FDLIB, performs the time integration using the implicit method starting from a specified initial velocity profile:

```
%-----
% parameters
%-----

h = 1.0; mu = 0.6; rho = 0.5; N = 32; dpdx = -2.0;
gx = 0.4;
V1 = 0.1; V2 = -0.5;
alpha = 0.40;     % alpha
nstep = 20000;     % number of steps
nu = mu/rho;
Dy = h/N
Dt = alpha*Dy*Dy/nu;

%---
% grid and initial condition
%---

for i=1:N+1
  y(i) = (i-1)*Dy;
  u(i) = 0.0;
end
u(1) = V1;
u(N+1) = V2;

%---
% formulate the tridiagonal projection matrix
% atr is the diagonal line of the coefficient matrix
% btr is the superdiagonal line of the coefficient matrix
% ctr is the subdiagonal line of the coefficient matrix
%---

for i=1:N-1
  atr(i) = 1.0 + 2*alpha;
  btr(i) = -alpha;
  ctr(i) = -alpha;
end
```

```
%---
% time stepping
%---

t=0.0;

for step=1:nstep

  for i=1:N-1     % right-hand side
    s(i) = u(i+1) + Dt*(-dpdx/rho+gx);
  end

  t = t + Dt;
  u(1) = V1;
  u(N+1) = V2;
  s(1) = s(1) + alpha*u(1);
  s(N-1) = s(N-1) + alpha*u(N+1);

  sol = thomas(N-1,atr,btr,ctr,s);

  for i=2:N
    u(i) = sol(i-1);
  end

  if(step==1)
    Handle1 = plot(u,y,'o-');
    set(Handle1, 'erasemode', 'xor');
    set(gca,'fontsize',15)
    axis([min(V1,V2) max(V1,V2)+2.0 0 h])
    xlabel('u','fontsize',15)
    ylabel('y','fontsize',15)
  else
    set(Handle1,'XData',u,'YData',y);
    pause(0.2)
    drawnow
  end

end     % of time stepping
```

The code calls the function *thomas* discussed in Section 8.2.4 to solve a tridiagonal system of equations.

Numerical stability

Numerical experimentation reveals, and theoretical analysis confirms, that the implicit method of updating the velocity based on equation (8.2.11) is free of numerical oscillations irrespective of the size of the time step, Δt. Accordingly, the implicit finite-difference method is an unconditionally stable and thus highly desirable method.

8.2.4 Thomas algorithm

To formalize Thomas' algorithm in general terms, we consider a linear system of K equations in K unknowns,

$$\mathbf{D} \cdot \mathbf{x} = \mathbf{s}, \tag{8.2.15}$$

for an unknown vector, $\mathbf{x}$, where $\mathbf{s}$ is a given vector. The $K \times K$ coefficient matrix, $\mathbf{D}$, is assumed to have the tridiagonal form

$$\mathbf{D} = \begin{bmatrix} a_1 & b_1 & 0 & \cdots & 0 & 0 & 0 \\ c_2 & a_2 & b_2 & \cdots & 0 & 0 & 0 \\ 0 & c_3 & a_3 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & a_{K-2} & b_{K-2} & 0 \\ 0 & 0 & 0 & \cdots & c_{K-1} & a_{K-1} & b_{K-1} \\ 0 & 0 & 0 & \cdots & 0 & c_K & a_K \end{bmatrix}. \tag{8.2.16}$$

Note that only the diagonal, superdiagonal, and subdiagonal elements of $\mathbf{D}$ are nonzero. Thomas's algorithm proceeds in two stages.

In the first stage, the tridiagonal system (8.2.15) is transformed into an upper bidiagonal system,

$$\mathbf{D}' \cdot \mathbf{x} = \mathbf{y}, \tag{8.2.17}$$

involving an upper bidiagonal coefficient matrix with *ones* along the diagonal,

$$\mathbf{D}' = \begin{bmatrix} 1 & d_1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & d_2 & \cdots & 0 & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 1 & d_{K-2} & 0 \\ 0 & 0 & 0 & \cdots & 0 & 1 & d_{K-1} \\ 0 & 0 & 0 & \cdots & 0 & 0 & 1 \end{bmatrix}, \tag{8.2.18}$$

where $\mathbf{y}$ is an intermediate solution vector.

In the second stage, the upper bidiagonal system (8.2.17) is solved by backward substitution, which involves solving the last equation for the last unknown, $x_K = y_K$, and then moving upward to compute the rest of the unknowns in a sequential fashion.

The combined algorithm, shown in Table 8.2.1, is implemented in the following MATLAB function:

```
Reduction to bidiagonal :

    $\begin{bmatrix} d_1 \\ y_1 \end{bmatrix} = \frac{1}{a_1} \begin{bmatrix} b_1 \\ s_1 \end{bmatrix}$

    Do $i = 1, K-1$

      $\begin{bmatrix} d_{i+1} \\ y_{i+1} \end{bmatrix} = \frac{1}{a_{i+1} - c_{i+1} d_i} \begin{bmatrix} b_{i+1} \\ s_{i+1} - c_{i+1} y_i \end{bmatrix}$

    End Do

Backward substitution :
    $x_K = y_K$
    Do $i = K-1, 1 \quad (\text{step} = -1)$
      $x_i = y_i - d_i \, x_{i+1}$
    End Do
```

Table 8.2.1 Thomas algorithm for solving a system of K linear equations with a tridiagonal coefficient matrix.

```
function x = thomas (n,a,b,c,s)

%==================================================
% Thomas algorithm for a tridiagonal system
%
% n:  system size
% a,b,c:  diagonal, superdiagonal,
%         and subdiagonal elements
% s:  right-hand side
%==================================================

%------------------------------
% reduction to upper bidiagonal
%------------------------------

d(1) = b(1)/a(1);
y(1) = s(1)/a(1);

for i=1:n-2
  i1 = i+1;
  den = a(i1)-c(i1)*d(i);
  d(i1) = b(i1)/den;
  y(i1) = (s(i1)-c(i1)*y(i))/den;
```

```
end

den = a(n)-c(n)*d(n-1);
y(n) = (rhs(n)-c(n)*y(n-1))/den;

%------------------
% back substitution
%------------------

x(n) = y(n);

for i=n-1:-1:1
  x(i) = y(i)-d(i)*x(i+1);
end

%-----
% done
%-----

return;
```

In fact, the Thomas algorithm is a special implementation of the inclusive method of Gauss elimination discussed in Section 3.4.1 for a general system of linear equations. The key idea is to bypass idle multiplications by zeros.

8.2.5 Steady state

To obtain the velocity profile of channel flow at steady state, we return to equation (8.2.11) and set

$$\mathbf{u}(t+\Delta t) = \mathbf{u}(t) = \mathbf{u} \tag{8.2.19}$$

to obtain

$$(\mathbf{A} - \mathbf{I}) \cdot \mathbf{u} = \mathbf{b}, \tag{8.2.20}$$

where $\mathbf{I}$ is the unit matrix. Dividing the individual equations encapsulated in (8.2.20) by α, we obtain the simpler form

$$\mathbf{C} \cdot \mathbf{u} = \mathbf{d}, \tag{8.2.21}$$

involving the tridiagonal coefficient matrix

$$\mathbf{C} = \begin{bmatrix} 2 & -1 & 0 & \cdots & 0 & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 & 0 \\ 0 & 0 & 0 & \cdots & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & 0 & -1 & 2 \end{bmatrix}, \tag{8.2.22}$$

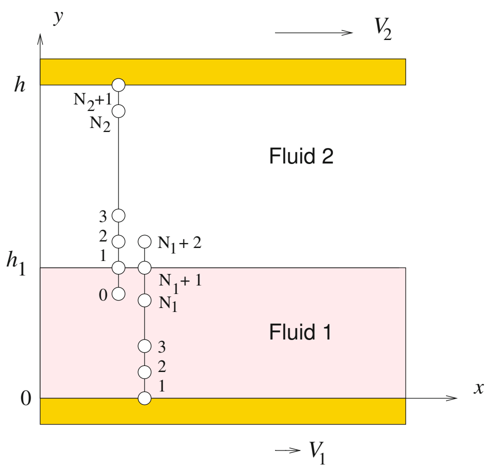


Figure 8.2.2 Illustration of a composite finite-difference grid with phantom nodes on either side an interface used to compute the velocity profile of unidirectional two-fluid channel flow. The interface is located at $y = h_1$.

the vector of unknown velocities at steady state

$$\mathbf{u} = \begin{bmatrix} u_2 \\ u_3 \\ \vdots \\ u_{N-1} \\ u_N \end{bmatrix}, \tag{8.2.23}$$

and the known vector

$$\mathbf{d} = \begin{bmatrix} -\frac{1}{\mu}\,\frac{\partial p}{\partial x} + \frac{\Delta y^2}{\nu}\,g_x + V_1 \\ \\ -\frac{1}{\mu}\,\frac{\partial p}{\partial x} + \frac{\Delta x^2}{\nu}\,g_x \\ \vdots \\ -\frac{1}{\mu}\,\frac{\partial p}{\partial x} + \frac{\Delta y^2}{\nu}\,g_x \\ \\ -\frac{1}{\mu}\,\frac{\partial p}{\partial x} + \frac{\Delta y^2}{\nu}\,g_x + V_2 \end{bmatrix}. \tag{8.2.24}$$

To compute the velocity profile at steady state, we simply solve the system of linear algebraic equations (8.2.21) using a numerical method.

8.2.6 Two-layer flow

Next, we consider the flow of two superimposed layers in a channel, as illustrated in Figure 8.2.2. The lower layer is labeled 1 and the upper layer is labeled 2. The fluids are separated by a flat interface located at $y = h_1$, where $h_1 < h$ is the lower-layer thickness and h is the channel width. The upper-layer thickness is $h_2 = h - h_1$.

Interfacial conditions

At the interface, we require three conditions: continuity of velocity, continuity of shear stress, and continuity of normal stress. To satisfy the third condition, we require that the streamwise pressure gradient, $\partial p/\partial x$, is the same inside both layers. Continuity of velocity at the interface requires that

$$u_x^{(1)}(y = h_1) = u_x^{(2)}(y = h_1) \tag{8.2.25}$$

and continuity of shear stress requires that

$$\mu_1 \left(\frac{\partial u_x^{(1)}}{\partial y} \right)_{y=h_1} = \mu_2 \left(\frac{\partial u_x^{(2)}}{\partial y} \right)_{y=h_1}. \tag{8.2.26}$$

where $u_x^{(1)}$ is the lower-layer velocity and $u_x^{(2)}$ is the upper-layer velocity. Using the equation of motion (8.2.1), we find that, if (8.2.25) is true at the initial instant, it will also be true at any time provided that

$$-\frac{1}{\rho_1}\frac{\partial p}{\partial x} + \nu_1 \left(\frac{\partial^2 u_x^{(1)}}{\partial y^2} \right)_{y=h_1} = -\frac{1}{\rho_2}\frac{\partial p}{\partial x} + \nu_2 \left(\frac{\partial^2 u_x^{(2)}}{\partial y^2} \right)_{y=h_1}, \tag{8.2.27}$$

where the second partial derivative are evaluated at the interface.

Finite-difference implementation

We begin developing the finite-difference method by dividing the lower layer into N_1 evenly spaced intervals defined by N_1+1 grid points, $y_i^{(1)}$ for $i = 1, \ldots, N_1+1$, and the upper layer into N_2 evenly spaced intervals defined by $N_2 + 1$ grid points, $y_i^{(2)}$ for $i = 1, \ldots, N_2 + 1$, as shown in Figure 8.2.2.

For reasons that will become apparent, we also extend the domain of definition of each layer into the adjacent layer by one artificial grid point labeled $N_1 + 2$ for the lower layer or 0 for the upper layer.

To simplify the notation, we denote

$$u_i^{(1)} \equiv u_x^{(1)}(y_i^{(1)}), \qquad u_i^{(2)} \equiv u_x^{(2)}(y_i^{(2)}). \tag{8.2.28}$$

Approximating the derivatives in (8.2.26) and (8.2.27) with centered finite differences, we derive two equations relating the values of the velocity at the extended nodes, $u_{N_1+2}^{(1)}$ and $u_0^{(2)}$,

$$\mu_1 \frac{u_{N_1+2}^{(1)} - u_{N_1}^{(1)}}{2\Delta y_1} = \mu_2 \frac{u_2^{(2)} - u_0^{(2)}}{2\Delta y_2} \tag{8.2.29}$$

and

$$-\frac{1}{\rho_1}\frac{\partial p}{\partial x} + \nu_1 \frac{u^{(1)}_{N_1+2} - 2\,u^{(1)}_{N_1+1} + u^{(1)}_{N_1}}{\Delta y_1^2} = -\frac{1}{\rho_2}\frac{\partial p}{\partial x} + \nu_2 \frac{u^{(2)}_2 - 2\,u^{(2)}_1 + u^{(2)}_0}{\Delta y_2^2}, \tag{8.2.30}$$

where $\Delta y_1 \equiv h_1/N_1$ and $\Delta y_2 \equiv h_2/N_2$ are the grid spacings. Setting $u^{(1)}_{N_1+1} = u^{(2)}_1$ and introducing the ratios

$$\lambda \equiv \frac{\mu_2}{\mu_1}, \qquad \delta \equiv \frac{\rho_2}{\rho_1}, \qquad \gamma \equiv \frac{\nu_2}{\nu_1} = \frac{\lambda}{\delta}, \qquad \beta \equiv \frac{\Delta y_2}{\Delta y_1}, \tag{8.2.31}$$

we recast equations (8.2.29) and (8.2.30) into a system of two linear equations for the velocity at the extended nodes,

$$\beta\, u^{(1)}_{N_1+2} + \lambda\, u^{(2)}_0 = \beta\, u^{(1)}_{N_1} + \lambda\, u^{(2)}_2 \tag{8.2.32}$$

and

$$\beta^2\, u^{(1)}_{N_1+2} - \gamma\, u^{(2)}_0 = 2\,(\beta^2-\gamma)\, u^{(1)}_{N_1+1} - \beta^2 u^{(1)}_{N_1} + \gamma\, u^{(2)}_2 + \frac{\Delta y_2^2}{\mu_1}\,(1-\frac{1}{\delta})\,\frac{\partial p}{\partial x}. \tag{8.2.33}$$

In matrix notation,

$$\begin{bmatrix} \beta & \lambda \\ -\beta^2 & \gamma \end{bmatrix} \cdot \begin{bmatrix} u^{(1)}_{N_1+2} \\ u^{(2)}_0 \end{bmatrix} = \begin{bmatrix} \beta\, u^{(1)}_{N_1} + \lambda\, u^{(2)}_2, \\ -2\,(\beta^2-\gamma)\, u^{(1)}_{N_1+1} + \beta^2 u^{(1)}_{N_1} - \gamma u^{(2)}_2 - \dfrac{\Delta y_2^2}{\mu_1}\,(1-\dfrac{1}{\delta})\,\dfrac{\partial p}{\partial x} \end{bmatrix}. \tag{8.2.34}$$

Solving for the velocity at the lower extended node, we find that

$$u^{(1)}_{N_1+2} = a_1\, u^{(1)}_{N_1} + a_2\, u^{(1)}_{N_1+1} + a_3\, u^{(2)}_2 + a_4\,\frac{\partial p}{\partial x}, \tag{8.2.35}$$

where

$$\begin{aligned} a_1 &= \frac{\gamma-\beta\lambda}{\gamma+\beta\lambda}, & a_2 &= 2\lambda\frac{\beta^2-\gamma}{\beta(\gamma+\beta\lambda)}, \\ a_3 &= \frac{2\,\gamma\lambda}{\beta(\gamma+\beta\lambda)}, & a_4 &= \frac{\lambda}{\beta(\gamma+\beta\lambda)}\,\frac{\Delta y_2^2}{\mu_1}\,\Big(1-\frac{1}{\delta}\Big) \end{aligned} \tag{8.2.36}$$

are four constants.

When the physical properties of the layers are matched, $\lambda = \gamma = \delta = 1$, and the lower and upper grid sizes are equal, $\beta = 1$, then $u^{(1)}_{N_1+2} = u^{(2)}_2$ by equation (8.2.35), and $u^{(2)}_0 = u^{(1)}_{N_1}$ by equation (8.2.32), as required.

Explicit time integration

Working as in the case of single-fluid flow, we derive the explicit finite-difference equation

$$u_i^{(1)}(t+\Delta t) = \alpha_1\, u_{i+1}^{(1)}(t) + (1-2\,\alpha_1)\, u_i^{(1)}(t) + \alpha_1\, u_{i-1}^{(1)}(t) - \Delta t\,\big(-\frac{1}{\rho_1}\frac{\partial p}{\partial x}(t) + g_x\,\big) \tag{8.2.37}$$

for the lower layer, and a corresponding equation for the upper layer,

$$u_i^{(2)}(t+\Delta t) = \alpha_2\, u_{i+1}^{(2)}(t) + (1-2\,\alpha_2)\, u_i^{(2)}(t) + \alpha_2\, u_{i-1}^{(2)}(t) - \Delta t\,\big(-\frac{1}{\rho_2}\frac{\partial p}{\partial x}(t) + g_x\,\big), \tag{8.2.38}$$

where

$$\alpha_1 \equiv \frac{\nu_1 \Delta t}{\Delta y_1^2}, \qquad \alpha_2 \equiv \frac{\nu_2 \Delta t}{\Delta y_2^2} \tag{8.2.39}$$

are the numerical diffusion numbers for the lower and upper layer. The numerical procedure involves the following steps:

1. Initialize the nodal velocities.
2. Compute the velocity at the lower extended node, $u_{N_1+2}^{(1)}$, from equation (8.2.35).
3. Use equation (8.2.38) to update the velocity at the grid points in the lower layer for $i = 2, \ldots, N_1+1$.
4. Set $u_1^{(2)} = u_{N_1+1}^{(1)}$.
5. Use equation (8.2.39) to update the velocity at the internal grid nodes in the upper layer for $i = 2, \ldots, N_2$.
6. Use the boundary conditions to update the velocity at the lower and upper walls.
7. Return to Step 2 and repeat the computation for another step.

The method is implemented in the following MATLAB code entitled *two_layers*, located in directory *channel* inside directory *11_fdm* of FDLIB, performing the animation of the developing velocity profile:

```
%---
% parameters
%---

h = 1.0; h1 = 0.25;
N1 = 4; N2 = 32;
mu1 = 1.0; mu2 = 2.0;
rho1 = 1.5; rho2 = 1.0;
dpdx = -1.0; gx = 0.2;
```

```
alpha = 0.4;

%---
% prepare
%---

h2 = h-h1;
Dy1 = h1/N1;
Dy2 = h2/N2;

lambda = mu2/mu1;
delta = rho2/rho1;
gamma = lambda/delta;
beta = Dy2/Dy1;

tmp = gamma+beta*lambda;
a1 = (gamma-beta*lambda)/tmp;
a2 = 2*lambda*(beta*beta-gamma)/(beta*tmp);
a3 = 2*gamma*lambda/(beta*tmp);
a4 = lambda*Dy2*Dy2*(1-1/delta)/(beta*mu1*tmp);

Dt1 = rho1*alpha*Dy1*Dy1/mu1;
Dt2 = rho2*alpha*Dy2*Dy2/mu2;

Dt = min(Dt1,Dt2)

al1 = Dt*mu1/(Dy1*Dy1*rho1);
al2 = Dt*mu2/(Dy2*Dy2*rho2);

%---
% initialize and define the grid
%---

for i=1:N1+1
  u1(i) = 0.0;
  y1(i) = (i-1)*Dy1;
end

for i=1:N2+1
  u2(i) = 0.0;
  y2(i) = (i-1)*Dy2+h1;
end

%---
% time stepping
%---

for step=1:1000
```

```
  u1(N1+2) = a1*u1(N1)+a2*u1(N1+1)+a3*u2(2)+a4*dpdx;

  unew1(1) = V1;

  for i=2:N1+1
    unew1(i) = al1*u1(i+1)+(1-2*al1)*u1(i)+al1*u1(i-1)...
      +Dt*(-dpdx/rho1+gx);
  end

  unew2(1) = unew1(N1+1);

  for i=2:N2
    unew2(i) = al2*u2(i+1)+(1-2*al2)*u2(i)+al2*u2(i-1)...
      +Dt*(-dpdx/rho2+gx);
  end

  unew2(N2+1) = V2;

  u1 = unew1; u2 = unew2;

%---
 % animation
%---

  if(step==1)
    handle1 = plot(u1,y1,'o-',u2,y2,'o-');
    set(gca,'fontsize',15)
    axis([0 0.1 0 h])
    xlabel('u','fontsize',15)
    ylabel('y','fontsize',15)
  else
    set(handle1,'XData',[u1, u2],'YData',[y1, y2],'Marker','o')
    drawnow
    pause(0.01)
  end

%---
end     % of time stepping
%---
```

Snapshots of an evolving profile are shown in Figure 8.2.3 for two values of α defined as the minimum of α_1 and α_2.

Problems

8.2.1 *Steady state*

Derive the system (8.2.20) departing from the explicit finite-difference formula (8.2.4).

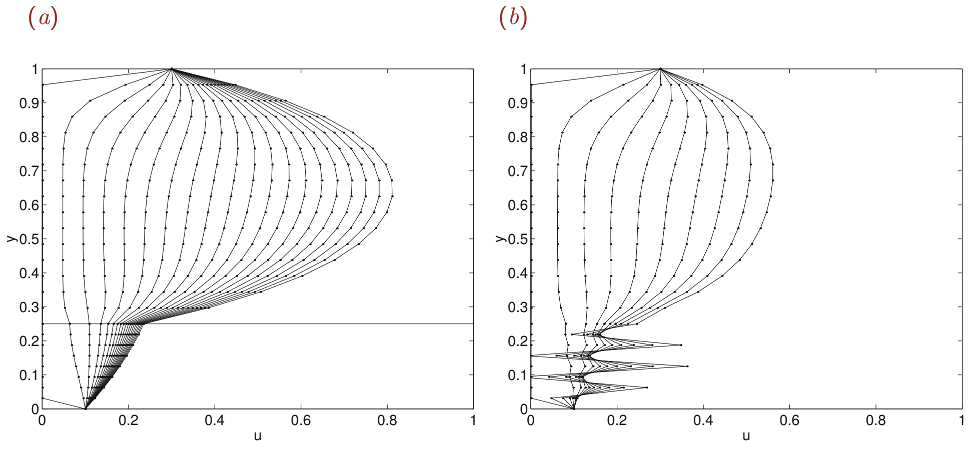


Figure 8.2.3 Evolving profiles of two-layer flow in a channel, computed by an explicit finite difference method for (a) $\alpha = 0.4$ and (b) 0.52, where α is the minimum of α_1 and α_2. A numerical instability arises in the second case.

8.2.2 *Two-layer channel flow*

Derive a system of finite-difference equations governing the velocity profile of a two-layer channel flow at steady state.

8.2.3 *Flow in a circular tube*

Develop an explicit finite-difference method based on the velocity/pressure formulation for computing the velocity profile developing inside a tube with circular cross-section due to a suddenly imposed constant pressure gradient.

8.2.4 *Thomas algorithm*

Use the MATLAB function *thomas* listed in the text to solve a system of equations of your choice. Verify the accuracy of the solution by confirming that it satisfies the tested system of equations.

8.2.5 *Single-fluid channel flow*

(a) Write a code that computes the evolution of the velocity profile in a channel with stationary walls due to a sinusoidal pressure gradient based on the explicit finite-difference method discussed in the text. Run the program for fluid properties and flow conditions of your choice, and for several time step sizes, Δt, corresponding to numerical diffusion number α that is larger and lower than 0.5. Discuss the performance of the numerical method.

(b) Repeat (a) for the implicit finite-difference method discussed in the text.

8.2.6 *Two-layer channel flow*

Develop an implicit finite-difference method for computing the evolution of a two-layer flow. Implement the method in a program that computes the evolution of the velocity profile in a channel with stationary walls due to the sudden application of a constant pressure gradient. Run the program for fluid properties and flow conditions of your choice and for several time step sizes. Discuss the performance of the numerical method.

8.3 Unidirectional flow; velocity/vorticity formulation

The vorticity transport equation for unsteady unidirectional flow along the x axis reduces to the unsteady diffusion equation for the z component of the vorticity, ω_z,

$$\frac{\partial \omega_z}{\partial t} = \nu \, \frac{\partial^2 \omega_z}{\partial y^2}, \tag{8.3.1}$$

where ν is the kinematic viscosity of the fluid. Invoking the definition of the vorticity, $\boldsymbol{\omega} = \boldsymbol{\nabla} \times \mathbf{u}$, we find that

$$\omega_z = -\frac{\partial u_x}{\partial y}. \tag{8.3.2}$$

Integrating equation (8.3.2) with respect to y from the lower wall up to an arbitrary point, we obtain an integral representation for the velocity in terms of the vorticity,

$$u_x(y) = V_1 - \int_0^y \omega_z(y') \, \mathrm{d}y'. \tag{8.3.3}$$

Without loss of generality, we have chosen to satisfy the boundary condition at the lower wall located at $y = 0$, requiring that $u_x(0) = V_1$. It remains to ensure that the no-slip boundary condition is also satisfied at the upper wall.

The numerical method involves computing the evolution of the vorticity profile from a specified initial state using (8.3.1), while simultaneously recovering the evolution of the velocity field based on equation (8.3.2) or its integrated version shown in (8.3.3). Since the velocity does not appear in equation (8.3.1), the two steps are decoupled.

8.3.1 Boundary conditions for the vorticity

Because the unsteady diffusion equation (8.3.1) is a second-order differential equation with respect to y, two boundary conditions for the vorticity are required, one at each end of the solution domain located at $y = 0$ and h. The boundary conditions must be such that the integral constraint

$$\int_0^h \omega_z(\eta) \, \mathrm{d}\eta = V_1 - V_2 \tag{8.3.4}$$

is satisfied so that the right-hand side of (8.3.3) is consistent with the upper-wall no-slip boundary condition $u_x(y = h) = V_2$, and either the flow rate through the channel has a prescribed value, $Q(t)$, or the streamwise pressure gradient has a prescribed value $\partial p(t)/\partial x$.

Concentrating on flow subject to a specified pressure gradient, we recast the x component of the equation of motion for unidirectional flow,

$$\frac{\partial u_x}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial x} + \nu\,\frac{\partial^2 u_x}{\partial y^2} + g_x, \tag{8.3.5}$$

into the form

$$\frac{\partial u_x}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial x} - \nu\,\frac{\partial \omega_z}{\partial y} + g_x. \tag{8.3.6}$$

Evaluating (8.3.6) at the lower and upper walls and rearranging, we obtain boundary conditions for the slope of the vorticity,

$$\Big(\frac{\partial \omega_z}{\partial y}\Big)_{y=0} = -\frac{1}{\nu}\frac{\mathrm{d}V_1}{\mathrm{d}t} - \frac{1}{\mu}\frac{\partial p}{\partial x} + \frac{1}{\nu}\,g_x \tag{8.3.7}$$

and

$$\Big(\frac{\partial \omega_z}{\partial y}\Big)_{y=h} = -\frac{1}{\nu}\frac{\mathrm{d}V_2}{\mathrm{d}t} - \frac{1}{\mu}\frac{\partial p}{\partial x} + \frac{1}{\nu}\,g_x. \tag{8.3.8}$$

These equations provide us with Neumann boundary conditions at either end of the solution domain.

Special attention must be paid to the case of impulsive motion. If a wall is set in motion suddenly in an impulsive fashion with the velocity changing from one value to another over an infinitesimal period of time, the corresponding time derivative on the right-hand side of one or both of equations (8.3.7) and (8.3.8) develops an infinite spike described by the Dirac delta function discussed in Chapter 11. This singular behavior is too demanding to be handled by the numerical method.

Next, we investigate whether the vorticity boundary conditions (8.3.7) and (8.3.8) ensure the satisfaction of the integral constraint (8.3.4), which is necessary for the satisfaction of the no-slip boundary condition at the upper wall. Integrating (8.3.1) with respect to y across the channel height, from 0 to h, interchanging the order of the integration and time differentiation on the left-hand side, and using (8.3.7) to simplify the right-hand side, we obtain

$$\frac{\mathrm{d}}{\mathrm{d}t}\int_0^h \omega_z(y')\,\mathrm{d}y' = \frac{\mathrm{d}}{\mathrm{d}t}(V_1 - V_2). \tag{8.3.9}$$

Time-integration of (8.3.9) reproduces (8.3.4) up to a time-independent constant determined by the initial state. Thus, if (8.3.4) is satisfied at the initial instant, it will also be satisfied at any subsequent time.

8.3.2 Alternative set of equations

In an alternative approach, we take the derivative of (8.3.2) with respect to y to derive the second-order equation,

$$\frac{\partial^2 u_x}{\partial y^2} = -\frac{\partial \omega_z}{\partial y} \equiv -q, \tag{8.3.10}$$

where $q \equiv \partial\omega_z/\partial y$ is the slope of the vorticity. To compute the velocity, we integrate the second-order equation (8.3.10) with respect to y using the velocity boundary conditions $u_x(y=0) = V_1$ and $u_x(y=h) = V_2$.

The important benefit stemming from using (8.3.10) instead of (8.3.2), is that, in order to compute the velocity, the slope of the vorticity q, instead of the vorticity itself, is required. An evolution equation for q arises by differentiating both sides of (8.3.1) with respect to y, finding that

$$\frac{\partial q}{\partial t} = \nu\,\frac{\partial^2 q}{\partial y^2}. \tag{8.3.11}$$

Boundary conditions are provided by equations (8.3.7) and (8.3.8).

In summary, the numerical procedure involves integrating in time equation (8.3.11) from an initial state subject to the derived boundary conditions (8.3.7) and (8.3.8), while simultaneously computing the velocity profile by solving the second-order equation (8.3.10) subject to the velocity boundary conditions, $u_x(y=0) = V_1$ and $u_x(y=h) = V_2$.

Explicit finite-difference method

To implement a finite-difference method, we divide the flow domain $0 \le y \le h$ into N intervals separated by $N+1$ grid points, as shown in Figure 8.2.1, and evaluate equation (8.3.11) at time t at the interior nodes for $i = 2, \ldots, N$.

Approximating the time derivative on the left-hand side with a first-order finite difference and the y derivative on the left-hand side with a second-order finite difference, we obtain

$$\frac{q_i(t+\Delta t) - q_i(t)}{\Delta t} = \nu\,\frac{q_{i-1}(t) - 2\,q_i(t) + q_{i+1}(t)}{\Delta y^2}, \tag{8.3.12}$$

where we have defined

$$q_i \equiv q(y_i). \tag{8.3.13}$$

Solving for $q_i(t+\Delta t)$ on the left-hand side, we obtain

$$q_i(t+\Delta t) = \alpha\,q_{i-1}(t) + (1-2\,\alpha)\,q_i(t) + \alpha\,q_{i+1}(t), \tag{8.3.14}$$

where $\alpha \equiv \nu\,\Delta t/\Delta y^2$ is the numerical diffusion number. Equation (8.3.14) allows us to update explicitly the values of q at the grid points subject to boundary conditions for q_1 and q_{N+1} given by the right-hand sides of equations (8.3.7) and (8.3.8).

The centered-difference discretization of equation (8.3.10) leads us to the linear system (8.2.21), where the coefficient matrix $\mathbf{C}$ is given in (8.2.22) and the constant vector on the right-hand side is given by

$$\mathbf{d} = \begin{bmatrix} \Delta y^2\, q_2 + V_1 \\ \Delta y^2\, q_3 \\ \vdots \\ \Delta y^2\, q_{N-1} \\ \Delta y^2\, q_N + V_2 \end{bmatrix}. \tag{8.3.15}$$

The linear system can be solved efficiently using the Thomas algorithm.

8.3.3 Comparison with the velocity/pressure formulation

Comparing the velocity/vorticity formulation with the velocity/pressure formulation discussed in Section 8.2, we find that the latter is significantly simpler in conception and implementation. While this is undoubtedly true in the case of unidirectional flow presently considered, the vorticity–velocity formulation is more competitive in the more general case of two- and three-dimensional flow.

Problems

8.3.1 *Steady flow*

Discuss the implementation of the velocity/vorticity formulation for steady channel flow due to a specified pressure gradient.

8.3.2 *Two-layer flow*

Develop a velocity/vorticity formulation for two-layer channel flow discussed in Section 8.2.

8.3.3 *Flow in a circular tube*

Develop an explicit method based on the velocity/vorticity formulation for computing the velocity profile developing inside a circular tube due to a suddenly imposed constant pressure gradient.

8.3.4 *Explicit finite-difference method*

Write a code that computes the evolution of the velocity profile in a channel with stationary walls due to the sudden application of a constant pressure gradient based on the explicit finite-difference method discussed in the text. Run the program for fluid properties and flow conditions of your choice and several time step sizes corresponding to numerical diffusion number α that is higher or lower than the critical threshold, 0.5. Discuss the performance of the numerical method.

8.4 Unidirectional flow; stream function/vorticity formulation

In Sections 8.2 and 8.3, we discussed methods of computing the evolution of the velocity profile in channel flow subject to a specified pressure gradient. In this section, we consider the complementary case of flow subject to a specified flow rate and develop a numerical method based on the velocity/vorticity formulation.

For reasons that will become apparent, we introduce the stream function, ψ, satisfying the equation

$$u_x = \frac{\partial \psi}{\partial y}. \tag{8.4.1}$$

In the case of unidirectional flow, u_x, and thus ψ, is a function of position, y, and time, t. The flow rate across a line that begins and ends at two parallel planes located at $y = y_1$ and y_2 is equal to the difference in the corresponding values of the stream function, $Q_{12} = \psi(y_2) - \psi(y_1)$. The flow rate through the entire channel height is $Q = \psi(y = h) - \psi(y = 0)$.

Using equation (8.3.2), we find that the nonzero vorticity component is related to the stream function by the equation

$$\omega_z = -\frac{\partial^2 \psi}{\partial y^2}. \tag{8.4.2}$$

The numerical method involves computing the evolution of the vorticity profile from a specified initial state using the vorticity transport equation (8.3.1), while simultaneously recovering the evolution of the stream function using the one-dimensional Poisson equation (8.4.2). Since the differential equations (8.3.1) and (8.4.2) are of second order with respect to y, two boundary conditions for the vorticity and two boundary conditions for the stream function are required, one at each end of the solution domain, $y = 0$ and h.

Since adding an arbitrary constant to the stream function does not affect the velocity, the base level of the stream function can be specified at will. Accordingly, we may stipulate that $\psi(y = 0) = 0$, finding

$$\psi(y = h) = Q(t). \tag{8.4.3}$$

It is now evident that, by introducing the stream function, we have facilitated the implementation of the condition on the flow rate.

8.4.1 Boundary conditions for the vorticity

The boundary conditions for the vorticity must involve the specified wall velocities, V_1 and V_2, by way of the no-slip boundary condition. To illustrate the implementation of this condition, we divide the flow domain, $0 \le y \le h$, into N intervals defined by $N + 1$ grid points, as shown in Figure 8.2.1, and evaluate (8.4.2) at time t at the boundary nodes corresponding to $i = 1$ and $N + 1$. To simplify the notation, we denote

$$\omega_i \equiv \omega_z(y_i), \qquad \psi_i \equiv \psi(y_i). \tag{8.4.4}$$

Approximating the y derivative on the right-hand side with a combination of finite differences, we obtain

$$\omega_1 = -\frac{(\frac{\partial \psi}{\partial y})_{\frac{1}{2}(y_1+y_2)} - (\frac{\partial \psi}{\partial y})_{y_1}}{\frac{1}{2}\,\Delta y} = -2\,\frac{\frac{\psi_2 - \psi_1}{\Delta y} - V_1}{\Delta y} \tag{8.4.5}$$

or

$$\omega_1 = 2\,\frac{\psi_1 - \psi_2}{\Delta y^2} + 2\,\frac{V_1}{\Delta y}, \tag{8.4.6}$$

and

$$\omega_{N+1} = -\frac{(\frac{\partial \psi}{\partial y})_{y_{N+1}} - (\frac{\partial \psi}{\partial y})_{\frac{1}{2}(y_N + y_{N+1})}}{\frac{1}{2}\,\Delta y} = -2\,\frac{V_2 - \frac{\psi_{N+1} - \psi_N}{\Delta y}}{\Delta y} \tag{8.4.7}$$

or

$$\omega_{N+1} = 2\,\frac{\psi_{N+1} - \psi_N}{\Delta y^2} - 2\,\frac{V_2}{\Delta y}. \tag{8.4.8}$$

It is somewhat distressing to realize that the no-slip condition is implemented indirectly in terms of the vorticity. Specifically, it is not clear that solving (8.4.2) for the stream function and subsequently differentiating it to recover the velocity generates a velocity profile that is consistent with the prescribed boundary velocity. However, a thorough analysis of the numerical method reveals that this is the case indeed, except under unusual circumstances associated with singular boundary conditions involving discontinuous functions.

8.4.2 A semi-implicit method

Proceeding with the finite-difference implementation, we apply equation (8.3.1) at the interior nodes corresponding to $i = 2, \ldots, N$ at time t. Approximating the time derivative on the left-hand side with a first-order finite difference and the y derivative on the right-hand side with a second-order second finite difference, we obtain

$$\frac{\omega_i(t + \Delta t) - \omega_i(t)}{\Delta t} = \nu\,\frac{\omega_{i-1}(t) - 2\,\omega_i(t) + \omega_{i+1}(t)}{\Delta y^2}. \tag{8.4.9}$$

Solving for $\omega_i(t + \Delta t)$, we obtain

$$\omega_i(t + \Delta t) = \alpha\,\omega_{i+1}(t) + (1 - 2\,\alpha)\,\omega_i(t) + \alpha\,\omega_{i-1}(t), \tag{8.4.10}$$

where $\alpha \equiv \nu \Delta t / \Delta y^2$ is the numerical diffusion number. Equation (8.4.10) allows us to explicitly update the values of the vorticity at the interior grid points, subject to boundary conditions for ω_1 and ω_{N+1} given by the right-hand sides of (8.4.6) and (8.4.8); the stream function at time t is assumed to be known.

The implicit discretization of equation (8.4.2) leads us to a linear system,

$$\mathbf{C} \cdot \boldsymbol{\psi}(t + \Delta t) = \mathbf{d}(t + \Delta t), \tag{8.4.11}$$

where the coefficient matrix $\mathbf{C}$ is given in (8.2.22), the vector $\boldsymbol{\psi}$ is defined as

$$\boldsymbol{\psi} = \begin{bmatrix} \psi_2 \\ \psi_3 \\ \vdots \\ \psi_{N-1} \\ \psi_N \end{bmatrix}, \tag{8.4.12}$$

and the vector on the right-hand side of (8.4.11) is given by

$$\mathbf{d} = \begin{bmatrix} \Delta y^2 \, \omega_2 \\ \Delta y^2 \, \omega_3 \\ \vdots \\ \Delta y^2 \, \omega_{N-1} \\ \Delta y^2 \, \omega_N + Q \end{bmatrix}. \tag{8.4.13}$$

The system (8.4.11) can be solved efficiently using the Thomas algorithm.

The numerical method involves the following steps:

1. Assign initial values to the stream function and vorticity at all nodes.
2. Compute the vorticity at the boundary nodes using equations (8.4.6) and (8.4.8).
3. Update the vorticity at the internal nodes using equation (8.4.10).
4. Update the stream function at the interior nodes by solving the linear system (8.4.11) for a known right-hand side.
5. Return to Step 2 and repeat the calculation for another step.

The velocity profile arises by numerically differentiating the stream function with respect to y.

Problems

8.4.1 *Steady flow*

Develop a finite-difference method based on the stream function/vorticity formulation for computing the velocity profile of steady channel flow subject to a specified flow rate.

8.4.2 *Two-layer flow*

Develop a finite-difference method based on the stream function/vorticity formulation for unsteady two-layer channel flow discussed in Section 8.2.

8.4.3 *Flow inside a circular tube*

Develop a finite-difference method based on the stream function/vorticity formulation for computing the velocity profile developing inside a circular tube, subject to a specified flow rate.

8.4.4 💻 *Explicit finite-difference method*

Write a code that computes the evolution of the velocity profile in a channel with stationary walls using an explicit finite-difference method. The flow rate increases gradually toward a steady value according to the equation

$$Q(t) = Q_0 \left(1 - \exp(-\beta \, \frac{\nu t}{h^2})\right), \tag{8.4.14}$$

where Q_0 is the constant flow rate prevailing of long times, a and β is a dimensionless constant. Run the program for fluid properties and flow conditions of your choice and for several sizes of the time step corresponding to α higher or lower than 0.5. Discuss the performance of the numerical method.

8.5 Two-dimensional flow; stream function/vorticity formulation

Having discussed finite-difference methods for unidirectional flow, now we turn our attention to the more general case of two-dimensional flow where further issues concerning the satisfaction of the continuity equation and choice of boundary conditions are encountered. In this section we discuss the stream function/vorticity formulation as an extension of the corresponding formulation for unidirectional flow discussed in Section 8.4.

Hello world

Texts on computer language programming introduce elementary programming procedures traditionally by explaining the structure of a program entitled *world*, which prints the important message *Hello World.* Texts on computational fluid dynamics (CFD) explain numerical methods by discussing the prototypical example of flow in a two-dimensional cavity driven by a moving lid, known as the *driven-cavity flow.* We will follow this time-honored tradition.

8.5.1 Flow in a cavity

Consider flow in a cavity driven by a lid that translates parallel to itself with a generally time-dependent velocity, $V(t)$, as illustrated in Figure 8.5.1.

We begin developing the stream function/vorticity formulation by introducing the stream function, ψ, defined from the differential relations

$$u_x = \frac{\partial \psi}{\partial y}, \qquad u_y = -\frac{\partial \psi}{\partial x}, \tag{8.5.1}$$

where u_x and u_y are the x and y velocity components. The no-penetration boundary condition requires that the component of the velocity normal to each of the four walls is zero. In terms of the stream function,

$$\psi = 0 \quad \text{over all walls}, \tag{8.5.2}$$

so that the tangential derivative of the stream function, which is equal to the normal component of the velocity, is also zero. The zero on the right-hand side of (8.5.2) could have been replaced by an arbitrary constant without consequences on the numerical solution or physical structure of the flow.

The no-slip boundary condition requires that the tangential component of the velocity is zero at the bottom, left, and right walls, and equal to $V(t)$ at the upper wall. In terms of

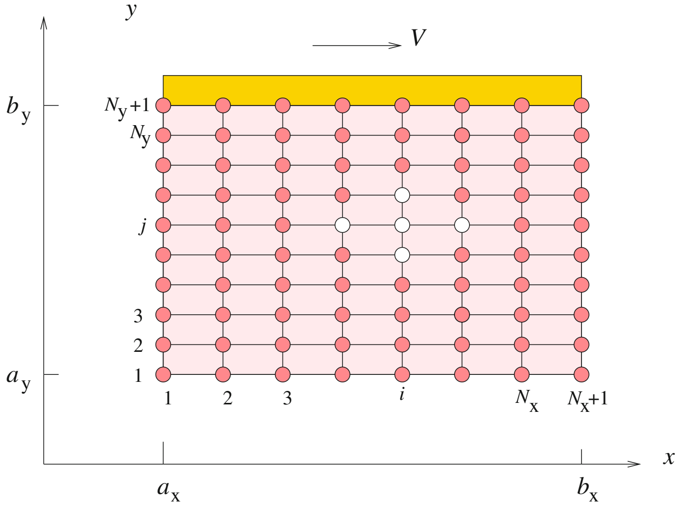


Figure 8.5.1 Illustration of a finite-difference grid used to compute flow in a cavity driven by a sliding lid.

the stream function,

$$\frac{\partial \psi}{\partial y} = 0 \quad \text{at the bottom}, \qquad \frac{\partial \psi}{\partial x} = 0 \quad \text{at the sides}, \tag{8.5.3}$$

$$\frac{\partial \psi}{\partial y} = V(t) \quad \text{at the top.}$$

Enforcing the boundary conditions for the velocity, we derive simplified expressions for the boundary values of the only non-vanishing vorticity component in terms of the stream function,

$$\omega_z \equiv -\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}. \tag{8.5.4}$$

For example, recalling that $u_y = 0$, and thus $\partial u_y/\partial x = 0$, over the bottom wall, we find that

$$\omega_z = -\frac{\partial u_x}{\partial y} = -\frac{\partial^2 \psi}{\partial y^2}. \tag{8.5.5}$$

Working in this fashion, we find that

$$\omega_z = -\frac{\partial^2 \psi}{\partial y^2} \quad \text{at the top and bottom} \tag{8.5.6}$$

and

$$\omega_z = -\frac{\partial^2 \psi}{\partial x^2} \quad \text{at the sides,} \tag{8.5.7}$$

which are simplified versions of the more general expression for the vorticity in terms of the stream function,

$$\omega_z = -\frac{\partial^2 \psi}{\partial x^2} - \frac{\partial^2 \psi}{\partial y^2}. \tag{8.5.8}$$

8.5.2 Finite-difference grid

To prepare the ground for the implementation of the finite-difference method, we cover the rectangular solution domain with a uniform two-dimensional Cartesian grid consisting of $N_x + 1$ uniformly spaced vertical lines and $N_y + 1$ uniformly spaced horizontal lines, as shown in Figure 8.5.1. Parallel grid lines are separated by intervals Δx or Δy, defining the grid size.

The intersections of grid lines define grid points or nodes labeled by a pair of integers, (i, j) for $i = 1, \ldots, N_x + 1$ and $j = 1, \ldots, N_y + 1$. The vertical side walls correspond to $i = 1$ and $N_x + 1$, and the bottom and top walls correspond to $j = 1$ and $N_y + 1$.

The goal of the finite-difference method is to generate values of flow variables of interest at the grid points. To simplify the notation, we denote

$$\omega_{i,j} \equiv \omega_z(x_i, y_j), \qquad \psi_{i,j} \equiv \psi(x_i, y_j). \tag{8.5.9}$$

Similar notation is used for other variables.

8.5.3 Unsteady flow

Following the general protocol of methods based on the vorticity transport equation, we compute the evolution of the flow by advancing the vorticity field using the vorticity transport equation for two-dimensional flow written in the form of an evolution equation for the vorticity,

$$\frac{\partial \omega_z}{\partial t} = -u_x \frac{\partial \omega_z}{\partial x} - u_y \frac{\partial \omega_z}{\partial y} + \nu \left(\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2}\right), \tag{8.5.10}$$

subject to appropriate derived boundary conditions for the vorticity, while simultaneously following the evolution of the stream function by solving the Poisson equation

$$\nabla^2 \psi \equiv \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega_z, \tag{8.5.11}$$

subject to specified boundary conditions for the stream function.

A simple method for computing the evolution of the flow when the lid starts translating suddenly involves the following steps:

1. At the initial instant, we set the stream function and velocity at all interior and boundary grid nodes to zero. Then we set the x component of the velocity at the grid nodes along the lid equal to $V(t = 0)$.

2. At the second step, we differentiate the velocity to obtain the vorticity using the definition $\omega_z \equiv -\partial u_x/\partial y + \partial u_y/\partial x$.

   For the interior grid points, we use centered differences to obtain

$$\omega_{i,j} = -\frac{(u_x)_{i,j+1} - (u_x)_{i,j-1}}{2\Delta y} + \frac{(u_y)_{i+1,j} - (u_y)_{i-1,j}}{2\Delta x}. \tag{8.5.12}$$

   For the top wall, we use backward differences to obtain

$$\omega_{i,N_y+1} \simeq -\Big(\frac{\partial u_x}{\partial y},\Big)_{i,N_y+1} \simeq \frac{-3\,V + 4\,(u_x)_{i,N_y} - (u_x)_{i,N_y-1}}{2\Delta y}, \tag{8.5.13}$$

   involving values at interior grid points.

   For the rest of the walls, we use forward or backward differences to obtain

$$\omega_{i,1} \simeq -(\frac{\partial u_x}{\partial y})_{i,1} \simeq \frac{-4\,(u_x)_{i,2} + (u_x)_{i,3}}{2\,\Delta y} \tag{8.5.14}$$

   for the bottom wall,

$$\omega_{1,j} \simeq (\frac{\partial u_y}{\partial x})_{1,j} \simeq \frac{4\,(u_y)_{2,j} - (u_y)_{3,j}}{2\,\Delta x} \tag{8.5.15}$$

   for the left wall, and

$$\omega_{N_x+1,j} \simeq (\frac{\partial u_y}{\partial x})_{N_x+1,j} \simeq \frac{-4\,(u_y)_{N_x,j} + (u_y)_{N_x-1,j}}{2\,\Delta x} \tag{8.5.16}$$

   for the right wall.

3. Now we integrate in time equation (8.5.10) to obtain the vorticity at the interior grid points at time $t + \Delta t$. Using a fully explicit method, we set

$$\omega_{i,j}(t + \Delta t) = \omega_{i,j}(t) + G_{(i,j)}(t), \tag{8.5.17}$$

   where $G_{(i,j)}(t)$ is the right-hand side of (8.5.10) evaluated at the (i, j) grid point at time t.

   To evaluate $G_{(i,j)}(t)$, we approximate the first spatial derivatives and the Laplacian of the vorticity using centered differences. For example, the Laplacian of the vorticity can be approximated with the finite-difference formula shown in equation (3.3.15), written for ω_z.

4. Next, we solve the Poisson equation (8.5.11) for the stream function, subject to the boundary condition $\psi = 0$, using a slightly generalized version of the finite-difference method for Laplace's equation discussed in Section 3.3.

Approximating the second derivatives with centered differences, we obtain the counterpart of equation (3.3.16),

$$\psi_{i+1,j} - 2\,(1+\beta)\,\psi_{i,j} + \psi_{i-1,j} + \beta\,\psi_{i,j+1} + \beta\,\psi_{i,j-1} = -\Delta x^2\,\omega_{i,j}, \tag{8.5.18}$$

where $\beta \equiv (\Delta x/\Delta y)^2$.

5. Finally, we differentiate the stream function to compute the velocity components at time $t + \Delta t$ at the interior grid points.

Having completed one time step, we return to Step 2 and repeat the computation for another time step.

8.5.4 Steady flow

To compute the steady flow, we follow a somewhat different approach. In this case, the left-hand side of (8.5.10) vanishes, yielding a differential relation between the velocity and the vorticity. Solving for the Laplacian of the vorticity, we obtain a Poisson equation for the vorticity forced by the *a priori* unknown source function on the right-hand side,

$$\frac{\partial^2 \omega_z}{\partial x^2} + \frac{\partial^2 \omega_z}{\partial y^2} = \frac{1}{\nu}\,\Big(u_x \frac{\partial \omega_z}{\partial x} + u_y \frac{\partial \omega_z}{\partial y}\Big). \tag{8.5.19}$$

Computing the flow in terms of the stream function and vorticity involves simultaneously solving equations (8.5.11) and (8.5.19) according to the following steps:

1. Guess the distribution of the stream function and associated vorticity distribution.
2. Solve the Poisson equation (8.5.11) for the stream function, ψ

$$\nabla^2 \psi \equiv \frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\omega_z, \tag{8.5.20}$$

   subject to the no-penetration boundary condition, $\psi = 0$.
3. Compute the right-hand side of (8.5.19).
4. Derive boundary conditions for the vorticity using the stream function obtained in Step 2.
5. Solve the Poisson equation (8.5.19) for the vorticity.
6. Check whether the vorticity computed in Step 5 agrees with that assigned in Step 1 within a specified tolerance. If not, we replace the latter by the former, return to Step 2, and repeat the calculations for another cycle.

Implementation

The method is implemented according to the following steps:

Step 1

Assign values to the stream function at all $(N_x + 1) \times (N_y + 1)$ interior and boundary grid nodes and to the vorticity at all $N_x \times N_y$ interior grid nodes. A simple choice is to set them all equal to zero.

Step 2

Solve the Poisson equation (8.5.11), subject to the boundary condition $\psi = 0$, using an iterative method. To perform the iterations, we approximate the second derivatives with centered differences and obtain equation (8.5.18), which we express in the form

$$\mathcal{R}_{i,j} \equiv \psi_{i+1,j} - 2\,(1+\beta)\psi_{i,j} + \psi_{i-1,j} + \beta\psi_{i,j+1} + \beta\psi_{i,j-1} + \Delta x^2\,\omega_{i,j} = 0, \qquad (8.5.21)$$

where $\mathcal{R}_{i,j}$ is a residual. The iterative method involves computing a time-like sequence of grid values parametrized by an index, ℓ, using the formula

$$\psi_{i,j}^{(\ell+1)} = \psi_{i,j}^{(\ell)} + \frac{\varrho}{2\,(1+\beta)}\,\mathcal{R}_{i,j}^{(\ell)} \qquad (8.5.22)$$

for $l = 1, 2, \ldots,$ where ϱ is a specified relaxation factor used to control the iterations.

Step 3

Compute the vorticity at the boundary grid nodes taking into consideration the velocity boundary conditions.

Considering grid nodes at the lid, we expand the stream function in a Taylor series with respect to y about a top grid node. Evaluating the expansion at the grid node immediately below, we obtain

$$\psi_{i,N_y} \simeq \psi_{i,N_y+1} - \Delta y\left(\frac{\partial\psi}{\partial y}\right)_{i,N_y+1} + \tfrac{1}{2}\,\Delta y^2\left(\frac{\partial^2\psi}{\partial y^2}\right)_{i,N_y+1}. \qquad (8.5.23)$$

Setting

$$\left(\frac{\partial\psi}{\partial y}\right)_{i,N_y+1} = V, \qquad \omega_{i,N_y+1} = -\left(\frac{\partial^2\psi}{\partial y^2}\right)_{i,N_y+1}, \qquad (8.5.24)$$

as discussed in the paragraph following equation (8.5.3), and solving for ω_{i,N_y+1}, we obtain

$$\omega_{i,N_y+1} = 2\,\frac{\psi_{i,N_y+1} - \psi_{i,N_y}}{\Delta y^2} - 2\,\frac{V}{\Delta y}. \qquad (8.5.25)$$

Working in a similar fashion, we derive corresponding expressions for the bottom, left, and right walls,

$$\omega_{i,1} = 2\,\frac{\psi_{i,1} - \psi_{i,2}}{\Delta y^2}, \qquad \omega_{1,j} = 2\,\frac{\psi_{1,j} - \psi_{2,j}}{\Delta y^2}, \qquad (8.5.26)$$

and

$$\omega_{N_x+1,j} = 2\,\frac{\psi_{N_x+1,j} - \psi_{N_x,j}}{\Delta y^2}. \tag{8.5.27}$$

More accurate expressions can be derived using higher-order expansions.

Step 4

Differentiate the stream function to generate the velocity at the interior grid nodes, subject to the no-penetration condition, $\psi = 0$.

Step 5

Differentiate the vorticity to obtain the x and y derivatives at the interior grid points, subject to boundary values computed in Step 3.

Step 6

Compute the right-hand side of (8.5.19) at the interior grid nodes.

Step 7

Solve equation (8.5.19) by iteration, as discussed in Step 2. The counterparts of equations (8.5.21) and (8.5.22) are

$$\mathcal{R}_{i,j} \equiv \omega_{i+1,j} - 2\,(1+\beta)\,\omega_{i,j} + \omega_{i-1,j} + \beta\,\omega_{i,j+1} + \beta\,\omega_{i,j-1} - \Delta x^2\,N_{i,j} = 0 \tag{8.5.28}$$

and

$$\omega_{i,j}^{(\ell+1)} = \omega_{i,j}^{(\ell)} + \frac{\varrho}{2\,(1+\beta)}\,\mathcal{R}_{i,j}^{(\ell)}, \tag{8.5.29}$$

where $N_{i,j}$ is the right-hand side of (8.5.19) evaluated at the (i,j) grid point, and ϱ is a relaxation factor.

The method is implemented in the following MATLAB code entitled *cvt_sv,* located in directory *11_fdm* of FDLIB:

```
%============
% code cvt_sv
%============

%-----
% parameters
%-----

Vlid = 1.0;     % lid velocity
Lx = 2.0; Ly = 1.0;     % cavity dimensions
```

```
Nx = 32; Ny = 16;     % grid size
visc = 0.01;     % viscosity
rho = 1.0;     % density
relax = 0.5;     % relaxation parameter
Niteri = 5;     % number of inner iterations
Niterg = 100;     % number of global iterations
vort_init = 0.0;     % initial vorticity

%-------
% prepare
%--------

Dx = Lx/Nx; Dy = Ly/Ny;
Dx2 = 2.0*Dx; Dy2 = 2.0*Dy;
Dxs = Dx*Dx;
Dys = Dy*Dy;
beta = Dxs/Dys; beta1 = 2.0*(1.0+beta);
nu = visc/rho;     % kinematic viscosity

%-----------------------------------------
% generate the grid
% initialize stream function and vorticity
%-----------------------------------------

for i=1:Nx+1
  for j=1:Ny+1
    x(i,j) = (i-1.0)*Dx;
    y(i,j) = (j-1.0)*Dy;
    psi(i,j) = 0.0;     % stream function
    vort(i,j) = -vort_init;
  end
end

%------------------
% global iterations
%------------------

for iter=1:Niterg

  save = vort;

%---------------------------------------
% Jacobi updating of the stream function
% at the interior nodes
%---------------------------------------

  for iteri=1:Niteri
    for j=2:Ny
      for i=2:Nx
```

```
        res = (psi(i+1,j)+psi(i-1,j)+ beta*psi(i,j+1) ...
        +beta*psi(i,j-1)+Dxs*vort(i,j))/beta1-psi(i,j);
        psi(i,j) = psi(i,j) + relax*res;
      end
    end
  end

%--------------------------------------
% Compute the vorticity at boundary grid points
% using the velocity boundary conditions
% (lower-order boundary conditions are commented out)
%--------------------------------------

%---
% top and bottom walls
%---

for i=2:Nx
% vort(i,1) = 2.0*(psi(i,1)-psi(i,2))/Dys;
% vort(i,Ny+1) = 2.0*(psi(i,Ny+1)-psi(i,Ny))/Dys-2.0*Vlid/Dys;
  vort(i,1) = (7.0*psi(i,1)-8.0*psi(i,2)+psi(i,3))/(2.0*Dys);
  vort(i,Ny+1) = (7.0*psi(i,Ny+1)-8.0*psi(i,Ny) ...
                +psi(i,Ny-1))/(2.0*Dys)-3.0*Vlid/Dy;
end

%---
% left and right walls
%---

for j=2:Ny
% vort(1,j) = 2.0*(psi(1,j)-psi(2,j) )/Dxs;
% vort(Nx+1,j) = 2.0*(psi(Nx+1,j)-psi(Nx,j))/Dxs;
  vort(1,j) = (7.0*psi(1,j)-8.0*psi(2,j)+psi(3,j))/(2.0*Dxs);
  vort(Nx+1,j) = (7.0*psi(Nx+1,j)-8.0*psi(Nx,j) ...
  +psi(Nx-1,j))/(2.0*Dxs);
end

%--------------------------------
% compute the velocity at the interior
% grid points by central differences
%--------------------------------

for j=2:Ny
  for i=2:Nx
    ux(i,j) = (psi(i,j+1)-psi(i,j-1))/Dy2;
    uy(i,j) = - (psi(i+1,j)-psi(i-1,j))/Dx2;
  end
end
```

```
%--------------------------------------------------
% iterate on Poisson's equation for the vorticity
%--------------------------------------------------

for iteri=1:Niteri
  for j=2:Ny
      for i=2:Nx
      source(i,j) = ux(i,j)*(vort(i+1,j)-vort(i-1,j))/Dx2...
          + uy(i,j)*(vort(i,j+1)-vort(i,j-1))/Dy2;
      source(i,j) = -source(i,j)/nu;
      res = (vort(i+1,j)+vort(i-1,j) + beta*vort(i,j+1) ...
          +beta*vort(i,j-1)+Dxs*source(i,j))/beta1-vort(i,j);
      vort(i,j) = vort(i,j) + relax*res;
    end
  end
end     % of iteri

%------------------
% monitor the error
%------------------

cormax = 0.0;

for i=1:Nx+1
  for j=1:Ny+1
    res = abs(vort(i,j)-save(i,j));
    if(res>cormax)
      cormax = res;
    end
  end
end

if(cormax<tol)
  break
end

end     % of iter
%--

%============
% graphics
%============

for i=1:Nx+1     % set up plotting vectors
  xgr(i) = Dx*(i-1);
end

for j=1:Ny+1
  ygr(j) = Dy*(j-1);
```

```
end

figure(1)
surf(20*xgr,20*ygr,vort')
xlabel('x','fontsize',15)
ylabel('y','fontsize',15)
zlabel(\verb1'\omega'1,'fontsize',15)
set(gca,'fontsize',15)
axis([0 Lx 0 Ly -10 10])
axis equal

figure(2)
contour(xgr,ygr,psi',32)
xlabel('x','fontsize',15)
ylabel('y','fontsize',15)
zlabel('\psi','fontsize',15)
axis([0 Lx 0 Ly])
axis equal
```

The graphics module at the end of the code invokes the internal MATLAB functions *surf* and *contour.*

Vorticity and stream function contour plots for a cavity with aspect ratio $L_x/L_y = 2$ at Reynolds number $\mathrm{Re} = VL_x/\nu = 1$ and 100 are shown in Figure 8.5.2. Stream function contours are streamlines and particle paths in a two-dimensional flow. As the Reynolds number increases, the center of the eddy developing inside the cavity is shifted toward the right wall due to the fluid inertia. Regions of recirculating flow develop at the bottom two corners, requiring increased spatial resolution.

A local analysis in the context of Stokes flow shows that the shear stress diverges at the upper two cavity corners and an infinite force is required to slide the lid as a result of the sharp-corner idealization. It is remarkable that the singular behavior of the vorticity at these corners due to the discontinuous boundary velocity does not deter the overall performance of the numerical method.

8.5.5 Summary

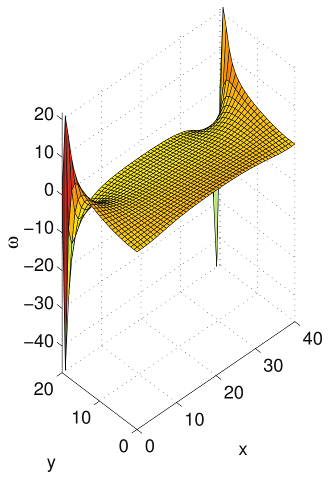
In summary, the stream function/vorticity formulation is distinguished by the following features:

1. Expressing the velocity in terms of the stream function ensures the automatic satisfaction of the continuity equation.

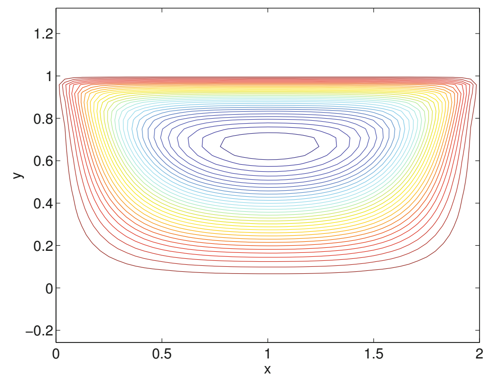
2. The formulation bypasses the computation of the pressure. If we had to solve for the pressure, we would have to derive appropriate boundary conditions, as discussed in Section 8.6.

3. Enforcing the no-penetration and the no-slip boundary condition is done sequentially rather than simultaneously. The no-penetration condition is enforced when solving for

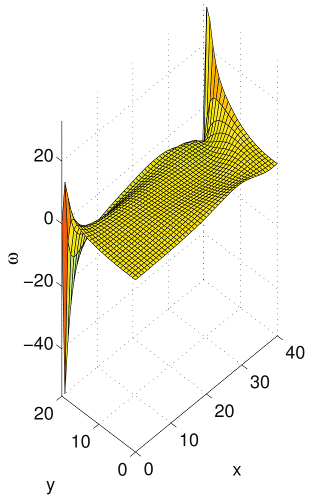
(*a*)



(*b*)



(*c*)



(*d*)

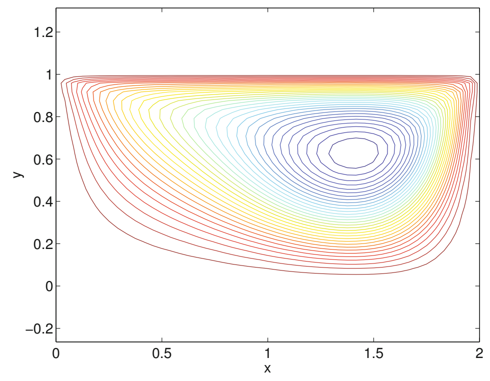


Figure 8.5.2 Vorticity and stream function contour plots for flow in a rectangular cavity with aspect ratio $L_x/L_y = 2$, at Reynolds number $(a,\ b)$ $\mathrm{Re} = V L_x/\nu = 1$ and $(c,\ d)$ 100.

the stream function, while the no-slip condition is enforced when deriving boundary conditions for the vorticity.

Similar simplifications occur when solving for the Stokes stream function describing axisymmetric flow in the absence of swirling motion.

Problems

8.5.1 *Computation of the pressure*

Show that the pressure distribution in an incompressible fluid satisfies Poisson's equation

$$\frac{1}{2\rho}\,\nabla^2 p = \frac{\partial^2\psi}{\partial x^2}\,\frac{\partial^2\psi}{\partial y^2} - \Big(\frac{\partial^2\psi}{\partial x\partial y}\Big)^2. \tag{8.5.30}$$

Hint: Take the divergence of the Navier–Stokes equation and use the continuity equation.

8.5.2 *Axisymmetric flow*

Develop a finite-difference method based on the stream function/vorticity formulation for steady axisymmetric flow in an annular cavity depressed on a circular cylinder. The flow is driven by a sleeve sliding along the cylinder surface.

8.5.3 *Steady flow in a cavity*

(*a*) Run the code *cvt_sv* and prepare velocity vector plots for the two flows illustrated in Figure 8.5.2. Discuss the structure of the streamline pattern.

(*b*) Investigate the performance of the numerical method for a square cavity at high Reynolds numbers.

(*c*) Duplicate the results shown in Figure 8.5.2 for a slender cavity with aspect ratio $L_x/L_y = 8$. Discuss the structure of the flow.

(*d*) Implement a stopping check so that the computations terminate when the vorticity field has been computed within a specified accuracy.

(*e*) Resolve and discuss the structure of the eddies developing near the lower two corners.

8.6 Velocity/pressure formulation

Although the stream function/vorticity formulation discussed in Section 8.5 is simple and efficient, its extension to three dimensions and its generalization to flow in the presence of interfaces are cumbersome. To handle arbitrary flow configurations, we develop a direct formulation in primary variables, including the velocity and the pressure.

Evolution equations

To compute the evolution of an unsteady flow, we require an evolution equation for the velocity and another evolution equation for the pressure. The former is provided by the Navier–Stokes equation stated as

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{N}(\mathbf{u}) - \frac{1}{\rho}\,\boldsymbol{\nabla} p + \nu\,\mathbf{L}(\mathbf{u}), \tag{8.6.1}$$

where $\mathbf{N}(\mathbf{u})$ is a nonlinear convection operator and $\mathbf{L}(\mathbf{u})$ is a linear diffusion operator defined as

$$\mathbf{N}(\mathbf{u}) \equiv -\mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u}, \qquad \mathbf{L}(\mathbf{u}) \equiv \nabla^2\mathbf{u}. \tag{8.6.2}$$

If the fluid were compressible, the continuity equation would provide us with an evolution equation for the density, as shown in equations (2.7.13) and (2.7.14). An evolution equation for the pressure could then be obtained by introducing an equation of state relating the density to the local pressure and temperature.

An explicit evolution equation for the pressure developing in a incompressible fluid is not available. Instead, the continuity equation takes the form a kinematic constraint,

$$\boldsymbol{\nabla} \cdot \mathbf{u} = 0, \tag{8.6.3}$$

which requires that the pressure field evolves so that the rate of expansion, $\boldsymbol{\nabla} \cdot \mathbf{u}$, is zero throughout the domain of flow at any time.

To convert this requirement into a mathematical restriction, we take the divergence of the Navier–Stokes equation (8.6.1), interchange the divergence with the time derivative on the left-hand side, and thus derive an evolution equation for the rate of expansion,

$$\frac{\partial\, \boldsymbol{\nabla} \cdot \mathbf{u}}{\partial t} = \boldsymbol{\nabla} \cdot \mathbf{N}(\mathbf{u}) - \frac{1}{\rho}\, \nabla^2 p + \nu\, \boldsymbol{\nabla} \cdot \mathbf{L}(\mathbf{u}). \tag{8.6.4}$$

Note that the divergence operator and the nonlinear operator $\mathbf{N}$ on the right-hand side do not commute, that is,

$$\boldsymbol{\nabla} \cdot \mathbf{N}(\mathbf{u}) \neq \mathbf{N}(\boldsymbol{\nabla} \cdot \mathbf{u}). \tag{8.6.5}$$

For simplicity, we have assumed that the density and viscosity are uniform throughout the domain of flow.

Pressure Poisson equation

Equation (8.6.3) requires that the left-hand side of (8.6.4) vanishes at any time, which will be true if the pressure satisfies the pressure Poisson equation (PPE),

$$\nabla^2 p = \rho\, \boldsymbol{\nabla} \cdot \mathbf{N}(\mathbf{u}) + \mu\, \boldsymbol{\nabla} \cdot \mathbf{L}(\mathbf{u}). \tag{8.6.6}$$

It could be argued that, since the divergence operator and the linear operator $\mathbf{L}$ commute,

$$\boldsymbol{\nabla} \cdot \mathbf{L}(\mathbf{u}) = \mathbf{L}(\boldsymbol{\nabla} \cdot \mathbf{u}), \tag{8.6.7}$$

the last term on the right-hand side of (8.6.6) could be set to zero, yielding the simplified pressure Poisson equation (SPPE),

$$\nabla^2 p = \rho\, \boldsymbol{\nabla} \cdot \mathbf{N}(\mathbf{u}). \tag{8.6.8}$$

However, in practice, the magnitude of the last term on the right-hand side of (8.6.6) is nonzero due to numerical error associated with the approximation of partial derivatives with finite differences. It turns out that the complete absence of this term may be detrimental to the performance of the numerical method by fostering the growth of small oscillations. To prevent the onset of these oscillations, the PPE is preferred over its simplified counterpart.

8.6.1 Alternative system of governing equations

The preceding discussion suggests a numerical procedure for computing the evolution of an unsteady flow based on equations (8.6.1) and (8.6.6) or (8.6.8): compute the evolution of the velocity using (8.6.1), and simultaneously obtain the evolution of the pressure by solving the Poisson equation (8.6.6) or (8.6.8).

The method is analogous to that employed in the stream function/vorticity formulation discussed in Section 8.5. One important difference is that, by employing the stream function, the satisfaction of the continuity equation (8.6.3) is guaranteed, independent of the magnitude of the numerical error.

To examine whether the velocity/pressure formulation ensures the satisfaction of the continuity equation (8.6.3), we substitute (8.6.6) into the right-hand side of the pressure Poison equation (8.6.4) and obtain the expected result

$$\frac{\partial\,\boldsymbol{\nabla}\cdot\mathbf{u}}{\partial t}=0, \tag{8.6.9}$$

which ensures that, if the rate of expansion vanishes at the initial instant by a sensible choice of the initial condition, it will also vanish at any time.

Substituting (8.6.8) into the right-hand side of the simplified pressure Poison equation (8.6.4), we obtain an unsteady diffusion equation for the rate of expansion,

$$\frac{\partial\,\boldsymbol{\nabla}\cdot\mathbf{u}}{\partial t}=\nu\,\boldsymbol{\nabla}\cdot\mathbf{L}(\mathbf{u}), \tag{8.6.10}$$

which ensures that, if the rate of expansion vanishes at the initial instant by an appropriate choice of an initial condition, it will also vanish at any time provided that the boundary values of the rate of expansion also vanish at any time. The additional condition underlines the importance of accurately satisfying mass conservation at the boundaries and explains why (8.6.6) is preferred over its simplified counterpart (8.6.8).

8.6.2 Pressure boundary conditions

To solve the pressure Poisson equation, we require a pressure boundary condition derived from specified boundary conditions for the velocity. The pressure boundary condition emerges by evaluating the Navier–Stokes equation (8.6.1) at the boundaries of the flow, and then taking the inner product of both sides with the unit vector normal to the boundaries pointing outward, **n**. The result is the Neumann boundary condition

$$\mathbf{n}\cdot\boldsymbol{\nabla}p=\rho\,\mathbf{n}\cdot\big(-\frac{\partial\mathbf{u}}{\partial t}+\mathbf{N}(\mathbf{u})+\nu\,\mathbf{L}(\mathbf{u})\,\big). \tag{8.6.11}$$

The left-hand side is the derivative of the pressure normal to the boundaries, expressing the rate of change of the pressure with respect to distance normal to the boundaries. The right-hand side is then simplified by implementing the no-slip and no-penetration boundary conditions.

For example, in the case of two-dimensional flow over a horizontal stationary wall located at $y = 0$, we require that $u_x = 0$ and $u_y = 0$ at $y = 0$, and obtain

$$\mathbf{n} \cdot \boldsymbol{\nabla} p = \frac{\partial p}{\partial y} = \mu \, \frac{\partial^2 u_y}{\partial y^2} \tag{8.6.12}$$

in a steady or unsteady flow. The left-hand side is the normal derivative of the pressure, while the right-hand side is the negative of the normal derivative of the vorticity multiplied by the fluid viscosity.

8.6.3 Compatibility condition for the pressure

The Poisson equation governing the pressure distribution in an incompressible fluid is analogous to the Poisson equation governing the steady-state temperature distribution in a conductive medium identified with the domain of flow, subject to a homogeneous heat production term expressed by the right-hand side.

The boundary condition (8.6.11) specifies the boundary distribution of the flux in terms of the instantaneous velocity. Physical reasoning suggests that a steady distribution will exist only if the total rate of heat production is balanced by the total rate of heat removal across the boundaries, so that heat neither accumulates to elevate the temperature nor is depleted to lower the temperature.

In the case of two-dimensional flow, the mathematical expression of this requirement takes the form of a compatibility condition, stating that the areal integral of the right-hand side of (8.6.6) or (8.6.8) over the domain of flow should be equal to the line integral of the right-hand side of (8.6.11) or (8.6.12) over the boundaries. If the compatibility condition is not fulfilled, a solution for the pressure cannot be found.

In the case of three-dimensional flow, the compatibility condition requires that the volume integral of the right-hand side of (8.6.6) or (8.6.8) over the domain of flow should be equal to the surface integral of the right-hand side of (8.6.11) or (8.6.12) over the boundaries. If the compatibility condition is not fulfilled, a solution for the pressure cannot be found.

In numerical practice, this compatibility condition is enforced implicitly or explicitly depending on the particulars of the implementation of the numerical method. In some over-simplified approaches, the compatibility condition is altogether ignored and an approximate solution is found.

Problem

8.6.1 *Pressure boundary condition*

Derive the pressure boundary condition (8.6.12).

8.7 Operator splitting and solenoidal projection

In practice, the velocity/pressure formulation is implemented in a way that expedites the numerical solution and reduces the computational cost. For the purpose of illustration, we discuss the computation of an evolving two-dimensional flow. Extending the methodology to three-dimensional flow is straightforward in principle and implementation.

Operator splitting

In the most popular implementation of the velocity/pressure formulation, the Navier–Stokes equation (8.6.1) is resolved into two constituent equations,

$$\frac{\partial \mathbf{u}}{\partial t} = \mathbf{N}(\mathbf{u}) + \nu\,\mathbf{L}(\mathbf{u}) \tag{8.7.1}$$

and

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho}\,\boldsymbol{\nabla} p, \tag{8.7.2}$$

where the operators $\mathbf{N}(\mathbf{u})$ and $\mathbf{L}(\mathbf{u})$ are defined in equations (8.6.2),

$$\mathbf{N}(\mathbf{u}) \equiv -\mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u}, \qquad \mathbf{L}(\mathbf{u}) \equiv \nabla^2\mathbf{u}. \tag{8.7.3}$$

The right-hand sides of equations (8.7.1) and (8.7.2) arise by splitting the full Navier–Stokes operator on the right-hand side of (8.6.1) into two parts, subject to the following interpretation.

Consider the change in the velocity field over a small time interval, Δt, following the current time, t. The decomposition into (8.7.1) and (8.7.2) is inspired by the idea of updating the velocity in two sequential stages, where the first update is due to inertia and viscosity, while the second update is due to the pressure gradient alone. Time is to $t + \Delta t$ after the completion of the second stage. We will see that this decomposition significantly simplifies the implementation of the numerical method by allowing the convection–diffusion and pressure gradient steps to be handled independently using appropriate numerical methods.

Two main issues arise. First, the boundary condition for the velocity to be used for integrating (8.7.1) cannot be the same as the specified physical boundary condition, otherwise the second step mediated by (8.7.2) will cause a departure. Second, the boundary condition for the pressure may no longer be computed from (8.6.11), but should be derived instead using equation (8.7.2).

Projection function

The second observation suggests that p in equation (8.7.2) may no longer be regarded as the hydrodynamic pressure, p, and should be interpreted instead as a fictitious pressure whose role is to ensure that the velocity field is solenoidal at the end of the second step. To make this distinction clear, we replace equation (8.7.2) with the equation

$$\frac{\partial \mathbf{u}}{\partial t} = -\frac{1}{\rho}\,\boldsymbol{\nabla}\chi, \tag{8.7.4}$$

where χ is a projection function. Equation (8.7.4) receives the velocity field delivered by the convection–diffusion equation (8.7.1), which is not necessarily solenoidal, and removes the non-solenoidal component in a process that can be described as projection into the space of solenoidal functions.

The choice of boundary conditions for the projection function, χ, has been the subject of extensive discussion. It can be shown that the homogeneous Neumann boundary condition, requiring that the derivative of the projection function χ with respect to distance normal to a boundary vanishes, is appropriate. The associated boundary conditions for the velocity will be discussed in Section 8.7.3.

Next, we discuss the implementation of numerical methods for performing the convection–diffusion and projection steps expressed by equations (8.7.1) and (8.7.4).

8.7.1 Convection–diffusion step

To prevent numerical instability, we perform the convection–diffusion step expressed by equation (8.7.1) by an implicit finite-difference method. This means that updating the velocity requires solving linear systems of algebraic equations for the velocity at all nodes.

Evaluating the x and y components of equation (8.7.1) at the (i, j) grid point at time $t + \Delta t$, and approximating the time derivatives with backward differences and the spatial derivatives with differences of our choice, we derive a system of equations for the unknown velocity vector comprised of the x and y velocity components at the grid points at time $t+\Delta t$. The size of the velocity vector is twice the number of grid points. For a 32×32 grid, we obtain a velocity vector with nearly $2,000$ unknowns and an equal number of equations whose solution requires a significant computational cost.

Directional splitting

As an alternative, we split the operator on the right-hand side of (8.7.1) into two spatial constituents expressing convection–diffusion in the x or y direction, given by

$$\frac{\partial \mathbf{u}}{\partial t} = -u_x \frac{\partial \mathbf{u}}{\partial x} + \nu \frac{\partial^2 \mathbf{u}}{\partial x^2} \tag{8.7.5}$$

and

$$\frac{\partial \mathbf{u}}{\partial t} = -u_y \frac{\partial \mathbf{u}}{\partial y} + \nu \frac{\partial^2 \mathbf{u}}{\partial y^2}, \tag{8.7.6}$$

and advance the velocity over the time interval Δt in a sequential fashion based on this decomposition.

Crank–Nicolson integration

To achieve second-order accuracy, we discretize equation (8.7.5) using the Crank–Nicolson method. The implementation involves evaluating (8.7.5) at the (i, j) grid point at time

$t+\frac{1}{2}\,\Delta t$, approximating the time and space derivatives with central differences, and averaging the space derivatives over the time levels t and $t+\Delta t$,

$$\frac{\mathbf{u}^*_{i,j}-\mathbf{u}_{i,j}(t)}{\Delta t} = -\frac{1}{2}\,(u_x)_{i,j}(t)\,\Big[\,\big(\frac{\mathbf{u}_{i+1,j}-\mathbf{u}_{i-1,j}}{2\Delta x}\big)(t) + \frac{\mathbf{u}^*_{i+1,j}-\mathbf{u}^*_{i-1,j}}{2\Delta x}\,\Big]$$
$$+\frac{1}{2}\,\nu\,\Big[\,\big(\frac{\mathbf{u}_{i+1,j}-2\,\mathbf{u}_{i,j}+\mathbf{u}_{i-1,j}}{\Delta x^2}\big)(t) + \frac{\mathbf{u}^*_{i+1,j}-2\,\mathbf{u}^*_{i,j}+\mathbf{u}^*_{i-1,j}}{\Delta x^2}\,\Big]. \qquad (8.7.7)$$

An asterisk designates the first intermediate velocity field.

To simplify the notation, we define

$$\mathbf{u}^n_{i,j} \equiv \mathbf{u}_{i,j}(t), \qquad (8.7.8)$$

where the superscript n denotes the nth time level corresponding to time t. Rearranging equation (8.7.7), we derive the finite-difference equation

$$-(c_x+2\alpha_x)\,\mathbf{u}^*_{i-1,j} + 4(1+\alpha_x)\,\mathbf{u}^*_{i,j} + (c_x-2\alpha_x)\,\mathbf{u}^*_{i+1,j}$$
$$= (c_x+2\alpha_x)\,\mathbf{u}^n_{i-1,j} + 4(1-\alpha_x)\,\mathbf{u}^n_{i,j} - (c_x-2\alpha_x)\,\mathbf{u}^n_{i+1,j}, \qquad (8.7.9)$$

involving the local x convection number,

$$c_x \equiv \frac{(u_x)^n_{i,j}\,\Delta t}{\Delta x}, \qquad (8.7.10)$$

and the x diffusion number,

$$\alpha_x \equiv \frac{\nu\,\Delta t}{\Delta x^2}. \qquad (8.7.11)$$

The right-hand side of (8.7.9) can be computed in terms of the velocity at the grid points at the nth time level, which is available.

Evaluating (8.7.9) at grid points that lie along y grid lines corresponding to fixed values of j, we obtain tridiagonal systems of equations for the x and y components of the first intermediate velocity. The salient advantage of the method of directional splitting is that these tridiagonal systems can be solved efficiently using the Thomas algorithm discussed in Section 8.2.4, subject to boundary conditions discussed in Section 8.7.3.

An analogous discretization of (8.7.6) yields

$$-(c_y+2\alpha_y)\,\mathbf{u}^{**}_{i,j-1} + 4(1+\alpha_y)\,\mathbf{u}^{**}_{i,j} + (c_y-2\alpha_y)\,\mathbf{u}^{**}_{i,j+1}$$
$$= (c_y+2\alpha_y)\,\mathbf{u}^*_{i,j-1} + 4(1-\alpha_y)\,\mathbf{u}^*_{i,j} - (c_y-2\alpha_y)\,\mathbf{u}^*_{i,j+1}, \qquad (8.7.12)$$

where

$$c_y \equiv \frac{(u_y)^n_{i,j}\,\Delta t}{\Delta y} \qquad (8.7.13)$$

is the local y convection number and

$$\alpha_y \equiv \frac{\nu\,\Delta t}{\Delta y^2} \tag{8.7.14}$$

is the y diffusion number. A double asterisk in (8.7.12) designates the second intermediate velocity field.

The right-hand side of (8.7.12) can be computed in terms of the first intermediate velocity delivered by equation (8.7.9). Evaluating (8.7.12) at grid points that lie along x grid lines corresponding to fixed values of i, we obtain tridiagonal systems of equations for the x and y components of the second intermediate velocity. The solution can be found using the Thomas algorithm discussed in Section 8.2.4, subject to boundary conditions discussed in Section 8.7.3.

8.7.2 Projection step

Next, we advance the velocity field using the projection step (8.7.4), where the projection function is computed to satisfy the continuity equation at the end of this step. Evaluating (8.7.5) at the (i,j) grid point and approximating the time derivative with a finite difference, we obtain

$$\frac{\mathbf{u}_{i,j}(t+\Delta t) - \mathbf{u}^{**}_{i,j}}{\Delta t} = -\frac{1}{\rho}\,(\boldsymbol{\nabla}\chi)^n_{i,j}, \tag{8.7.15}$$

which can be rearranged to give

$$\mathbf{u}_{i,j}(t+\Delta t) = \mathbf{u}^{**}_{i,j} - \frac{\Delta t}{\rho}\,(\boldsymbol{\nabla}\chi)^n_{i,j}. \tag{8.7.16}$$

The gradient on the right-hand side of (8.7.16) can be approximated by centered, forward, or backward differences.

Now we consider the numerical discretization of the continuity equation, $\boldsymbol{\nabla}\cdot\mathbf{u} = 0$. Using centered differences, we approximate the rate of expansion at the (i,j) grid point with the discrete form

$$D_{i,j} \equiv (\boldsymbol{\nabla}\cdot\mathbf{u})_{i,j} \simeq \frac{(u_x)_{i+1,j} - (u_x)_{i-1,j}}{2\Delta x} + \frac{(u_y)_{i,j+1} - (u_y)_{i,j-1}}{2\Delta y}. \tag{8.7.17}$$

Evaluating (8.7.17) at the $n+1$ time level corresponding to time $t+\Delta t$, requiring that the left-hand side is zero, and using (8.7.16) to express $\mathbf{u}(t+\Delta t)$ on the right-hand side in terms of the second intermediate velocity denoted by the double asterisk and the projection function, we obtain the expression

$$\frac{\rho}{\Delta t}\,(\boldsymbol{\nabla}\cdot\mathbf{u}^{**})_{i,j} = \frac{(\frac{\partial\chi}{\partial x})_{i+1,j} - (\frac{\partial\chi}{\partial x})_{i-1,j}}{2\Delta x} + \frac{(\frac{\partial\chi}{\partial y})_{i,j+1} - (\frac{\partial\chi}{\partial y})_{i,j-1}}{2\Delta y}. \tag{8.7.18}$$

The right-hand side of (8.7.18) is recognized as the discrete divergence of the gradient of the projection function χ.

For grid points that are not adjacent to a wall, we approximate the partial derivatives of the right-hand side of (8.7.18) with centered differences and simplify to obtain

$$\frac{\rho}{\Delta t}\,(\boldsymbol{\nabla}\cdot\mathbf{u}^{**})_{i,j} = \frac{\chi_{i-2,j} - 2\,\chi_{i,j} + \chi_{i+2,j}}{4\,\Delta x^2} + \frac{\chi_{i,j-2} - 2\,\chi_{i,j} + \chi_{i,j+2}}{4\,\Delta y^2}. \qquad (8.7.19)$$

The right-hand side of (8.7.19) is recognized as the finite-difference approximation of the Laplacian of χ, computed with spatial intervals equal to $2\,\Delta x$ and $2\,\Delta y$.

For points that are adjacent to a wall, we derive corresponding formulas incorporating the homogeneous Neumann boundary condition. For example, applying equation (8.7.18) at the near-corner point $i = 2$ and $j = 2$, and setting $(\partial\chi/\partial y)_{2,1} = 0$ and $(\partial\chi/\partial x)_{1,2} = 0$, we obtain

$$\frac{\rho}{\Delta t}\,(\boldsymbol{\nabla}\cdot\mathbf{u}^{**})_{2,2} = \frac{\chi_{4,2} - \chi_{2,2}}{4\,\Delta x^2} + \frac{\chi_{2,4} - \chi_{2,2}}{4\,\Delta y^2}. \qquad (8.7.20)$$

Returning to (8.7.19), we reduce the intervals of the centered spatial differences to Δx and Δy, and derive the alternative expression

$$\frac{\rho}{\Delta t}\,(\boldsymbol{\nabla}\cdot\mathbf{u}^{**})_{i,j} = \frac{\chi_{i+1,j} - 2\chi_{i,j} + \chi_{i-1,j}}{\Delta x^2} + \frac{\chi_{i,j+1} - 2\chi_{i,j} + \chi_{i,j-1}}{\Delta y^2}, \qquad (8.7.21)$$

which is applicable at all interior grid points. This finite-difference equation could have been derived directly from (8.7.15) by taking the divergence of both sides and then approximating the emerging Laplacian of χ on the right-hand side with the five-point formula, as shown in (8.7.21).

Evaluating (8.7.19) or (8.7.21) at the interior grid points and their counterparts for the wall-adjacent points, and introducing boundary conditions for χ, we derive a system of linear equations for the grid values of χ, which is the counterpart of the linear system descending from the pressure Poisson equation discussed in Section 8.6. Having computed the grid values of the projection function, we return to equation (8.7.16) and perform the final step, advancing the velocity to the $n+1$ time level corresponding to time $t + \Delta t$.

Because the coefficient matrix of the linear system associated with (8.7.19) or (8.7.21) is independent of time, we may either compute the matrix inverse at the outset and then solve the system at each step by simple matrix-vector multiplication, or employ efficient custom-made iterative solution algorithms.

8.7.3 Boundary conditions for the intermediate velocity

Next, we address the issue of boundary conditions for the intermediate velocities denoted by a single or double asterisk. The choice of these boundary conditions is pivoted on a key observation: because of the homogeneous Neumann condition chosen for the projection function, the projection step introduces a tangential but not a normal component of the boundary velocity. Accordingly, the boundary conditions for the intermediate velocity should be such that the tangential velocity introduced in the projection step brings the total velocity to the specified physical value at the end of a complete step. In practice, this is done by estimating the magnitude of the intermediate slip velocity and then improving the guess by iteration, as explained in Section 8.7.4.

8.7.4 Flow in a cavity

The implementation of the numerical method involves further considerations that are best illustrated with reference to the familiar problem of two-dimensional flow in a cavity driven by a sliding lid.

Homogeneous Neumann boundary condition for the projection function

Consider the numerical implementation of the condition of zero normal derivative of the projection function at the boundaries of the cavity illustrated in Figure 8.5.1. Requiring that $\partial\chi/\partial y = 0$ at the bottom and top walls, and approximating the first derivative with a second-order forward or backward finite difference, we obtain

$$\left(\frac{\partial\chi}{\partial y}\right)_{i,1} \simeq \frac{-3\,\chi_{i,1} + 4\,\chi_{i,2} - \chi_{i,3}}{2\Delta y} = 0 \tag{8.7.22}$$

and

$$\left(\frac{\partial\chi}{\partial y}\right)_{i,N_y+1} \simeq \frac{\chi_{i,N_y-1} - 4\,\chi_{i,N_y} + 3\,\chi_{i,N_y+1}}{2\Delta y} = 0. \tag{8.7.23}$$

Requiring that $\partial\chi/\partial x = 0$ at the left and right walls, and approximating the first derivative with a second-order forward or backward finite-difference, we obtain

$$\left(\frac{\partial\chi}{\partial x}\right)_{1,j} \simeq \frac{-3\,\chi_{1,j} + 4\,\chi_{2,j} - \chi_{3,j}}{2\Delta x} = 0 \tag{8.7.24}$$

and

$$\left(\frac{\partial\chi}{\partial x}\right)_{N_x+1,j} \simeq \frac{\chi_{N_x-1,j} - 4\,\chi_{N_x,j} + 3\,\chi_{N_x+1,j}}{2\Delta x} = 0. \tag{8.7.25}$$

These difference equations complement those arising from the discretization of the Poisson equation.

Compatibility condition for system (8.7.19)

The linear system descending from the discrete Poisson equation (8.7.19) accompanied by the homogeneous Neumann boundary conditions is singular, which means that it has either no solution or an infinite number of solutions, depending on the right-hand side. If multiple solutions exist, any particular solution can be offset by an arbitrary constant vector with equal elements. Correspondingly, the value of the projection function at the grid points can be offset by a physically irrelevant constant. Reference to (8.7.16) ensures that this constant has no effect on the structure of the flow.

When the discrete divergence of the second intermediate velocity is computed using (8.7.17), the discrete form of the compatibility condition discussed at the end of Section 8.6 is fulfilled and the linear system has a multiplicity of solutions. A solution can be found by assigning an arbitrary value to one of the unknowns, discarding one equation, and solving the rest of the equations for the remaining unknowns. Unfortunately, the numerical solution computed in this fashion can be contaminated by artificial oscillations described as *odd-even coupling*.

Compatibility condition for system (8.7.21)

The linear system descending from the discrete Poisson equation (8.7.21) accompanied by the homogeneous Neumann boundary conditions is also singular, reflecting the arbitrary level of the projection function. Unfortunately, when the discrete divergence of the second intermediate velocity field is computed using (8.7.17), the discrete form of the compatibility condition is not satisfied. Consequently, one equation of the linear system cannot be satisfied to machine precision.

Resisting the temptation to fudge the computation by discarding one arbitrary equation, we add a small term to the right-hand side of (8.7.21) and then adjust the magnitude of this term to satisfy the compatibility condition of a modified system of equations. If

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{8.7.26}$$

is the linear system corresponding to (8.7.21), then the modified system is

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} + \epsilon\, \mathbf{c}, \tag{8.7.27}$$

where ϵ is an *a priori* unknown constant and $\mathbf{c}$ is a constant vector that emerges by replacing the left-hand side of (8.7.18) with an arbitrary value, while retaining the linear equations implementing the homogeneous Neumann boundary conditions.

Our objective is to adjust the value of the constant ϵ so that the system (8.7.27) has an infinite number of solutions. In one approach, we work as follows:

1. First, we set the last component of $\mathbf{x}$ to zero, discard the last equation of $\mathbf{A}\cdot\mathbf{x} = \mathbf{b}$, solve the remaining equations, and call the solution $\mathbf{x}^{(1)}$. Then we evaluate the difference between the left-hand side and the right-hand of the last equation, $r^{(1)}$.

2. Second, we set the last component of $\mathbf{x}$ to zero, discard the last equation of $\mathbf{A} \cdot \mathbf{x} = \mathbf{c}$, solve the remaining equations, and call the solution $\mathbf{x}^{\text{ref}}$. Then evaluate the difference between the left-hand side and the right-hand of the last equation, denoted by r^{ref}.

3. The desired solution is

$$\mathbf{x} = \mathbf{x}^{(1)} + \epsilon\, \mathbf{x}^{\text{ref}}, \tag{8.7.28}$$

   where

$$\epsilon = -\frac{r^{(1)}}{r^{\text{ref}}}. \tag{8.7.29}$$

A more rigorous approach involves removing from the right-hand side, **b**, its projection on the eigenvector corresponding to the null eigenvalue of the transpose of the coefficient matrix, $\mathbf{A}^{\mathrm{T}}$.

Boundary conditions for the intermediate velocity

The boundary conditions for the intermediate velocity must be such that the right-hand side of (8.7.16) is consistent with the specified physical boundary conditions at time $t + \Delta t$. Requiring that the left-hand side of (8.7.16) is zero over a stationary boundary, we obtain the boundary condition

$$\mathbf{u}^{**}_{i,j} = \frac{\Delta t}{\rho}\,(\boldsymbol{\nabla}\chi)^n_{i,j}. \tag{8.7.30}$$

Because the projection function was required to satisfy the homogeneous Neumann boundary condition, the right-hand side of (8.7.30) has only a tangential component expressing *numerical wall slip.*

An apparent difficulty in computing the tangential component of the intermediate velocity is that the right-hand side of (8.7.30) is not available during the convection–diffusion step. To circumvent this difficulty, we may approximate the projection function with that at the previous step, proceed with the projection step, and then improve the approximation by repeating the convection–diffusion step until the slip velocity has fallen below a sufficiently small threshold.

Code cvp_pm

The following MATLAB code entitled *cvt_pm,* located inside directory *11_fdm* of FDLIB, performs the time integration using the projection method discussed in this section and animates the evolving velocity vector field. The code should be read in two columns on each page:

```
close all
clear all

%====================================
% Computation of evolving flow
% in a rectangular cavity
% in primary variables using the
% velocity/pressure formulation
%
% The flow is computed using a
% projection method
%
% SYMBOLS:
%
% x,y:      grid nodes
% ux, uy:   velocity components
```

```
% chi:      projection function
% uxi, uyi: intermediate velocity
%           (generic)
%====================================

%-----------------------
% settings and parameters
%-----------------------

Lx = 1.0; % cavity length
Ly = 0.5; % cavity depth

Nx = 4*8; % grid size
Ny = 4*4;

Dt = 0.1; % time step
```

```
visc = 0.01; % viscosity
dens = 1.0; % density

Vlidamp = 1.0; % lid velocity amplitude

Nstep = 2000;  % number of steps

%-----------------------------------
% parameters for solving the Poisson
% equation for the projection function
%-----------------------------------

itermax = 50000;
tol = 0.000001;
relax = 0.2;

qleft = 0.0;  % Neumann bound cond
qright = 0.0;
qbot = 0.0;
qtop = 0.0;

Ishift = 1;

%----------------------------------
% parameters for slip vel iterations
%----------------------------------

slipN = 50;        % max iteration no
sliptol = 0.00001; % tolerance
sliprel = 1.0;     % relaxation

%--------
% prepare
%--------


nu = visc/dens; % kinematic viscosity

Dx = Lx/Nx;
Dy = Ly/Ny;

Dx2 = 2.0*Dx;
Dy2 = 2.0*Dy;

Dxs = Dx*Dx;
Dys = Dy*Dy;

Dtor = Dt/dens;
```

```
%--------
% lid velocity
%--------

for i=1:Nx+1
  Vlid(i) = Vlidamp;
end

%----------------------------------
% define grid lines
% initialize velocity (ux, uy)
% and the projection function (chi)
%----------------------------------

time = 0.0;

for j=1:Ny+1
 for i=1:Nx+1
   x(i,j) = (i-1.0)*Dx;
   y(i,j) = (j-1.0)*Dy;
   ux(i,j) = 0.0;  % x velocity
   uy(i,j) = 0.0;  % y velocity
   chi(i,j) = 0.0; % projection function
   chroma(i,j) = 0.0; % for plotting
 end
end

for i=1:Nx+1
   ux(i,Ny+1) = Vlid(i);
end

%----------------------------------
% naive velocity boundary conditions
%----------------------------------

for i=1:Nx+1
  BCxt(i) = Vlid(i);  % top wall
  BCxb(i) = 0.0;      % bottom wall
end

for j=1:Ny+1
  BCyl(j) = 0.0; % left wall
  BCyr(j) = 0.0; % right wall
end

%-------------
% time stepping
%-------------
```

```
%===============
for step=1:Nstep
%===============

%---
% animation
%---

if(step==1)
  Handle1 = quiver(x,y,ux,uy,'k');
  set(Handle1, 'erasemode', 'xor');
  axis ([-0.10,Lx+0.10,-0.10,Ly+0.10])
  axis equal
  set(gca,'fontsize',15)
  xlabel('x','fontsize',15)
  ylabel('y','fontsize',15)
  hold on
  plot([0, Lx, Lx, 0, 0] ...
      ,[0, 0, Ly, Ly, 0],'k');
else
  set(Handle1,'UData',ux,'VData',uy);
  pause(0.01)
  drawnow
end

%--------------------------------------
% initialize the intermediate velocity
%--------------------------------------

for j=1:Ny+1
  for i=1:Nx+1
   uxi(i,j) = ux(i,j);
   uyi(i,j) = uy(i,j);
  end
end

%==============================
% perform inner iterations
% for the projection function
% to satisfy the no-slip
% boundary condition
%==============================

 for inner=1:slipN

%---
% zero the tridiagonal matrix
%---

 for i=1:Nx-1
   atr(i) = 0.0;
   btr(i) = 0.0;
   ctr(i) = 0.0;
 end

%--------------------------------------
%  Integrate conv-diff equation in x
%  using the Crank-Nicolson method
%  Advance the velocity from u^n to u*
%--------------------------------------

  Iskip = 0;
  if(Iskip==0)

  al = nu*Dt/Dxs;

%--------------
  for j=2:Ny          % run over rows
%----

    for i=2:Nx
      Rc = ux(i,j)*Dt/Dx;
      C1 =   Rc + 2.0*al;
      C2 =        4.0*(1.0-al);
      C3 =  -Rc + 2.0*al;
      ctr(i-1) =  -Rc - 2.0*al;
      atr(i-1) =        4.0*(1.0+al);
      btr(i-1) =   Rc - 2.0*al;
      rhsx(i-1) = C1*ux(i-1,j) ...
                + C2*ux(i,j) ...
                + C3*ux(i+1,j);
      rhsy(i-1) = C1*uy(i-1,j)  ...
                + C2*uy(i,j) ...
                + C3*uy(i+1,j);
    end

    rhsy(1)    = rhsy(1) ...
               - ctr(1)   *BCyl(j);
    rhsy(Nx-1) = rhsy(Nx-1) ...
                - btr(Nx-1)*BCyr(j);

    % x component:
    solx = thomas(Nx-1,atr,btr,ctr,rhsx);
    % y component:
    soly = thomas(Nx-1,atr,btr,ctr,rhsy);

    for k=1:Nx-1
```

```
     uxi(k+1,j) = solx(k);
     uyi(k+1,j) = soly(k);
    end

%---
   end  % End of running over rows
%-----------

  for i=1:Nx-1     % reset
    atr(i) = 0.0;
    btr(i) = 0.0;
    ctr(i) = 0.0;
  end

 end  % of skip

%-------------------------------------
%  Integrate Conv-Diff equation in y
%  using the Crank-Nicolson method
%  Advance the velocity from u* to u**
%-------------------------------------

  Iskip = 0;
  if(Iskip==0)

  al = nu*Dt/Dys;

%---
  for i=2:Nx     % run over columns
%---
    for j=2:Ny     % from bottom to top
      Rc = uy(i,j)*Dt/Dy;
      C1 =  Rc +2.0*al;
      C2 =      4.0*(1.0-al);
      C3 = -Rc +2.0*al;
      ctr(j-1) = -Rc -2.0*al;
      atr(j-1) =      4.0*(1.0+al);
      btr(j-1) =  Rc -2.0*al;
      rhsx(j-1) = C1*uxi(i,j-1)  ...
                + C2*uxi(i,j) ...
                + C3*uxi(i,j+1);
      rhsy(j-1) = C1*uyi(i,j-1) ...
                + C2*uyi(i,j) ...
                + C3*uyi(i,j+1);
    end

    rhsx(1)    = rhsx(1)    ...
      - ctr(1)   *BCxb(i);
    rhsx(Ny-1) = rhsx(Ny-1)  ...
         - btr(Ny-1)*BCxt(i);

    solx = thomas (Ny-1,atr,btr,ctr,rhsx);
    soly = thomas (Ny-1,atr,btr,ctr,rhsy);

    for k=1:Ny-1
      uxi(i,k+1) = solx(k);
      uyi(i,k+1) = soly(k);
    end
%---
  end   % of run over columns
%---

 end  % of skip

%-------------------------------------
% Compute intermediate compressibility
% by centered differences
%
%       Divus = Div u**
%-------------------------------------

% initialize

  for j=1:Ny+1
    for i=1:Nx+1
      Divus(i,j) = 0.0;
    end
  end

% interior nodes

  for i=2:Nx
    for j=2:Ny
      DuDx = (uxi(i+1,j)-uxi(i-1,j))/Dx2;
      DvDy = (uyi(i,j+1)-uyi(i,j-1))/Dy2;
      Divus(i,j) = DuDx+DvDy;
     end
   end

% left wall

  for j=1:Ny+1
   DuDx = (-3.0*uxi(1,j) ...
     +4.0*uxi(2,j)-uxi(3,j))/Dx2;
   DvDy = 0.0;
   Divus(1,j) = DuDx+DvDy;
  end
   % save for corners:
```

```
   save11l = Divus(1,j);
   save12l = Divus(1,Ny+1);

% bottom wall

   for i=1:Nx+1
     DuDx = 0.0;
     DvDy = (-3.0*uyi(i,1) ...
       +4.0*uyi(i,2)-uyi(i,3))/Dy2;
     Divus(i,1) = DuDx+DvDy;
   end

   % save for corners:
   save11b = Divus(1,1);
   save21b = Divus(Nx+1,1);

% right wall

   for j=1:Ny+1
     DuDx = (3.0*uxi(Nx+1,j) ...
            -4.0*uxi(Nx,j) ...
            +uxi(Nx-1,j))/Dx2;
     DvDy = 0.0;
     Divus(Nx+1,j) = DuDx+DvDy;
   end

   % save for corners:
   save21r = Divus(Nx+1,1);
   save22r = Divus(Nx+1,Ny+1);

% top wall

   for i=1:Nx+1
     DuDx = 0.0;
     DvDy = (3.0*uyi(i,Ny+1) ...
        -4.0*uyi(i,Ny) ...
            +uyi(i,Ny-1))/Dy2;
     Divus(i,Ny+1) = DuDx+DvDy;
   end

   % save for corners:
   save12t = Divus(1,   Ny+1);
   save22t = Divus(Nx+1,Ny+1);

% corners by averaging

   Divus(1,      1) ...
      = 0.5*(save11l+save11b);
   Divus(1,   Ny+1) ...
      = 0.5*(save12l+save12t);
   Divus(Nx+1,   1) ...
      = 0.5*(save21b+save21r);
   Divus(Nx+1,Ny+1)  ...
      = 0.5*(save22r+save22t);

%----------------------------------
% Solve for the projection function
% by Gauss-Seidel (GS) iterations
%----------------------------------

   Iskip = 0;
   if(Iskip==0)

%---
% source term
%-----

   for i=1:Nx+1
    for j=1:Ny+1
      source(i,j) = -Divus(i,j)/Dtor;
     end
   end

  [chi,iter,Iflag] = pois_gs_nnnn ...
 ...
    (Nx,Ny,Dx,Dy,source ...
    ,itermax,tol,relax ...
    ,qleft,qright,qbot,qtop,chi,Ishift);

  if(Iflag==0)
   disp "cvt_pm: Poisson solver ...
      did not converge"
   break
  end

%----------------------------------
% project the velocity at all nodes
% except at the corner nodes
%----------------------------------

%---
% interior nodes
%----

  for i=2:Nx
    for j=2:Ny
     DchiDx = (chi(i+1,j)-chi(i-1,j))/Dx2;
     DchiDy = (chi(i,j+1)-chi(i,j-1))/Dy2;
     uxi(i,j) = uxi(i,j) - Dtor*DchiDx;
     uyi(i,j) = uyi(i,j) - Dtor*DchiDy;
    end
  end

%--------------
% lower boundary
%
% Use forward differences with special
% treatment of the near-corner nodes
%--------------

   for i=2:Nx
      DchiDy = 0.0;
      if(i==2)
       DchiDx = (-3.0*chi(2,1) ...
            +4.0*chi(3,1)-chi(4,1))/Dx2;
      elseif(i==Nx)
       DchiDx =-(-3.0*chi(Nx,1) ...
           +4.0*chi(Nx-1,1) ...
           -chi(Nx-2,1))/Dx2;
      else
       DchiDx  ...
         = (chi(i+1,1)-chi(i-1,1))/Dx2;
      end
      uxi(i,1) = BCxb(i) - Dtor*DchiDx;
      uyi(i,1) =         - Dtor*DchiDy;
    end

%--------------
% upper boundary
%
% Use backward difference with special
% treatment of the near-corner nodes
%--------------

   for i=2:Nx
    DchiDy = 0.0;
    if(i==2)
      DchiDx = (-3.0*chi(2,Ny+1) ...
        +4.0*chi(3,Ny+1) ...
        -chi(4,Ny+1))/Dx2;
    elseif(i==Nx)
      DchiDx =-(-3.0*chi(Nx,Ny+1) ...
        +4.0*chi(Nx-1,Ny+1) ...
       -chi(Nx-2,Ny+1))/Dx2;
    else
      DchiDx = (chi(i+1,Ny+1) ...
              -chi(i-1,Ny+1))/Dx2;
```

```
     end
    uxi(i,Ny+1) = BCxt(i) - Dtor*DchiDx;
    uyi(i,Ny+1) =         - Dtor*DchiDy;
   end

%-------------
% left boundary
%
% Use forward differences with special
% treatment of the near-corner nodes
%-------------

   for j=2:Ny
      DchiDx = 0.0;
      if(j==2)
        DchiDy = (-3.0*chi(1,2) ...
          +4.0*chi(1,3) ...
          -chi(1,4))/Dy2;
      elseif(j==Ny)
        DchiDy =-(-3.0*chi(1,Ny) ...
               +4.0*chi(1,Ny-1) ...
               -chi(1,Ny-2))/Dy2;
      else
        DchiDy  ...
         = (chi(1,j+1)-chi(1,j-1))/Dy2;
      end
      uxi(1,j) =         - Dtor*DchiDx;
      uyi(1,j) = BCyl(j) - Dtor*DchiDy;
   end

%--------------
% right boundary
%
% Use backward difference with special
% treatment of the near-corner nodes
%--------------

    for j=2:Ny
      DchiDx = 0.0;
      if(j==2)
        DchiDy = (-3.0*chi(Nx+1,2) ...
                  +4.0*chi(Nx+1,3) ...
                  -chi(Nx+1,4))/Dy2;
      elseif(j==Ny)
        DchiDy =-(-3.0*chi(Nx+1,Ny) ...
                  +4.0*chi(Nx+1,Ny-1) ...
                 -chi(Nx+1,Ny-2))/Dy2;
      else
        DchiDy = (chi(Nx+1,j+1) ...
                -chi(Nx+1,j-1))/Dy2;
      end
      uxi(Nx+1,j) =         - Dtor*DchiDx;
      uyi(Nx+1,j) = BCyr(j) - Dtor*DchiDy;
    end

  end % of Iskip

%-------------------------------------
% Compute the wall slip velocity
%
% and modify the boundary conditions
% for the intermediate (star) velocities
%-------------------------------------

    slipmax = 0.0;

% top and bottom:

    for i=1:Nx+1
     cor = uxi(i,Ny+1)-Vlid(i);
     if(abs(cor)>slipmax) ...
       slipmax = cor; end
     BCxt(i) = BCxt(i)-sliprel*cor;
      cor = uxi(i,1);
      if(abs(cor)>slipmax) ...
        slipmax = cor; end
      BCxb(i) = BCxb(i)-sliprel*cor;
    end

% left and right:

    for j=1:Ny+1
     cor = uyi(1,j);
     if(abs(cor)>slipmax) ...
       slipmax = cor; end
     BCyl(j) = BCyl(j)-sliprel*cor;
       corr = uyi(Nx+1,j);
       if(abs(cor)>slipmax) ...
         slipmax = cor; end
       BCyr(j) = BCyr(j)-sliprel*cor;
    end

    slipmax

    if(slipmax<sliptol) break; end

%=============================

  end % of inner iterations
%=============================

%-----------------------------------
% update velocity to the final value
%-----------------------------------

  for j=1:Ny+1
    for i=1:Nx+1
      ux(i,j) = uxi(i,j);
      uy(i,j) = uyi(i,j);
    end
  end

%------------
% update time
%------------

  time = time + Dt

%===========
 end  % of time stepping
%===========

figure(2)
mesh(x,y,ux);
set(gca,'fontsize',15)
xlabel('x','fontsize',15)
ylabel('y','fontsize',15)
zlabel('u_x','fontsize',15)
set(gca,'fontsize',15)
box

figure(3)
mesh(x,y,uy);
xlabel('x','fontsize',15)
ylabel('y','fontsize',15)
zlabel('u_y','fontsize',15)
set(gca,'fontsize',15)
box

figure(4)
mesh(x,y,chi,chroma);
xlabel('x','fontsize',15)
ylabel('y','fontsize',15)
zlabel('\chi','fontsize',15)
set(gca,'fontsize',15)
box
```

The code calls the function *thomas* listed in Section 8.2.4 to solve tridiagonal systems of equations using the Thomas algorithm.

The code also calls the following MATLAB function entitled *pois_gs_nnnn* to solve the Poisson equation for the projection function, subject to the Neumann boundary condition along the four sides:

```
function [f,iter,Iflag] = pois_gs_nnnn ...
...
    (Nx,Ny,Dx,Dy,g ...
    ,itermax,tol,relax,qleft ...
      ,qright,qbot,qtop,f,Ishift)

%-------------------------------------------
% Solution of Poisson's equation
% in a rectangular domain
% with the uniform Neumann boundary condition
% along the four sides:
%
% bottom:  df/dy = qbot
% top:  df/dy = -qtop
% left:  df/dx = qleft
% right:  df/dx = -qright
%
% The solution is found by
% point Gauss--Seidel iterations
%-------------------------------------------

%--------
% prepare
%--------

Dx2 = 2.0*Dx;
Dy2 = 2.0*Dy;
Dxs = Dx*Dx;
Dys = Dy*Dy;
beta = Dxs/Dys;
beta1 = 2.0*(beta+1.0);
Iflag = 0;     % convergence flag, 1 indicates convergence

%------------------------
% Gauss-Seidel iterations
%------------------------

for iter=1:itermax

%------------------------
% update nodes row-by-row
%------------------------
```

```
fsv = f;
cormax = 0.0;

%---
% interior nodes
%---

for j=2:Ny
  for i=2:Nx
    res = (f(i+1,j)+f(i-1,j)+beta*(f(i,j+1)+f(i,j-1)) ...
    + Dxs*g(i,j))/beta1-f(i,j);
    f(i,j) = f(i,j) + relax*res;
  end
end

%--------------
% left boundary
%--------------

i=1;
for j=2:Ny
  res = (2*f(i+1,j)-Dx2*qleft+beta*(f(i,j+1)+f(i,j-1)) ...
    +Dxs*g(i,j))/beta1 - f(i,j);
  f(i,j) = f(i,j) + relax*res;
  end
end

% corner points:

j=1;
res = (2*f(i+1,j)-Dx2*qleft ...
    +beta*(f(i,j+1)+f(i,j+1)-Dy2*qbot) ...
    +Dxs*g(i,j))/beta1 - f(i,j);
f(i,j) = fsave(i,j) + relax*res;

j=Ny+1;
res = (2*f(i+1,j)-Dx2*qleft ...
    +beta*(f(i,j-1)+f(i,j-1)-Dy2*qtop) ...
    +Dxs*g(i,j))/beta1 - f(i,j);
f(i,j) = fsave(i,j) + relax*res;

%---------------
% right boundary
%---------------

i=Nx+1;
for j=2:Ny
  res = (2*f(i-1,j)+-Dx2*qright+beta*(f(i,j+1)+f(i,j-1)) ...
    +Dxs*g(i,j))/beta1 -f(i,j);
  f(i,j) = fsave(i,j) + relax*res;
end

% corner points:

j=1;
res = (2*f(i-1,j)-Dx2*qright ...
    +beta*(f(i,j+1)+f(i,j+1)-Dy2*qbot) ...
    +Dxs*g(i,j))/beta1 - f(i,j);
f(i,j) = fsave(i,j) + relax*res;

j=Ny+1;
res = (2*f(i-1,j)-Dx2*qright)+beta*(2*f(i,j-1)-Dy2*qtop) ...
    +Dxs*g(i,j))/beta1 - f(i,j);
f(i,j) = fsave(i,j) + relax*res;

%----------------
% bottom boundary
%----------------

j=1;
for i=2:Nx
  res = (f(i+1,j)+f(i-1,j)+beta*(2*f(i,j+1)-Dy2*qbot) ...
      +Dxs*g(i,j))/beta1 - f(i,j);
  f(i,j) = fsave(i,j) + relax*res;
end

%-------------
% top boundary
%-------------

j=Ny+1;
for i=2:Nx
  res = (f(i+1,j)+f(i-1,j)+beta*(2*f(i,j-1)-Dy2*qtop) ...
      +Dxs*g(i,j))/beta1 - f(i,j);
  f(i,j) = fsave(i,j) + relax*res;
end

%------
% shift
%------

if(Ishift==1)
  shift = f(Nx/2,Ny/2);
  for i=1:Nx+1
    for j=1:Ny+1
      f(i,j) = f(i,j)-shift;
    end
  end
end

%-------------------
% maximum correction
%-------------------

cormax = 0;

for i=1:Nx+1
  for j=1:Ny+1
    cor = abs(f(i,j)-fsv(i,j));
    if(abs(cor)>cormax) cormax = cor; end
  end
end

%-----
% stopping check
%-----

if(cormax<tol)
  Iflag = 1;
  break
end

%---
end     % of iterations
%---

%-----
% done
%-----

return
```

The graphics display generated by the code for the parameter values implemented in the code is shown in Figure 8.7.1. In the early stages of the motion, the flow is similar to that generated by the impulsive translation of a plate in a semi-infinite fluid. At later times, a fully developed recirculating flow is established.

8.7.5 Computation of the pressure

Two methods are available for extracting the pressure field, if desired. The first method involves combining equations (8.7.9), (8.7.12), and (8.7.16)–or any other appropriate set of equations–to derive a relationship between $\mathbf{u}(t)$ and $\mathbf{u}(t + \Delta t)$. Requiring that this relationship reduces to a spatially discretized version of the Navier–Stokes equation in the limit as Δt tends to zero, we derive an expression for an effective pressure. If the boundary conditions satisfied by the effective pressure are consistent with the Neumann boundary

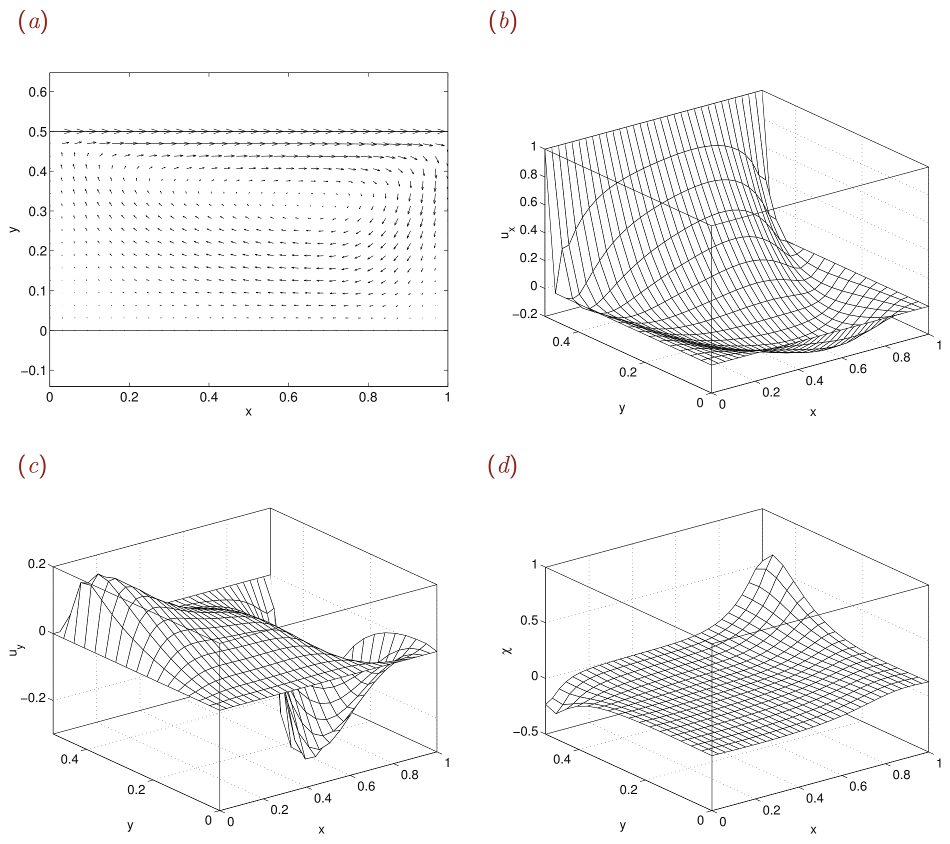


Figure 8.7.1 Flow in a rectangular cavity computed by a projection method. The graphs show (a) the velocity vector field, (b) the x velocity component, (c) the y velocity component, and (d) the projection function at steady state.

condition satisfied by the physical pressure, then the effective pressure can be accepted as an approximation to the physical pressure.

The second method involves substituting the computed velocity field into the Navier–Stokes equation and solving the resulting equation for the pressure subject to the Neumann boundary condition, as discussed in Section 8.6.2.

Problems

8.7.1 *Singular system for the projection function*

Show that equation (8.7.28) provides us with a solution of (8.7.27), subject to the homogeneous Neumann boundary condition.

8.7.2 💻 *Developing flow in a cavity*

Run the code *cvt_pm* located in directory *11_fdm* of FDLIBfor a cavity with aspect ratio $L_x/L_y = 8$ and other parameter values of your choice. Prepare velocity vector fields and discuss the structure of the flow and the performance of the numerical method.

8.8 Staggered grids

The derivation of explicit boundary conditions for the pressure can be bypassed by using a staggered grid consisting of two superposed displaced grids whose intersections define nodes where the velocity components or pressure is defined. The methodology is illustrated in this section with reference to steady two-dimensional Stokes flow in a rectangular cavity driven by a moving lid.

At sufficiently low Reynolds numbers, the motion of the fluid is governed by the equations of Stokes flow, including the continuity equation and the Stokes equation,

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0, \qquad -\boldsymbol{\nabla} p + \mu\nabla^2\mathbf{u} + \rho\mathbf{g} = \mathbf{0}, \tag{8.8.1}$$

where $\mathbf{g}$ is the acceleration of gravity imparting a body force.

Pressure and velocity nodes

The staggered grid consists of two interwoven grids parametrized by two pairs of indices, (i, j) and (i', j'), as illustrated in Figure 8.8.1 where the primed indices are printed in bold. The grid lines of the primary grid are represented by the solid lines and the grid lines of the secondary grid are represented by the broken lines. Note that the secondary grid conforms with the physical boundaries of the flow.

Discrete values of the pressure are assigned to the primary nodes, (i, j), defined by the intersection of the solid lines, shown as circles.

Discrete values of the x component of the velocity are defined at the intersection of horizontal primary grid lines and vertical secondary grid lines, (i', j), shown as horizontal arrows. The x-velocity node labeled $(2, 2)$ is shown with a circled horizontal arrow in Figure 8.8.1.

Discrete values of the y component of the velocity are defined at the intersection of vertical primary grid lines and horizontal secondary grid lines, (i, j'), shown as vertical arrows. The y-velocity node labeled $(2, 2)$ is shown with a circled vertical arrow in Figure 8.8.1.

Finite-difference equations

A distinguishing feature of the staggered grid method is that the governing equations are enforced at different nodes. For convenience, we denote u_x by u and u_y by v.

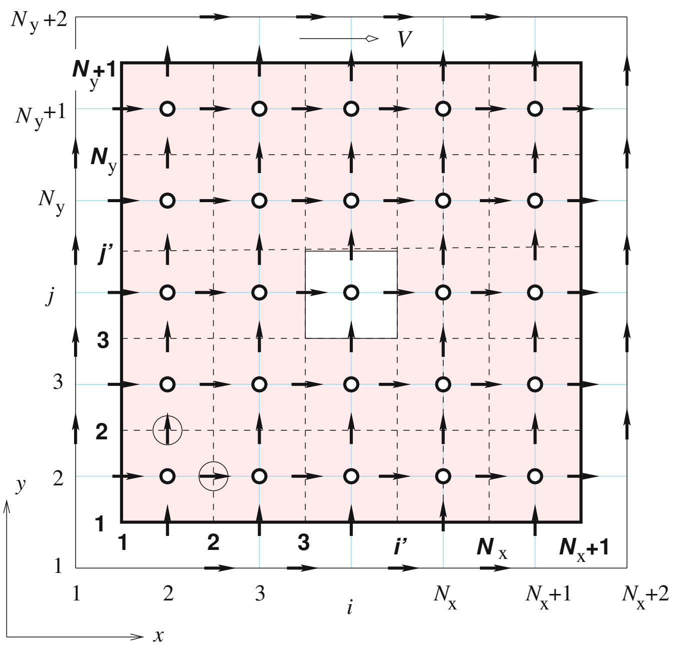


Figure 8.8.1 Illustration of a staggered grid for computing two-dimensional flow in a rectangular cavity. The pressure is defined at nodes indicated by the circles, the x velocity component is defined at nodes indicated by the horizontal arrows, and the y velocity component is defined at nodes indicated by the vertical arrows.

Applying the x component of the Stokes equation at the (i,j) interior x-velocity node and introducing similar difference approximations, we obtain

$$\frac{p_{i+1,j}-p_{i,j}}{\Delta x}=\mu\,\Big(\frac{u_{i-1,j}-2\,u_{i,j}+u_{i+1,j}}{\Delta x^2}+\frac{u_{i,j-1}-2\,u_{i,j}+u_{i,j+1}}{\Delta y^2}\Big) \tag{8.8.2}$$

for $i=2,\ldots,N_x$ and $j=2,\ldots,N_y+1$, providing us with $(N_x-1)\times N_y$ equations.

Applying the y component of the Stokes equation at the (i,j) interior y-velocity nodes and working in a similar fashion, we obtain

$$\frac{p_{i,j+1}-p_{i,j}}{\Delta y}=\mu\,\Big(\frac{v_{i-1,j}-2\,v_{i,j}+v_{i+1,j}}{\Delta x^2}+\frac{v_{i,j-1}-2\,v_{i,j}+v_{i,j+1}}{\Delta y^2}\Big) \tag{8.8.3}$$

for $i=2,\ldots,N+1$ and $j=2,\ldots,N$, providing us with $N_x\times(N_y-1)$ equations.

Enforcing the continuity equation at the (i,j) pressure node and approximating the partial derivative with central differences, we obtain

$$\frac{u_{i,j}-u_{i-1,j}}{\Delta x}+\frac{v_{i,j}-u_{i,j-1}}{\Delta y}=0 \tag{8.8.4}$$

for $i = 2, \ldots, N_x + 1$ and $j = 2, \ldots, N_y + 1$, providing us with $N_x \times N_y$ equations.

We have derived a total of

$$N = (N_x - 1)N_y + N_x(N_y - 1) + N_x N_y \tag{8.8.5}$$

difference equations involving interior and phantom exterior velocity nodes, as shown with arrows in Figure 8.8.1.

Boundary conditions

Unphysical x velocity components are required at the horizontal lines $j = 1$ and $j = N_y + 2$ for computing the second y derivative of u near the top and bottom boundaries. Corresponding unphysical y velocity components are required at the vertical levels $i = 1$ and $N_x + 2$ for computing the second x derivative of v near the left and right boundaries. These exterior velocities are computed by extrapolation to satisfy the boundary conditions at the physical levels $i' = 1$, $i' = N_x + 1$, $j' = 1$, and $j' = N_y + 1$.

For example, approximating u with a parabola near the lid located at $y = b_y$, and enforcing the no-slip boundary condition $u = V$, we obtain

$$u = V + A\,(y - b_y)^2 + B\,(y - b_y), \tag{8.8.6}$$

where V is the lid velocity and A, B are unknown coefficients. Applying this expression at three neighboring grid levels, we obtain,

$$u_{i,N_y} = V + A\,\tfrac{9}{4}\,h^2 - B\,\tfrac{3}{2}\,h, \qquad u_{i,N_y+1} = V + A\,\tfrac{1}{4}\,h^2 - B\,\tfrac{1}{2}\,h,$$
$$u_{i,N_y+2} = V + A\,\tfrac{1}{4}\,h^2 + B\,\tfrac{1}{2}\,h, \tag{8.8.7}$$

where $h = \Delta y$. Eliminating A and B, we obtain the velocity at the exterior node,

$$u_{i,N_y+2} = \tfrac{1}{3}\,(8\,V - 6\,u_{i,N_y+1} + u_{i,N_y}). \tag{8.8.8}$$

Similar expressions can be derived for the other x and y external velocities.

Code

The preceding difference equations provide us with a complete system of linear algebraic equations for the nodal velocities and pressures. The system can be compiled and solved at once, as illustrated in the following MATLAB code entitled *cvt_stag*, located in directory *11_fdm* of FDLIB:

```
%==========
% steady Stokes flow in a rectangular cavity
% occupying 0<x<Lx, 0<y<Ly
% computed on a staggered Cartesian grid
%======
Lx = 1.0;
Ly = 0.75;

Nx = 24;   % x divisions
Ny = 16;   % y divisions
visc = 1.0;   % viscosity
Vlid = 1;   % lid velocity

%---
% prepare
%---

Dx = ax/Nx; Dy = ay/Ny;
Dxs = Dx*Dx; Dys = Dy*Dy;

%---
% initialize
%---

% matrix size (no of eqs):
mats = (Nx-1)*Ny + Nx*(Ny-1) + Nx*Ny;

u = zeros(Nx+1,Ny+1);   % impulse matrix for ux
v = zeros(Nx+1,Ny+1);   % impulse matrix for uy
p = zeros(Nx+1,Ny+1);   % impulse matrix for p

%=======
% compile the coefficient matrix (MAT)
%=======

Jc = 0;   % counter of impulses

%---
for ipass=1:3   % impulse for u,v,p
%---

if(ipass==1)
  klim = Nx; llim = Ny+1;   % u vel
elseif(ipass==2)
  klim = Nx+1; llim = Ny;   % v vel
elseif(ipass==3)
  klim = Nx+1; llim = Ny+1;   % pressure
end

%---
for l=2:llim   % scan
  for k=2:klim
  Jc=Jc+1;
%---
```

```
  if(ipass==1); % impulse
    u(k,l) = 1.0;
  elseif(ipass==2);
    v(k,l) = 1.0;
  elseif(ipass==3);
    p(k,l) = 1.0;
  end

%---
% boundary conditions
%---

  for ii=2:Nx
    u(ii,1)= (-6*u(ii,2)+u(ii,3))/3.0;
    u(ii,Ny+2) = (-6*u(ii,Ny+1)+u(ii,Ny))/3.0;
  end
  for jj=2:Ny
    v(1,jj) = (-6*v(2,jj)+v(3,jj))/3.0;
    v(Nx+2,jj) = (-6*v(Nx+1,jj)+v(Nx,jj))/3.0;
  end
%---

  Ic=0;   % counter of equations

  for j=2:Ny+1
    for i=2:Nx   % x Stokes
      Ic = Ic+1;
      MAT(Ic,Jc) = -(p(i+1,j)-p(i,j))/Dx ...
        +visc*(u(i-1,j)-2*u(i,j)+u(i+1,j))/Dxs ...
        +visc*(u(i,j-1)-2*u(i,j)+u(i,j+1))/Dys;
      MAT(Ic,Jc) = MAT(Ic,Jc)*Dxs;
    end
  end

  for j=2:Ny
    for i=2:Nx+1   % y Stokes
      Ic = Ic+1;
      MAT(Ic,Jc) = -(p(i,j+1)-p(i,j))/Dy ...
        +visc*(v(i-1,j)-2*v(i,j)+v(i+1,j))/Dxs ...
        +visc*(v(i,j-1)-2*v(i,j)+v(i,j+1))/Dys;
      MAT(Ic,Jc) = MAT(Ic,Jc)*Dxs;
    end
  end
  for j=2:Ny+1
    for i=2:Nx+1   % continuity
      Ic = Ic+1;
        MAT(Ic,Jc) = (u(i,j)-u(i-1,j))/Dx ...
        +(v(i,j)-v(i,j-1))/Dy;
        MAT(Ic,Jc) = MAT(Ic,Jc)*Dx;
    end
  end

  u = zeros(Nx+1,Ny+1);   % reset
  v = zeros(Nx+1,Ny+1);
  p = zeros(Nx+1,Ny+1);

  end   % of for k
  end   % of for l

end   % of ipass

%====
% set the right-hand side
%===

for i=1:mats
  rhs(i) = 0.0;
end

Ic = (Ny-1)*(Nx-1);
for i=2:Nx         % x Stokes
  Ic = Ic+1;
  rhs(Ic) = rhs(Ic)-8.0*visc*Vlid*Dxs/3/Dys;
end

%====
% set p=0 at the last node
% and find the solution
%====

SOL = rhs(1:mats-1)/MAT(1:mats-1,1:mats-1)';
SOL(mats) = 0.0;

%====
% distribute the solution
%====

Ic = 0;
for j=2:Ny+1
  for i=2:Nx
    Ic = Ic+1;
    uvel(i,j) = SOL(Ic);
  end
end

for j=2:Ny
  for i=2:Nx+1
```

```
    Ic=Ic+1;
    vvel(i,j) = SOL(Ic);
  end
end

for j=2:Ny+1
  for i=2:Nx+1
    Ic = Ic+1;
    pressure(i,j) = SOL(Ic);
  end
end

%=====
% interpolate the velocity field at the pressure nodes
% for plotting purposes
%=====

for j=2:Ny+1
  uvel(1,j) = 0;
  uvel(Nx+1,j) = 0;
  for i=2:Nx+1
    uint(i,j) = 0.5*(uvel(i,j)+uvel(i-1,j));
  end
end

for i=2:Nx+1
  vvel(i,1) = 0;
  vvel(i,Ny+1) = 0;
  for j=2:Ny+1
    vint(i,j) = 0.5*(vvel(i,j)+vvel(i,j-1));
  end
end

%====
% prepare a velocity vector plot
%====

figure(1)
hold on
plot([0 Lx Lx 0 0],[0 0 Ly Ly 0])
axis([-0.1*Lx 1.1*Lx, -0.1*Lx 1.1*Lx])
axis equal
xlabel('x'); ylabel('y')

for j=2:Ny+1
  ylevel = 0.5*Dy+(j-2)*Dy;
  for i=2:Nx+1
    xlevel = 0.5*Dx+(i-2)*Dx;
    xarrow = 0.75*Dx*uint(i,j)/Vlid;
    yarrow = 0.75*Dx*vint(i,j)/Vlid;
    arrow = arrow_cp(xlevel,ylevel,xarrow,yarrow);
    plot(arrow(:,1),arrow(:,2));
  end
end
```

Velocity vector fields generated by the code for cavities with different aspect ratios are shown in Figure 8.8.2.

Particulars of the implementation

The implementation of the numerical procedure features the following particulars:

- The linear system is compiled so that the first block of equations encapsulates the x component of the Stokes equation, the second block encapsulates the y component of the Stokes equation, and the third block encapsulates the continuity equation.
- The first block of unknowns encapsulates the x velocity components, the second block encapsulates the y velocity components, and the third block encapsulates the pressure.
- To generate the coefficient matrix of the linear system, we sequentially set one nodal value of the x velocity component, y velocity component, or pressure to unity, while holding all other values to zero. The corresponding column of the coefficient matrix is given by the residual of the governing equations scanned in the aforementioned order.
- Since the pressure field is defined up to an arbitrary constant, the pressure is set arbitrarily to zero at the last pressure node. One equation expressing the discrete implementation of the continuity equation is then discarded to balance the number of equations to the number of unknowns.

After the computation has been completed, the code calls the custom-made function *arrow_cp* near the end to draw nice-looking arrows.

Summary

By using a staggered grid, we have been able to circumvent the explicit derivation of boundary conditions for the pressure. A careful analysis shows that implicit in the numerical formulation is the Neumann boundary condition that arises by projecting the equation of motion normal to the boundaries.

Unfortunately, the staggered-grid method becomes considerably more involved and prohibitively expensive when applied to grids defined in general curvilinear coordinates.

PROBLEMS

8.8.1 *Coefficient matrix*

(a) Present a pictorial depiction of the coefficient matrix arising from the difference equations and identify large non-zero blocks.

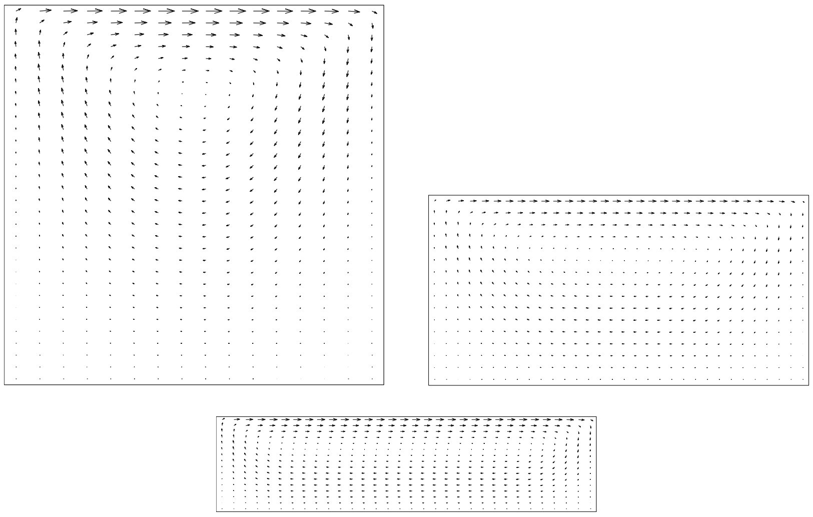


Figure 8.8.2 Velocity vector fields of two-dimensional Stokes flow in cavities with different aspect ratios driven by a sliding lid computed on a staggered grid using the FDLIB code *cvt_stag*.

(*b*) Verify that the coefficient matrix of the linear system is singular.

8.8.2 *Pressure distribution*

Modify the graphics portion of code *cvt_stag* to visualize the pressure field. Present and discuss contour plots of the pressure field for flow in a square cavity.