High Order Wide and Compact Schemes for the Steady Incompressible Navier-Stokes Equations

by

Allan Wittkopf

B.Sc., Okanagan University–College, 1991

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the Department
of
Mathematics & Statistics

SIMON FRASER UNIVERSITY
May 1999

APPROVAL

Name: Allan Wittkopf

Degree: Master of Science

Title of thesis: High Order Wide and Compact Schemes for the Steady Incompressible Navier-Stokes Equations

Examining Committee: Dr. S. K. Thomason
Chair

Dr. T. Tang
Senior Supervisor

Dr. R. D. Russell

Dr. R. W. Lardner

Dr. H. Huang
External Examiner

Date Approved:

Abstract

This thesis is concerned with solving the steady two dimensional Navier-Stokes equations using finite difference methods. It has been discovered that although central difference approximations are locally second-order accurate they often suffer from computational instability and the resulting solutions exhibit nonphysical oscillations. Although first-order and second-order upwind difference approximations are computationally stable, the resulting solutions exhibit the effects of artificial viscosity. As a result of this, there has been great interest in recent years to investigate high-order schemes.

In this thesis, two fourth-order accurate finite difference schemes are obtained for the Navier-Stokes equations expressed in streamfunction alone. The first one is the conventional fourth-order central difference scheme with a stencil extending over 29 points. The second one is of compact type with a stencil extending over a 5×5-square of points. This method is more efficient for solving the discrete non-linear system by Newton's method. We consider a number of test problems, including the driven cavity problem, to compare the two fourth-order schemes to each other, as well as second-order and fourth-order benchmark solutions. In spite of its wider stencil, it is found that the conventional scheme has sufficiently lower error for high Reynolds number than the compact scheme, making up for its greater width. The effects of numerical boundary conditions on convergence are also investigated.

Acknowledgements

I am deeply indebted to my senior supervisor, Dr. Tao Tang for his guidance, encouragement and patience during the preparation of this thesis.

I would also like to thank Dr. Robert Russell, and Dr. Cecil Graham for their guidance throughout my masters degree, as well as my advisory committee for their helpful comments on my thesis.

Thanks also go to Mrs. Sylvia Holmes and Mrs. Maggie Fankboner for their assistance on the administrative end of things.

Finally, financial support from Simon Fraser University is also appreciated.

Contents

Chapter 1

Introduction

In many physical problems, systems of partial differential equations may be derived from basic principles to accurately model the behavior of the physical system under consideration. Unfortunately in many fields of study, particularly that of fluid dynamics, these systems of partial differential equations are non-linear and may not be solvable in closed form, forcing one to rely on approximate techniques.

Finite difference methods may be used to reduce a non-linear partial differential equation to a system of non-linear equations with a finite number of unknowns, which must then be solved to obtain the desired approximate solution. These methods begin with a *discretization* of the domain, that is they place some grid over the domain, and apply Taylor series to the partial differential equation at each grid point. This procedure, which results in an equation to be solved at each grid point in the domain, is named the scheme. As well as giving approximate solutions, these methods provide a bound for the inaccuracy of the obtained solution due to the particular scheme and grid used. The order of accuracy is defined by the manner in which the error decreases as the grid is uniformly refined. A scheme is said to be n-th order accurate, if for sufficiently small h, $Error = Ch^n$, where h is some representative grid spacing and C, which is independent of h, is a constant with respect to the problem.

Initially first and second order accurate schemes were widely used, but disadvantages such as the requirement of a fine mesh, and hence many equations to be solved in order to obtain accurate solutions, have motivated researchers to obtain better schemes. It became clear that schemes with a higher order of accuracy were needed, but they required a larger computational stencil causing a substantial decrease in the efficiency in obtaining a solution for the resulting system. It was found that by careful use of differential extensions of the original partial differential equations, higher order schemes, named *compact* schemes, could be obtained, giving a higher order of accuracy without significantly altering the computational stencil.

When modelling fluid flow phenomena, it is found that these compact schemes give good agreement with experimental data for low to mid–velocity flows, even using a relatively coarse mesh. However high–velocity flows require a fine mesh for reasonable agreement.

In this thesis we shall compare the accuracy for a wide and a compact scheme at a range of flow velocities, beginning with a scalar form of the Navier–Stokes equations which is not commonly used. These schemes are derived analytically, and alternatively by use of a symbolic programming language such as Maple. The use of symbolic derivation can allow the programmer to constrain the scheme in such a way that desired properties, such as compactness and diagonal dominance, may be obtained. This strategy also has advantages for long or complex schemes as it may be made to output the schemes in the form of the language used for computation, as well as avoiding a long analytic derivation. A close examination of the effects of boundary conditions and the efficiency of solution techniques will also be presented.

We begin with a review of known work in Chapter 2, first presenting the governing equations, and the underlying assumptions used to obtain them in Section 2.1. We then proceed with a detailed description of the derivation of difference formulae, and two widely used solution techniques for large systems of equations in Section 2.2. The advantages and disadvantages of finite difference schemes which have been successfully used to compute flows are then discussed in Section 2.3.

Chapter 3 begins with the derivation of the wide and compact schemes and boundary conditions to be studied, and symbolic derivations are presented as an alternative to the more popular analytic derivation of schemes in Section 3.1. The model problems used to test the schemes, as well as solution techniques, and scheme-boundary condition combinations are presented in Section 3.2. The results obtained by application of the schemes to problems for which the solutions are known are then discussed with regards to accuracy, boundary conditions and efficiency in Section 3.3. Comparisons of these schemes applied to a driven cavity problem with known benchmark solutions are presented in Section 3.4 discussing the difference between our compact and wide schemes.

Finally the discussion and conclusions are presented in Chapter 4.

Chapter 2

Governing Equations and Solution Methods

2.1 Governing Equations

In order to provide physical background for the equations to be solved, we begin with a description of the governing equations of fluid dynamics, explaining the assumptions inherent in their derivation. We then narrow the class of solutions of those we study, also providing a physical interpretation for the restrictions required to do so.

2.1.1 Continuum Hypothesis and Conservation Principles

Although the mass in a fluid is concentrated in the nuclei of the atoms composing it, description of the flow from this microscopic viewpoint would be impossible due to the large number of molecules present even in a small quantity of fluid. In order to obtain the equations describing the macroscopic behavior of a fluid, one must define a fluid element as a small volume, and define any macroscopic properties of that element in terms of an average of the microscopic properties of the molecules within that volume. This viewpoint gives continuously distributed properties to a flow, and is called the *continuum hypothesis*. In this way, molecular velocities are broken down into two properties, flow velocity, which is the average velocity of the molecules, and temperature, which takes the energy due to the purely random velocities of the molecules into account. A purely microscopic phenomenon, such as diffusion, may be described in terms of one or more macroscopic phenomena, such as viscosity and thermal conductivity. For the definition of a fluid element to be valid, each element must contain a large number of molecules, otherwise defining the properties of that element as averages over the molecules in that volume would not be meaningful. This condition is reasonable for nearly all flows except those of rarefied gases.

We also note that we will be dealing with only *Newtonian* fluids, that is fluids for which the relation between the stress acting on the fluid, and the rate of strain is linear. For a detailed discussion of this property see Batchelor [2].

We now introduce the *material* or *convective* derivative of a quantity ϕ, where ϕ represents a local property of the fluid. This is defined by

$$\frac{D\phi}{Dt} := \frac{\partial \phi}{\partial t} + \nabla\phi \cdot \mathbf{u} \tag{2.1}$$

where $\mathbf{u}$ is the local velocity of the fluid. This derivative represents the time rate of change of ϕ moving with the fluid. This notation is useful for derivation of, and understanding of the governing equations.

Application of conservation of mass to a small volume element V yields the integral equation

$$\frac{d}{dt}\int_V \rho \, dV = -\int_S \rho \, \mathbf{u} \cdot \mathbf{n} \, dS \tag{2.2}$$

where ρ is the density of the fluid, and $\mathbf{n}$ is the outward normal of the surface S of the volume being considered. This equation simply states that the total change in mass in a given volume is equal to the mass flux on the surface of that volume. Using the divergence theorem and the fact that the volume is arbitrary yields the equation

$$\frac{D\rho}{Dt} + \rho\nabla \cdot \mathbf{u} = 0 \tag{2.3}$$

which is named the continuity equation.

Conservation of momentum gives us the vector equation

$$\rho\frac{Du_i}{Dt} = \rho F_i - \frac{\partial p}{\partial x_i} + \frac{\partial}{\partial x_j}(2\mu(e_{ij} - \frac{1}{3} \triangle \delta_{ij})) \tag{2.4}$$

where repeated indices are summed, F_i represents the i-th component of the body force acting on the fluid, p is the kinetic pressure, μ is the coefficient of viscosity, $e_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right)$

is the rate of strain tensor, and $\triangle = e_{ii} = \nabla \cdot \mathbf{u}$ is the rate of expansion. This equation is named the Navier-Stokes equation and may be interpreted as a balance of forces acting on an element of the fluid. We now give a physical interpretation of each term in (2.4). The term on the left describes the mass times the acceleration of a given particle of fluid. The first term on the right represents the effect of all body forces (such as gravity and electromagnetism), the second term on the right represents the force due to a pressure gradient, and the last term represents the force due to a non-isotropic stress in the fluid (such as shearing motion or a coupled tension-compression).

Conservation of energy gives an equation involving temperature, density, and stresses. When coupled with an equation of state, closure of the system is obtained. Since we are concerned with flows for which temperature variation has little effect on density, pressure, and viscosity, or flows for which the temperature is nearly uniform, these equations are not presented here.

2.1.2 Incompressibility

Flows for which the density of a given element of fluid remains constant throughout its motion are called *incompressible* flows. This can be described mathematically as

$$\frac{D\rho}{Dt} = 0,$$

and can greatly simplify the calculation of fluid flows. The conditions which must be met for incompressibility are derived and discussed in great detail in Batchelor [2] and will be presented briefly here.

One condition which arises frequently in practice is on the ratio of the velocity U to the velocity of sound in the fluid c. This ratio is termed the mach number and denoted M, and the condition is

$$M^2 = \frac{U^2}{c^2} \ll 1,$$

where the largest value of $|\mathbf{u}|$ is chosen for U, and some representative value of c is used.

For oscillatory flow fields, where n is the dominant frequency, and L is a representative length scale, the condition

$$\frac{n^2 L^2}{c^2} \ll 1$$

must be met.

For atmospheric flows it is also required that the difference in static fluid pressures between the top and bottom of the region of interest is small compared to the absolute pressure. Near sea level this means the region of interest must be small compared to 8.4 km.

Given that none of these conditions are violated, the flow will behave incompressibly, and the continuity equation (2.3) becomes

$$\nabla \cdot \mathbf{u} = 0, \tag{2.5}$$

and the Navier-Stokes equations become

$$\rho \frac{D\mathbf{u}}{Dt} = \rho \mathbf{F} - \nabla p + \mu \nabla^2 \mathbf{u}. \tag{2.6}$$

2.1.3 The Two-Dimensional Streamfunction Vorticity Equations

Many problems for which there is some degree of translational symmetry can be reduced to two dimensions. In this case a simple non-dimensional form of the Navier-Stokes equations can be obtained. Without loss of generality we choose the direction of translational symmetry to be z and set $\mathbf{u} = (u, v, 0)$. Equation (2.5) becomes

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0$$

which suggests the existence of a streamfunction. We introduce the non-dimensional streamfunction ψ and the vorticity ζ as new variables where their relationship to the non-dimensional velocities u and v is given by the relations

$$-\frac{\partial \psi}{\partial x} = v, \qquad \frac{\partial \psi}{\partial y} = u, \qquad \zeta = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}. \tag{2.7}$$

We now consider that we are interested in the large time behavior, or *steady-state* behavior, of flows being acted upon by purely conservative forces. Using this information in conjunction with (2.7) gives the equations

$$\frac{\partial^2 \psi}{\partial x^2} + \frac{\partial^2 \psi}{\partial y^2} = -\zeta, \tag{2.8}$$

$$\frac{\partial^2 \zeta}{\partial x^2} + \frac{\partial^2 \zeta}{\partial y^2} = \mathrm{Re}\left(\frac{\partial \psi}{\partial y}\frac{\partial \zeta}{\partial x} - \frac{\partial \psi}{\partial x}\frac{\partial \zeta}{\partial y}\right), \tag{2.9}$$

where $\text{Re} = \frac{UL}{\nu}$ is the non-dimensional Reynolds number, U and L are characteristic velocity and length scales for the flow, and $\nu = \frac{\mu}{\rho}$ is the kinematic viscosity. Equations (2.8) and (2.9) are known as the *steady, two-dimensional, incompressible Navier-Stokes equations in streamfunction-vorticity form.*

Some difficulties may arise in numerically solving (2.9) as the boundary conditions are generally given in terms of ψ and $\frac{\partial\psi}{\partial n}$. In order to avoid this an equation for ψ alone can be derived by substitution of ζ in (2.8) into (2.9) giving

$$\frac{\partial^4\psi}{\partial x^4} + \frac{\partial^4\psi}{\partial y^2\partial x^2} + \frac{\partial^4\psi}{\partial y^4} + \text{Re}\left[\frac{\partial\psi}{\partial x}\left(\frac{\partial^3\psi}{\partial x^2\partial y} + \frac{\partial^3\psi}{\partial y^3}\right) - \frac{\partial\psi}{\partial y}\left(\frac{\partial^3\psi}{\partial x^3} + \frac{\partial^3\psi}{\partial x\partial y^2}\right)\right] = 0$$

which is named the *pure streamfunction form of the steady, two-dimensional, incompressible Navier-Stokes equations.* This equation may also be written in the form

$$\Delta\Delta\psi + \text{Re}\ (\psi_x\Delta\psi_y - \psi_y\Delta\psi_x) = 0, \tag{2.10}$$

where variable subscripts denote differentiation, and $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial x^2}$ is the Laplacian operator. This derivative notation will be used from this point forward.

2.2 Difference Approximations and Solution Techniques

2.2.1 Finite–Difference Formulae

In order to obtain approximate solutions for systems of P.D.E's a discretization of the domain of interest must be used. Using the values of the unknowns at the grid points defined by this discretization, approximations to the derivatives may be obtained using finite difference formulae. The following difference formulae are derived in [11] for an equal spaced grid with spacing h. All are centered differences written as 1-D stencils.

$$
\begin{array}{lclllllllllcl}
\psi_x & = & & [& -1 & 0 & 1 &] & & /2h & \psi & + & O(h^2) \\
\psi_x & = & [& 1 & -8 & 0 & 8 & -1 &] & /12h & \psi & + & O(h^4) \\
\psi_{xx} & = & & [& 1 & -2 & 1 &] & & /h^2 & \psi & + & O(h^2) \\
\psi_{xx} & = & [& -1 & 16 & -30 & 16 & -1 &] & /12h^2 & \psi & + & O(h^4) \\
\psi_{xxx} & = & [& -1 & 2 & 0 & -2 & 1 &] & /2h^3 & \psi & + & O(h^2) \\
\psi_{xxx} & = & [\ 1 & -8 & 13 & 0 & -13 & 8 & -1\ ] & /8h^3 & \psi & + & O(h^4) \\
\psi_{xxxx} & = & [& 1 & -4 & 6 & -4 & 1 &] & /h^4 & \psi & + & O(h^2) \\
\psi_{xxxx} & = & [\ -1 & 12 & -39 & 56 & -39 & 12 & -1\ ] & /6h^4 & \psi & + & O(h^4) \\
\psi_{xxxxx} & = & [\ -1 & 4 & -5 & 0 & 5 & -4 & 1\ ] & /2h^5 & \psi & + & O(h^2) \\
\psi_{xxxxxx} & = & [\ 1 & -6 & 15 & -20 & 15 & -6 & 1\ ] & /h^6 & \psi & + & O(h^2)
\end{array}
\tag{2.11}
$$

Many methods may be used to derive these difference formulae, such as Taylor series and polynomial interpolation, but all are fairly straightforward. A code which derives these formulae has been written in the symbolic programming language Maple, and has been included in Appendix 1. We now proceed with a short description of the code and its use.

The code uses Taylor series to obtain an approximation for the value of a one-dimensional derivative at $x = 0$. The command used to call the routine is

$>$ findif(*derivorder*,*pointslist*, *remainders*(optional));

where *derivorder* is the order of the derivative to be approximated, *pointslist* is a list of the x-coordinates of the function values to be used (given in grid spacings), and *remainders* is the number of remainder terms to be retained for output (default is 0). As an example the call

$>$ findif$(2, [-1, 0, 1], 1)$;

gives the approximation

$$U_{xx} = \frac{u(-1) - 2\,u(0) + u(1)}{h^2} - \frac{h^2}{12}\,U_{xxxx} + O(h^4)$$

giving the form of the second-order error term for this approximation in (2.11).

As another example, we examine the accuracy of a variable position finite difference approximation. Although it is well known that central differences are generally more accurate than one-sided differences, it may be useful to obtain the dependence of the accuracy on displacement from the center point of the approximation. To obtain a variable position

5-point approximation for U_x we use the call
$>$ findif$(1, [-2-n, -1-n, -n, 1-n, 2-n], 1)$;
where the resulting expression may be interpreted as an approximation to $U_x(nh)$, given the values $u(-2h), u(-h), u(0), u(h),$ and $u(2h)$. We may write this approximation as

$$U_x(nh) \;=\; U_x^c + U_{xx}^c \frac{(nh)^1}{1!} + U_{xxx}^c \frac{(nh)^2}{2!} + U_{xxxx}^c \frac{(nh)^3}{3!} + U_{xxxxx} \frac{(4-15n^2+5n^4)h^4}{120}$$

where the superscript c refers to the use of the central difference approximations given in (2.11). We may now use the form of this error term to obtain a ratio of the error for an approximation at $x = nh$ to that of the centered approximation at $x = 0$. This ratio r is given by

$$r \;=\; 1 + \frac{5}{4} n^2 (n^2 - 3).$$

For the value $n = 2$, which corresponds to a one-sided approximation, we see that $r = 6$, clearly demonstrating the increase in error resulting from the use of this approximation instead of a centered approximation.

Throughout this thesis we will denote fourth-order accurate differences with a $\tilde{D}_x$ and second-order with a D_x. Our fourth-order accurate discrete Laplacian is denoted $\tilde{\Delta}^h = \tilde{D}_{xx} + \tilde{D}_{yy}$.

2.2.2 Successive Over-Relaxation (S.O.R.)

S.O.R. is an iterative method which may be used to obtain solutions for discretized elliptic problems. Although S.O.R. requires relatively little memory and can be very efficient, *diagonal dominance* of the overall system is necessary to guarantee convergence to the solution. A sufficient condition for diagonal dominance which may be obtained using matrix methods follows (see, e.g. [30]). Writing an n point scheme in the form $\sum_{i=0}^{n-1} c_i \psi_i = b$ where b is independent of all ψ_i gives this condition as

$$|c_0| \geq \sum_{i=1}^{n-1} |c_i| \tag{2.12}$$

which must be satisfied for all equations in the system.

As an example we consider the Poisson equation

$$\Delta \psi \;=\; \psi_{xx} + \psi_{yy} \;=\; f \tag{2.13}$$

in a rectangular domain with fixed boundary conditions. Using central differences we discretize to second-order accuracy obtaining

$$\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j} \;=\; h^2 f_{i,j} \tag{2.14}$$

where h is the step size. Noting that $c_0 = -4$ and $c_1 = c_2 = c_3 = c_4 = 1$ shows that the condition for diagonal dominance (2.12) is met. S.O.R. proceeds from some initial guess $\psi^0_{i,j}$ by looping through all points i, j in the domain, replacing each $\psi^k_{i,j}$ with a corrected value, $\psi^{k+1}_{i,j}$. This process is repeated until

$$\sum_{i,j \in D} \left| \psi^{k+1}_{i,j} - \psi^k_{i,j} \right| < \epsilon \tag{2.15}$$

where ϵ is a fixed iterative error tolerance. The order in which the values are corrected has an effect on the efficiency of the process, and the standard ordering, called *lexographic* ordering, consists of an inner loop on i, and an outer loop on j (note: for a rigorous discussion of the effects of ordering on the nine-point Laplacian see [1]). Using this ordering, the formula for the corrected values may be written

$$\begin{aligned} \psi^{k+1}_{i,j} &= \psi^k_{i,j} + \lambda \left[\frac{1}{4} \left(\psi^k_{i+1,j} + \psi^{k+1}_{i-1,j} + \psi^k_{i,j+1} + \psi^{k+1}_{i,j-1} - h^2 f_{i,j} \right) - \psi^k_{i,j} \right] \\ &= \psi^k_{i,j} + \lambda \left[\psi^{corr}_{i,j} \right] \end{aligned} \tag{2.16}$$

where λ is termed the relaxation parameter, and $\psi^{corr}_{i,j}$ is the difference between the value of $\psi_{i,j}$ which would satisfy (2.14) and $\psi^k_{i,j}$.

When $\lambda = 1$, this reduces to the Gauss-Seidel method, and each $\psi^k_{i,j}$ is being replaced by the value obtained by solving (2.14) for $\psi_{i,j}$. The advantage of the S.O.R. method is that we may choose $1 < \lambda < 2$ with the assumption that overshooting the estimate for $\psi^{k+1}_{i,j}$ will converge to the solution more rapidly. This has proven to be the case for many problems, and when λ is chosen optimally, the number of iterations required to obtain convergence is dramatically reduced.

Further analysis of (2.16) shows that it may be viewed as a second-order accurate discretization of the equation

$$\psi_t \;=\; \psi_{xx} + \psi_{yy} - \frac{\Delta t}{h} \left(\psi_{xt} + \psi_{yt} \right) - f$$

with a time step size $\Delta t = \frac{\lambda h^2}{4 - 2\lambda}$. This equation has a steady-state solution consistent with (2.13), and the ψ_{xt} and ψ_{yt} terms are residual terms which result from the ordering used.

For a fixed h, the optimal value of λ can now be viewed as a balance between slow change due to a small time step, and the instability resulting from using too large a time step for an explicit numerical method.

One final consideration is application of S.O.R. to problems with derivative boundary conditions. For example we choose $\psi_x = g(y)$ along the $x = 0$ boundary which gives the first-order discretization

$$\frac{\psi_{1,j} - \psi_{0,j}}{h} = g_{0,j} + O(h),$$

giving our S.O.R. formula as

$$\psi_{0,j}^{k+1} = \psi_{0,j}^{k} + \mu \left[\psi_{1,j}^{k} - hg_{0,j} - \psi_{0,j}^{k}\right]$$

where μ is the boundary relaxation parameter. In some problems the application of derivative boundary conditions leads to instability of the S.O.R. algorithm. In these cases if μ is chosen to be small enough, the process will converge. This reduction in μ is called *boundary damping*, and though S.O.R. now converges, it is slowed considerably.

2.2.3 Newton's Method

Newton's method with LU-decomposition is a *direct* solution method used to obtain solutions for systems of non-linear equations. Although this method converges quadratically when close to the solution, it requires a sequence of matrix decompositions to do so, and for this reason it is considered computationally inefficient. Our description of the method begins with the system of equations to be solved:

$$f_i(\Phi) = 0 \quad for \quad i = 1...n \tag{2.17}$$

where $\Phi = (\phi_1, \phi_2, ..., \phi_n)$ are the unknowns to be solved for. With an initial guess Φ^0, a sequence of approximations for the solution of (2.17) is defined by

$$\Phi^{k+1} = \Phi^k + \Phi^{corr} \tag{2.18}$$

where Φ^{corr} is a correction for Φ^k. This correction is obtained as the solution of the linearization of (2.17) about Φ^k which is given by

$$J(\Phi^k)\Phi^{corr} = -F(\Phi^k) \tag{2.19}$$

where $F(\Phi^k) = (f_1(\Phi_k), f_2(\Phi_k), ..., f_n(\Phi_k))$, and $J(\Phi^k)$ is the Jacobian matrix of F evaluated at Φ^k. This process (i.e. application of equation (2.19) followed by equation (2.18)) is continued until $\sum_{i=1}^{n} |\phi_i^{corr}| < \epsilon$, where ϵ is an error tolerance.

An improvement of this method was used by Schreiber and Keller [28] in which the same Jacobian is used for several steps. In many cases only one or two matrix decompositions were required to obtain a solution. The steps for which the previously calculated decomposition is used to obtain a new correction are called *chord iterations*, and since their computational expense is nearly negligible compared to matrix decomposition, they make Newton's method more practical for large systems of equations.

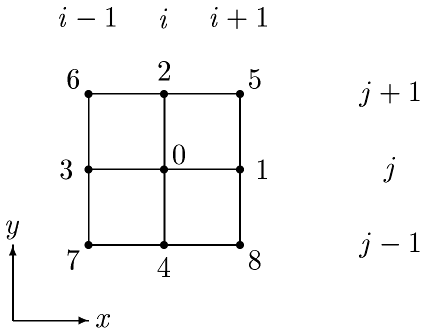

Figure 2.1: Computational stencil

2.3 Numerical Methods

We now discuss numerical methods which have been successfully used to compute flows for the steady, two-dimensional, incompressible Navier-Stokes equations, and their advantages and disadvantages. The notation in Figure 2.1 will be used for schemes with no more than nine points, while the notation in Section 2.2.1 will be used for the remaining schemes.

2.3.1 Second-Order Discerization and Upwind Scheme

Application of second-order accurate central differences to (2.8) and (2.9) yields the schemes

$$\psi_1 + \psi_2 + \psi_3 + \psi_4 - 4\psi_0 = -h^2\zeta_0 \tag{2.20}$$

$$\left(1 - \frac{h}{2}\mathrm{Re}\, u_0\right)\zeta_1 + \left(1 - \frac{h}{2}\mathrm{Re}\, v_0\right)\zeta_2 + \left(1 + \frac{h}{2}\mathrm{Re}\, u_0\right)\zeta_3 + \left(1 + \frac{h}{2}\mathrm{Re}\, v_0\right)\zeta_4 - 4\zeta_0 = 0 \tag{2.21}$$

where $u_0 = (\psi_2 - \psi_4)/2h$ and $v_0 = (\psi_3 - \psi_1)/2h$ are the second-order accurate discretizations of the velocities in the x and y directions respectively. Although this scheme has truncation error $O(h^2\,\mathrm{Re}\,)$, difficulties arise in obtaining the solution of the system. In order to solve the system efficiently it is preferable to use relaxation methods; however, these methods require diagonal dominance of the system for convergence. Using condition (2.12), there is clearly no difficulty for (2.20), but (2.21) gives the following conditions on the grid spacing h

$$h \le \frac{2}{\mathrm{Re}\ |u_0|} \qquad \text{and} \qquad h \le \frac{2}{\mathrm{Re}\ |v_0|} \tag{2.22}$$

which can be very restrictive for large values of Re.

For this reason an *upwind* scheme was developed for (2.9). To illustrate the effect of this scheme we discretize the x-derivatives of ζ as follows

$$\begin{aligned} & \zeta_{xx} - \mathrm{Re}\, u\zeta_x \\ = \; & \frac{1}{h^2}\left[(1-\alpha \mathrm{Re}\, hu_0)\,\zeta_1 - (2-(2\alpha-1)\mathrm{Re}\, hu_0)\,\zeta_0 + (1-(\alpha-1)\mathrm{Re}\, hu_0)\,\zeta_3\right] \\ & +(2\alpha-1)\frac{\mathrm{Re}\, hu_0}{2}\,\zeta_{xx} + O(h^2\,\mathrm{Re}\,) \end{aligned} \tag{2.23}$$

where a linear combination of one-sided derivatives was used for ζ_x, and α is a free parameter. Clearly setting $\alpha = \frac{1}{2}$ gives a central difference scheme as used in (2.21). From this result we see that if $u_0 > 0$, setting $\alpha = 0$ gives the following condition for diagonal dominance

$$|2 + \mathrm{Re}\, hu_0| \geq 1 + |1 + \mathrm{Re}\, hu_0|$$

which is satisfied. For $u_0 < 0$ a similar result can be obtained by setting $\alpha = 1$. The y-derivatives of ζ behave in an analogous way, and the same result follows. This method is called *upwinding* and stated simply it requires that the direction of one-sided discretization of ζ_x and ζ_y be in the opposite direction of the fluid motion.

One major disadvantage of this method is its local truncation error of $O(h\,\mathrm{Re}\,)$, which indicates that a fine grid is required to obtain a reasonably accurate solution. Viewed in another way, this scheme can be seen as a discretization of the equation

$$\left(1+\frac{1}{2}\mathrm{Re}\, h\,|u|\right)\zeta_{xx} + \left(1+\frac{1}{2}\mathrm{Re}\, h\,|v|\right)\zeta_{yy} = \mathrm{Re}\,\left(\psi_y\zeta_x - \psi_x\zeta_y\right)$$

with a local truncation error of $O(h^2\,\mathrm{Re}\,)$. In a region where $|u| \approx |v|$ and denoting the effective Reynolds number Re_{eff} we observe

$$\mathrm{Re}_{eff} \;=\; \frac{\mathrm{Re}}{1+\frac{1}{2}\mathrm{Re}\, h|u_0|},$$

which shows that the Reynolds number is being modified. This departure of the effective Reynolds number from the given Reynolds number is termed *artificial viscosity.* Allowing for no more than 12% reduction in Re the approximate relation

$$h \leq \frac{0.24}{\mathrm{Re}\,|u|}$$

may be obtained indicating a stronger restriction than that for diagonal dominance of the central difference scheme in (2.22).

2.3.2 Dennis and Hudson Scheme

In order to obtain diagonal dominance for the discretization of (2.9) the scheme

$$\begin{aligned} &\left(1-\frac{1}{2}\mathrm{Re}\,hu_0+\frac{\gamma}{16}(\mathrm{Re}\,hu_0)^2\right)\zeta_1+\left(1-\frac{1}{2}\mathrm{Re}\,hv_0+\frac{\gamma}{16}(\mathrm{Re}\,hv_0)^2\right)\zeta_2 \\ &+\left(1+\frac{1}{2}\mathrm{Re}\,hu_0+\frac{\gamma}{16}(\mathrm{Re}\,hu_0)^2\right)\zeta_3+\left(1+\frac{1}{2}\mathrm{Re}\,hv_0+\frac{\gamma}{16}(\mathrm{Re}\,hv_0)^2\right)\zeta_4 \\ &-\left(4+\frac{\gamma}{8}(\mathrm{Re}\,h)^2(u_0^2+v_0^2)\right)\zeta_4 \;=\; 0 \end{aligned} \tag{2.24}$$

with $\gamma = 2$ was derived by Dennis and Hudson in [7]. It can be shown that for this value of γ, each coefficient of ζ_i for $i = 1, \cdots, 4$ is of the form $1 + 2\alpha + 2\alpha^2 = (1 + \alpha)^2 + \alpha^2 > 0$ for suitable choice of α, and the scheme is diagonally dominant for all values of u_0,v_0, and Re. This scheme is consistent with (2.9) with a truncation error of $O(h^2\,\mathrm{Re}^2)$, and has been successfully used for a number of flows(see, e.g. [9, 21, 22]).

One disadvantage of this method becomes apparent for large Re flows. Bramley and Sloan [4] used a modification of the Dennis and Hudson scheme which reduces the error by a constant multiple. They used (2.24) with $\gamma = \frac{4}{3}$ which is also diagonally dominant as the form of the coefficients of ζ_i for $i = 1, \cdots, 4$ is $(1 + \alpha)^2 + \frac{1}{3}\alpha^2 > 0$ for suitable choice of α. Comparison of this method with (2.21) showed the central difference scheme to be at least two orders of magnitude more accurate for Re > 500. The central difference scheme is not diagonally dominant and required Newton's method to solve. It has a local truncation error of $O(h^2\,\mathrm{Re})$, and was only used for comparison purposes due to its computational expense.

Further examination of the modified scheme shows that it can be viewed as a discretization of the equation

$$\left(1+\frac{1}{12}(\mathrm{Re}\,hu)^2\right)\zeta_{xx}+\left(1+\frac{1}{12}(\mathrm{Re}\,hv)^2\right)\zeta_{yy} \;=\; \mathrm{Re}\,\left(\psi_y\zeta_x-\psi_x\zeta_y\right)$$

with a local truncation error of $O(h^2\,\mathrm{Re})$. Following the same line of reasoning as for the upwind scheme one may obtain

$$\mathrm{Re}_{eff} \;=\; \frac{\mathrm{Re}}{1+\frac{1}{12}(\mathrm{Re}\,hu)^2}.$$

Limiting the reduction of Re to no more than 12% gives the approximate relation

$$h \le \frac{1.2}{\mathrm{Re}\,|u|},$$

which is again a stronger restriction than that for diagonal dominance of the central difference scheme in (2.22).

2.3.3 Boundary Conditions

One difficulty with the streamfunction-vorticity formulation involves the application of no-slip boundary conditions to the vorticity equation. There are many techniques available for derivation of these boundary conditions and they are discussed in detail in [26]. One example is the Woods formula [31], given by

$$\zeta_{0,j} \;=\; -\frac{1}{h^2}\left(3\psi_{1,j} + \psi_{0,j+1} + \psi_{0,j-1} - 5\psi_{0,j} - 3h(\psi_x)_{0,j}\right) + \frac{1}{2}\zeta_{1,j}, \tag{2.25}$$

where the subscript i, j is used as a grid coordinate. This formula has been successfully used in many calculations (see, e.g. [9, 22, 21]). Unfortunately, this boundary scheme has been shown to require large amounts of boundary damping in [17], and hence results in slower convergence than other boundary schemes.

An alternative technique has been suggested by Huang and Yang [20], where two boundary conditions are applied on the streamfunction, but the streamfunction-vorticity form of the equations are still used. This eliminates the difficulties for application of boundary conditions on the vorticity, while maintaining a form of the Navier-Stokes equations for which S.O.R. may be used. This technique has been successfully applied in Li et al. [23].

A discussion of convergence rates for high Re cavity calculations using a transformed version of the Navier-Stokes equations may be found in Benjamin and Denny [3]. A detailed comparison of many different boundary schemes can be found in Gupta and Manohar [17], and this work strongly suggests that boundary schemes which give better accuracy are less efficient as they require a larger number of iterations for convergence. These considerations give us motivation to examine pure streamfunction methods.

2.3.4 Pure Streamfunction Difference Formulation

The scheme resulting from application of second-order central differences to (2.10) was studied by Schreiber and Keller in [28]. The major disadvantage of this scheme is its lack of diagonal dominance, thus forcing one to use a direct solver, such as Newton's method, to obtain a solution. In general, compared with iterative methods, direct methods are considered less efficient as they require more memory and more c.p.u time.

When assessing the value of this method, a number of advantages must also be taken into consideration. The vorticity is not present in the discrete formulation which may have advantages for problems in which it is singular (for example, driven cavity problems, and sudden compression problems). Application of boundary conditions has now become identical to that of the biharmonic problem, and straightforward interpolation and extrapolation may be used. The local truncation error for this scheme is $O(h^2\,\mathrm{Re})$ which has advantages over the upwind scheme and the Dennis and Hudson scheme at large Re. And finally since a direct solver is being used, convergence rates due to the boundary schemes need not be considered.

Using this scheme, Reynolds number continuation, and an efficient combination of Newton decompositions and chord iterations, Schreiber and Keller [28] were able to obtain solutions using only a few matrix decompositions at each Re. Reasonable results were obtained for up to Re $= 10000$.

Schreiber and Keller demonstrated in [28] that, for large h, spurious solutions exist for the above method. A convincing argument for the existence of spurious solutions is paraphrased as follows. [A system of N quadratic equations actually has 2^N solutions. Fortunately a great number of these solutions are complex, and cannot be reached with real calculations, but certainly a few exist.] In that paper they outlined a procedure for obtaining spurious solutions using arc-length continuation in Re and ψ_c, where ψ_c is the value of ψ at the center of the primary vortex. This allowed them to study the behavior of the spurious solutions in detail, providing information regarding the limitations and strengths of their method in this area. This procedure may be useful to adapt to other schemes, as it may be used to examine the behavior of their existing spurious solutions.

2.3.5 Fourth-Order Compact Schemes

The approximation

$$\frac{1}{6h^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}\psi = -\frac{1}{12}\begin{bmatrix} & 1 & \\ 1 & 8 & 1 \\ & 1 & \end{bmatrix}\zeta \qquad (2.26)$$

to (2.8) is probably the best known example of the Mehrstellenverfahren [6]. Although the left and right hand sides separately are only second-order accurate approximations if applied to arbitrary functions ψ and ζ, the approximation (2.26) is fourth-order accurate when applied to any solutions to equation (2.8). A number of schemes have been found which are computationally efficient and yield highly accurate numerical solutions [5, 8, 16, 23]. Gupta et. al [18], Dennis and Hudson [8], Gupta [16], and Li et. al [23] note that this technique can be generalized to provide a fourth-order accurate (9-point) compact scheme to (2.9). Each of these schemes has a local truncation error of $O(h^4\,\mathrm{Re}^2)$, and although there is no rigorous proof for diagonal dominance, S.O.R. can be used for reasonable Re. For high Re direct solution methods may be necessary. All schemes, with the exception of Li et al. require values of ψ_i which lie outside the 3×3 stencil, and all are lengthy so they will not be presented here. One key point to mention is that the the error term for each of these schemes indicates that reasonable results may be obtained for low to mid Reynolds number flows, but for higher Re the results may not be very accurate.

Chapter 3

Wide and Compact Scheme

3.1 Difference Formulations

In this section we derive the compact and wide schemes for the steady–state streamfunction equation. We do this using well known difference formulae (2.11) and a method which may be easily adapted to other elliptic equations. Derivations involving Maple, a symbolic programming language, are also presented.

3.1.1 Wide Scheme

Using the central difference approximations of the previous section we discretize (2.10) directly to fourth-order accuracy:

$$\begin{aligned}[NS] \;=\;& \tilde{D}_{xxxx}\psi + 2\tilde{D}_{xxyy}\psi + \tilde{D}_{yyyy}\psi \\ & + \mathrm{Re}\,\Big(\tilde{D}_x\psi\;(\tilde{D}_{xxy}\psi + \tilde{D}_{yyy}\psi) \;-\; \tilde{D}_y\psi\;(\tilde{D}_{xxx}\psi + \tilde{D}_{xyy}\psi)\Big) \;=\; 0. \end{aligned} \tag{3.1}$$

This gives a fourth-order scheme for the streamfunction equation. The computational stencil for this equation looks like:

$$[NS] = \begin{bmatrix} & & & \gamma & & & \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ \beta & x & x & x & x & x & \alpha \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ & & & \delta & & & \end{bmatrix} \psi = O(\mathrm{Re}\ \ h^4) \tag{3.2}$$

with

$$\alpha = -\frac{1}{6h^4} + \frac{\mathrm{Re}\,\tilde{D}_y\psi}{8h^3}, \qquad \beta = -\frac{1}{6h^4} - \frac{\mathrm{Re}\,\tilde{D}_y\psi}{8h^3},$$
$$\gamma = -\frac{1}{6h^4} - \frac{\mathrm{Re}\,\tilde{D}_x\psi}{8h^3}, \qquad \delta = -\frac{1}{6h^4} + \frac{\mathrm{Re}\,\tilde{D}_x\psi}{8h^3}.$$

This method gives a 29-point stencil for calculation of the streamfunction. This may have disadvantages for the solution of the discrete system of equations when the scheme is applied at each point in the interior. It raises the question of how to deal with points near the boundary, and its width gives us motivation to derive a compact scheme as we do in the next subsection.

3.1.2 Compact Scheme

In order to derive the compact scheme on a 5×5 computational stencil we need to eliminate the outliers (i.e. α, β, γ, and δ) using derivatives of the streamfunction equation. The analytic derivation of this subsection is from [12].

We will use the following difference approximations for which all derivatives higher than second-order are approximated by second-order differences, and all others by fourth-order differences:

$$\begin{aligned}[NS_x] \ = \ & D_x\Delta^h\Delta^h\psi \ + \ \mathrm{Re}\,(\tilde{D}_x\psi\ D_{xy}\Delta^h\psi \ - \ \tilde{D}_y\psi\ D_{xx}\Delta^h\psi \\ & + \tilde{D}_{xx}\psi\ D_y\Delta^h\psi \ - \ \tilde{D}_{xy}\psi\ D_x\Delta^h\psi),\end{aligned} \tag{3.3}$$

$$[NS_x] = \left[\begin{array}{ccccccc} & & & & & & \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ \zeta & x & x & x & x & x & \epsilon \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ & & & & & & \end{array} \right] \psi = O(\mathrm{Re}\ \ h^2), \tag{3.4}$$

with

$$\epsilon = \frac{1}{2h^5}, \qquad \zeta = -\frac{1}{2h^5};$$

$$\begin{aligned} [NS_y] \; = \; & D_y \Delta^h \Delta^h \psi \;\; + \;\; \mathrm{Re}\,(\tilde{D}_x \psi \; D_{yy} \Delta^h \psi \; - \; \tilde{D}_y \psi \; D_{xy} \Delta^h \psi \\ & + \tilde{D}_{xy} \psi \; D_y \Delta^h \psi \; - \; \tilde{D}_{yy} \psi \; D_x \Delta^h \psi), \end{aligned} \tag{3.5}$$

$$[NS_y] = \left[\begin{array}{ccccc} & & \eta & & \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ x & x & x & x & x \\ & & \theta & & \end{array} \right] \psi = O(\mathrm{Re}\ \ h^2), \tag{3.6}$$

with

$$\eta = \frac{1}{2h^5}, \qquad \theta = -\frac{1}{2h^5};$$

$$\begin{aligned} [\Delta NS] \; = \; & \Delta^h \Delta^h \Delta^h \psi \;\; + \;\; \mathrm{Re}\,(\tilde{D}_x \psi \; D_y \Delta^h \Delta^h \psi \; - \; \tilde{D}_y \psi \; D_x \Delta^h \Delta^h \psi \\ & + 2\tilde{D}_{xx} \psi \; D_{xy} \Delta^h \psi \; - \; 2\tilde{D}_{xy} \psi \; D_{xx} \Delta^h \psi \\ & + 2\tilde{D}_{xy} \psi \; D_{yy} \Delta^h \psi \; - \; 2\tilde{D}_{yy} \psi \; D_{xy} \Delta^h \psi), \end{aligned} \tag{3.7}$$

$$[\Delta NS] = \begin{bmatrix} & & & \lambda & & & \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ \kappa & x & x & x & x & x & \sigma \\ & x & x & x & x & x & \\ & x & x & x & x & x & \\ & & & \mu & & & \end{bmatrix} \psi = O(\mathrm{Re}\ h^2), \tag{3.8}$$

with

$$\sigma = \frac{1}{h^6} - \frac{\mathrm{Re}\,\tilde{D}_y\psi}{2h^5}, \qquad \kappa = \frac{1}{h^6} + \frac{\mathrm{Re}\,\tilde{D}_y\psi}{2h^5},$$
$$\lambda = \frac{1}{h^6} + \frac{\mathrm{Re}\,\tilde{D}_x\psi}{2h^5}, \qquad \mu = \frac{1}{h^6} - \frac{\mathrm{Re}\,\tilde{D}_x\psi}{2h^5};$$

Combining these difference schemes in such a way that the outliers are eliminated gives the equation

$$[NS] + h^2(\ [\Delta NS]/6 + \mathrm{Re}\,(\tilde{D}_x[NS_y] - \tilde{D}_y[NS_x])/12\ ) = O(\mathrm{Re}^2\ h^4), \tag{3.9}$$

which forms a fourth-order approximation to (2.10) on a 5×5 square of grid points.

Although the symbolic derivation methods presented in the next subsection can be very useful, an $O(\mathrm{Re}\ h^4)$ scheme extending over the 5×5 grid of points used above could not be found. A number of forms for this difference approximation were used, and this result strongly suggests that the compact scheme given above has the lowest Re power for the given grid.

3.1.3 Symbolic Scheme Derivation

The schemes derived in Sections 3.1.1 and 3.1.2 may be derived using a symbolic language such as Maple. There are many techniques available to obtain these schemes, two of which will be presented here.

The first technique is most useful in obtaining the highest order possible for a scheme using the coordinates of the grid points to be used, and the user allowed form of the scheme. As an example we look at equation (2.8). We define E by

$$E \;=\; \Delta\psi + \zeta \;=\; \psi_{xx} + \psi_{yy} + \zeta,$$

and form the difference approximation

$$\tilde{E} = \frac{1}{h^2}\sum_{i=0}^{n-1} b_i\psi_i + \sum_{i=0}^{n-1} c_i\zeta_i$$

using the n allowed grid points. We now take the difference between the method and the equation,

$$R = \tilde{E} - E - \sum_{m=1}^{r}\sum_{i=0}^{m} h^m a_{m-i,i}\frac{\partial^m E}{\partial^{m-i}x\,\partial^i y}$$

taking all differential consequences (derivatives of E) into account up to r, the predicted order of accuracy. Efficient substitution of appropriate order Taylor series into R gives a large expression involving the derivatives of ψ and ζ, powers of h, and the variables $a_{i,j}$, b_i, and c_i. Requiring that all terms vanish to the desired order of accuracy yields a set of linear equations in $a_{i,j}$, b_i, and c_i which must be solved to obtain the desired difference approximation. If this system is consistent then the difference approximation is given by $\tilde{E}$, where all remaining variables may be treated as free parameters.

Application of this technique to our example equation using the standard 9-point stencil gives us the fourth-order accurate approximation

$$\frac{1}{6h^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}\psi = -\frac{1}{12}\begin{bmatrix} \alpha & 1-2\alpha & \alpha \\ 1-2\alpha & 8+4\alpha & 1-2\alpha \\ \alpha & 1-2\alpha & \alpha \end{bmatrix}\zeta \tag{3.10}$$

where α is a free parameter. The code and relevant calls are located in Appendix 2. One restriction on the parameter α is that it must be at least $O(1)$, as if it were $O(\frac{1}{h})$, the scheme (3.10) would be only third-order accurate.

Dealing with non-linear equations, such as (2.10), is slightly more difficult. In this case we use the approximation

$$\tilde{E} = \frac{1}{h^4}\left[\sum_{i=0}^{n-1} b_i\psi_i + \mathrm{Re}\sum_{i=0}^{n-1}\sum_{j=0}^{i} c_{i,j}\psi_i\psi_j\right]$$

which has a much greater number of unknowns, and the Taylor series approximations must be done carefully to prevent expression explosion. When the difference is taken, and the linear equations are solved to fourth-order accuracy, it is found that there remain 280 free parameters. This allows for a great many schemes, and finding one which can be coded reasonably may be difficult. The code and commands for this derivation may be found in Appendix 3.

For the compact scheme, up to cubic terms may be present, hence the approximation

$$\tilde{E} = \frac{1}{h^4}\left[\sum_{i=0}^{n-1} b_i\psi_i + \mathrm{Re}\sum_{i=0}^{n-1}\sum_{j=0}^{i} c_{i,j}\psi_i\psi_j + \mathrm{Re}^2\sum_{i=0}^{n-1}\sum_{j=0}^{i}\sum_{k=0}^{j} d_{i,j,k}\psi_i\psi_j\psi_k\right]$$

must be used, and including all differential consequences can be quite difficult. These considerations motivate us to find a different method of scheme derivation for non-linear equations.

The second technique uses known difference formulae to obtain a scheme, and closely follows the analytic derivation used in Section 3.1.2. The resulting code is less procedural, requiring substantially more user input, for example which differential consequences to be used, and the way in which they are used must be supplied by the user. Each derivative is approximated by a linear combination of known differences. As an example we represent the derivative ψ_{xx} as

$$\psi_{xx} = (1-\alpha-\beta-\gamma)\tilde{D}_{xx}\psi + \alpha D_{xx}\psi + \beta\tilde{D}_x\tilde{D}_x\psi + \gamma D_x D_x\psi$$

where the notation of Section 2.2.1 is used for the differences. This process is performed for each derivative required by the method, including those derivatives required by the differential consequences, and the resulting parameters may be used to obtain some desired property for the scheme, such as elimination of certain outliers and/or diagonal dominance.

A sample code using no free parameters is given in Appendix 4. In addition to deriving the Wide scheme in Section 3.1.1, this code produces Fortran expressions used to calculate the difference approximations, and the Jacobian elements required for use of the scheme. This *translation* code is scheme dependent and must be significantly altered for use in derivation of other schemes. One other detail is that all parameters must be fully determined and substituted into the appropriate expressions before the Fortran code can be written.

To demonstrate the potential value of this technique a 33-point scheme having a width of 7 was obtained having a local truncation error of $O(h^6\,\mathrm{Re})$.

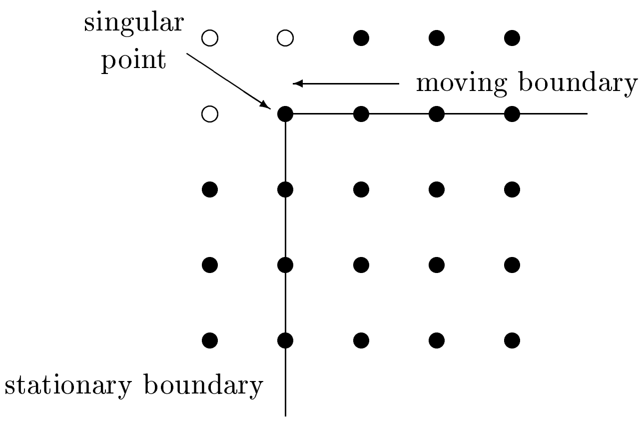

Figure 3.1: Compact Method with Corner Singularity

3.1.4 Rectangular-Domain Boundary Conditions

Application of normal derivative boundary conditions for rectangular domain problems may be done in a number of ways, but in the case of a driven cavity problem care must be taken in the corners as singularities may be present (for an analytic description of these singularities see [24]). Straightforward application of either scheme at each point in the domain results in a system for which finite differences are taken across singularities with no obvious way of determining the exterior corner values (see Figure 3.1). Using interpolation for the first interior points along the boundary avoids this difficulty as the exterior corner values are no longer required for either scheme. We may use any of the following relations for this purpose:

$$\psi_x = (-3\psi_0 + 4\psi_1 - \psi_2)/2h + O(h^2) \tag{3.11}$$

$$\psi_x = (-11\psi_0 + 18\psi_1 - 9\psi_2 + 2\psi_3)/6h + O(h^3) \tag{3.12}$$

$$\psi_x = (-25\psi_0 + 48\psi_1 - 36\psi_2 + 16\psi_3 - 3\psi_4)/12h + O(h^4) \tag{3.13}$$

$$\psi_x = (-137\psi_0 + 300\psi_1 - 300\psi_2 + 200\psi_3 - 75\psi_4 + 12\psi_5)/60h + O(h^5) \tag{3.14}$$

The exterior points required for the wide scheme may be determined by extrapolation using any of the following relations:

$$\psi_x = (-\psi_{-1} + \psi_1)/2h + O(h^2) \tag{3.15}$$

$$\psi_x = (-2\psi_{-1} - 3\psi_0 + 6\psi_1 - \psi_2)/6h + O(h^3) \tag{3.16}$$

$$\psi_x = (-3\psi_{-1} - 10\psi_0 + 18\psi_1 - 6\psi_2 + \psi_3)/12h + O(h^4) \tag{3.17}$$

$$\psi_x = (-12\psi_{-1} - 65\psi_0 + 120\psi_1 - 60\psi_2 + 20\psi_3 - 3\psi_4)/60h + O(h^5) \tag{3.18}$$

Here ψ_i denotes the function value at the i-th interior point from the boundary, ψ_{-1} denotes the first exterior point from the boundary, and h is the step–size.

There is some ambiguity in the evaluation of the interior corner points as there are two boundary conditions to be applied at each. To obtain a relation for these points we take the two boundary conditions, namely $\psi_c = f_1(\psi_{i,j})$ and $\psi_c = f_2(\psi_{i,j})$ and minimize $(\psi_c - f_1(\psi_{i,j}))^2 + (\psi_c - f_2(\psi_{i,j}))^2$ with respect to ψ_c. The resulting relation is then simply

$$\psi_c = \frac{1}{2}\left(f_1(\psi_{i,j}) + f_2(\psi_{i,j})\right)$$

The combination of the above boundary conditions with the wide and compact schemes determines a unique relation for each point in the rectangular domain.

3.2 Model Problems and Computational Methods

In this section we describe the model problems used to test the wide and compact schemes, three of these with exact solutions, and the shear-driven cavity problem. We also discuss the computational methods used to obtain solutions of the resulting non-linear system of equations, and the combinations of schemes and boundary-conditions to be used.

3.2.1 Exact Model Problems

In order to test the accuracy of our schemes we use the following exact solutions to the streamfunction equation:

Problem 1

$$\psi(x,y) = a\ (x^2+y^2) + b\ (x^2+y^2)\ \ln(x^2+y^2) \tag{3.19}$$
$$(x,y) \in [1,2]\times[1,2] \qquad a,b \in R$$

Problem 2

$$\psi(x,y) = a\ (x^2+y^2) + b\ (x^2+y^2)\ \ln(x^2+y^2) \tag{3.20}$$
$$(x,y) \in [0,1]\times[0,1] \qquad a,b \in R$$

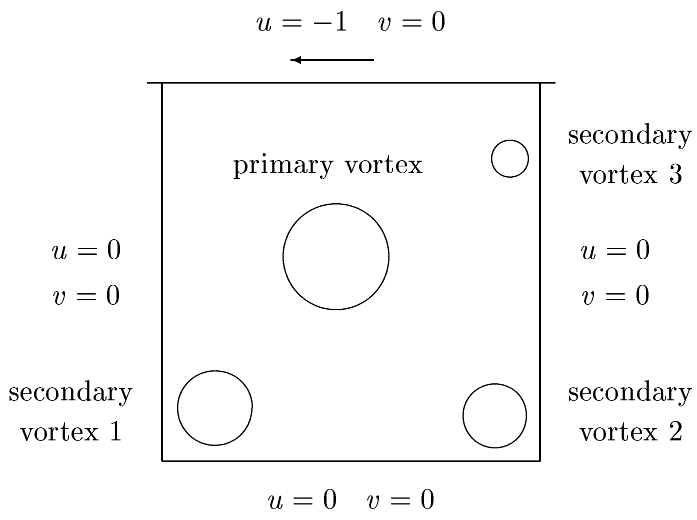

Figure 3.2: Driven cavity

Problem 3

$$\psi(x,y) = \frac{a\ (y-x)}{\mathrm{Re}} - b\ (e^{a(x+y)} - e^{a}) \tag{3.21}$$

$$(x,y) \in [0,1] \times [0,1] \qquad a,b \in R$$

These are straightforward generalizations of the test problems introduced by Richards and Crane [27]. In Problems 1 and 2 we choose $a = \frac{1}{2}$ and $b = -\frac{1}{4}$, here we note that Problem 2 possesses a weak singularity at the point (0,0) which may give some insight as to the behavior of the schemes when a corner point singularity is present. For Problem 3 we choose $a = \frac{4}{5}$ and $b = \frac{1}{5}$ yielding a solution for which higher derivatives decrease in value geometrically.

3.2.2 Driven Cavity

As a practical model problem, we consider the steady flow of an incompressible viscous fluid in a square cavity ($0 \le x \le 1, 0 \le y \le 1$). The flow is induced by the sliding motion of the top wall ($y = 1$) from right to left (see Figure 3.2) and is described by the streamfunction equation. The boundary conditions are those of no slip: on the stationary walls $u = \partial\psi/\partial y = 0$ and $v = -\partial\psi/\partial x = 0$; on the sliding wall $u = -1$ and $v = 0$. This problem has been used frequently as a model problem for testing and evaluating numerical techniques, in spite of the singularities at two of its corners. Highly accurate benchmark solutions of this problem are available in the literature (see, e.g. [13, 14, 28]).

3.2.3 Computational Methods

To solve these non-linear problems we apply Newton's method to the equations resulting from the finite difference approximations in Section 3.1. The sequence of linear systems is then solved using a banded LU-decomposition routine with partial pivoting. For efficiency we use the algorithm suggested by Schreiber and Keller in [28], performing several chord-iterations per decomposition, only re-evaluating and decomposing the Jacobian when chord-iteration convergence becomes slow. In the course of the calculations it is found that at most two decompositions are required for each continuation calculation.

For the driven cavity problem, Reynolds number continuation is used to obtain the coarse grid solutions, and a bi-quadratic interpolation routine is used to obtain a first estimate for finer grids. Calculations are performed for Reynolds numbers 100, 400, 1000, 3200, and 5000 requiring two continuations between each. The calculation at Reynolds number 100 is done with first approximation $\psi_{i,j} = 0$.

Three combinations of boundary conditions and schemes are used in all calculations; these are as follows:

W_4: Wide scheme with fourth-order accurate interpolation (3.13) and extrapolation (3.17)

W_5: Wide scheme with fifth-order accurate interpolation (3.14) and extrapolation (3.18)

C_4: Compact scheme with fourth-order accurate interpolation (3.13)

For the wide scheme we may use up to fifth-order accurate boundary conditions without significantly changing the bandwidth of the Jacobian, while the compact scheme is restricted to fourth-order or less to maintain compactness. The banded structure of the Jacobian matrix for W_4 is illustrated in Figure 3.3.

For the exact model problems three additional methods are used:

W_b: Wide scheme with exact boundary conditions

C_b: Compact scheme with exact boundary conditions

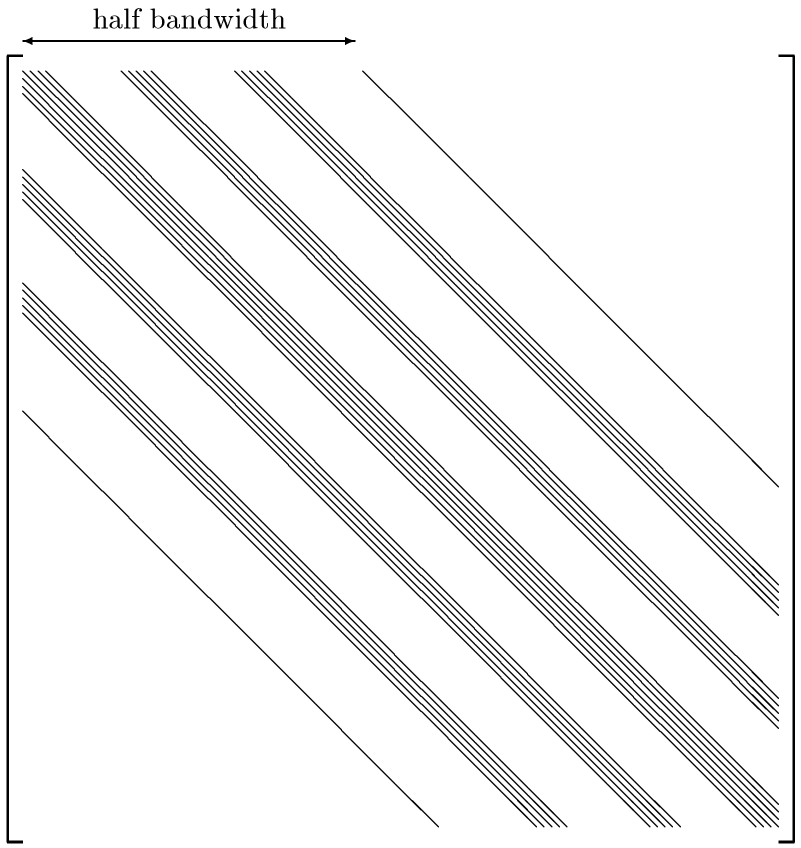

Figure 3.3: Jacobian Band Structure for W_4

S_2: Second-order scheme with second-order accurate extrapolation (3.15)

The first two schemes use exact boundary conditions fixing the points determined by approximate boundary conditions in the original three schemes to exact solution values. These methods are useful for comparison of the error resulting from the schemes to the error resulting from the boundary conditions. The remaining scheme is the second-order accurate scheme used by Schreiber and Keller in [28]. This scheme uses second-order accurate centered differences to discretize (2.10), applying it at all interior points, and second-order accurate extrapolation for the boundary conditions.

Two other schemes were originally used, and these were the wide scheme with sixth-order accurate boundary conditions, and the second-order scheme with third-order accurate boundary conditions. It was found that in using Newton's method and LU-decomposition with partial pivoting, Newton iteration only converged for very low Re. It is suspected that the higher order boundary conditions caused the resulting Jacobian matrix to have a large condition number, requiring full pivoting to perform the decomposition. An exception to this behavior is W_5 where partial pivoting had no adverse effect on the convergence of the Newton iterations. Since the expense of full pivoting is prohibitive, the schemes were not used.

3.3 Results for Exact Model Problems

In this section four sets of results are presented for the exact model problems using the methods outlined in the previous section. These calculations were performed for grid spacings from $\frac{1}{32}$ to $\frac{1}{64}$ on a SPARC 10. First the schemes which may be implemented on practical problems are compared and some conclusions are reached, next the effects of the approximate boundary conditions are discussed, and finally efficiency results are presented and the methods are discussed with all results in mind.

3.3.1 Accuracy of Schemes

For Problem 1, L^2-error calculations obtained for Re $= 100$ are displayed in Figure 3.4. The smallest error (by a factor of 10) was obtained using W_5, followed by C_4, W_4, and S_2. For h$=\frac{1}{64}$, S_2 has three orders of magnitude greater error than each of the fourth-order accurate schemes. It is also worth noting that increasing the accuracy of the boundary conditions from $O(h^4)$ to $O(h^5)$ (i.e. W_4, and W_5) reduced the error by approximately $\frac{1}{30}$ in the region plotted.

Increasing Re from 100 to 5000 shows an increase in the error for C_4 by a factor greater than 15, while the errors for W_4, W_5, and S_2 changed very little, which is consistent with each scheme's respective truncation error.

For Problem 2, results are presented for Re $= 100$ and Re $= 5000$ in Figure 3.5. Although the wide and compact methods have overall fourth-order truncation error, the results obtained have only third-order accurate behavior. Comparison of these results to Problem 1 leads us to the conclusion that this loss of order is a result of the weak singularity present at the origin. For Re $= 100$, C_4 displays the smallest error, and S_2 the largest. All four practical schemes remain within an order of magnitude of each other in the region plotted, indicating that for Re $= 100$ efficiency may be the larger consideration. Comparison of the results for Re $= 100$ to Re $= 5000$ again indicates a larger increase in the error for C_4 than for W_4, W_5, and S_2. At Re $= 5000$ W_5 has the smallest error being approximately $\frac{1}{5}$ that of C_4, and again the least accurate scheme is S_2.

Results for Problem 3 are nearly identical to those for Problem 1 and are not presented here.

Using the information presented in this subsection we can obtain some conclusions about the relative accuracy of the practical schemes. The largest error in all cases was obtained by S_2, even for the relatively coarse meshes used for these results. Refinement of the mesh will increase the accuracy of the other schemes considerably faster than for S_2, so in this respect the fourth-order schemes are always more accurate than S_2. For smooth problems (i.e. Problems 1 and 3) and low Re, W_5 is slightly more accurate than C_4, but when the singularity was present, C_4 gave more accurate results. However for higher Re, C_4 has a much larger error, and the most accurate method is W_5. In all cases W_5 produced less error than W_4 indicating that this higher order boundary condition only seeks to reduce the error, and may be beneficial to use whenever possible.

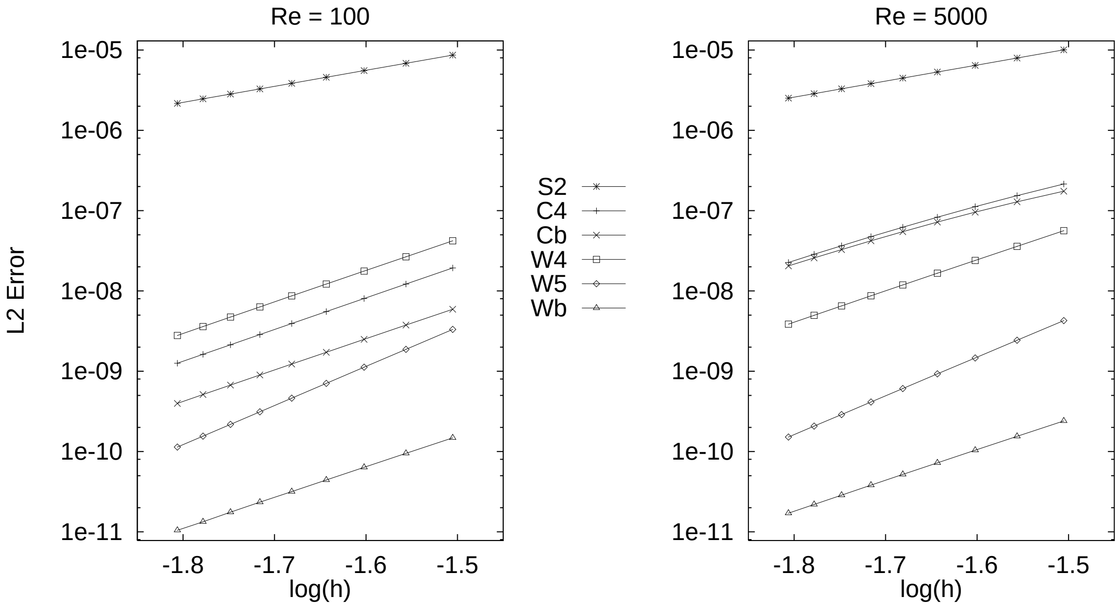

Figure 3.4: L2 Errors for Test Problem 1

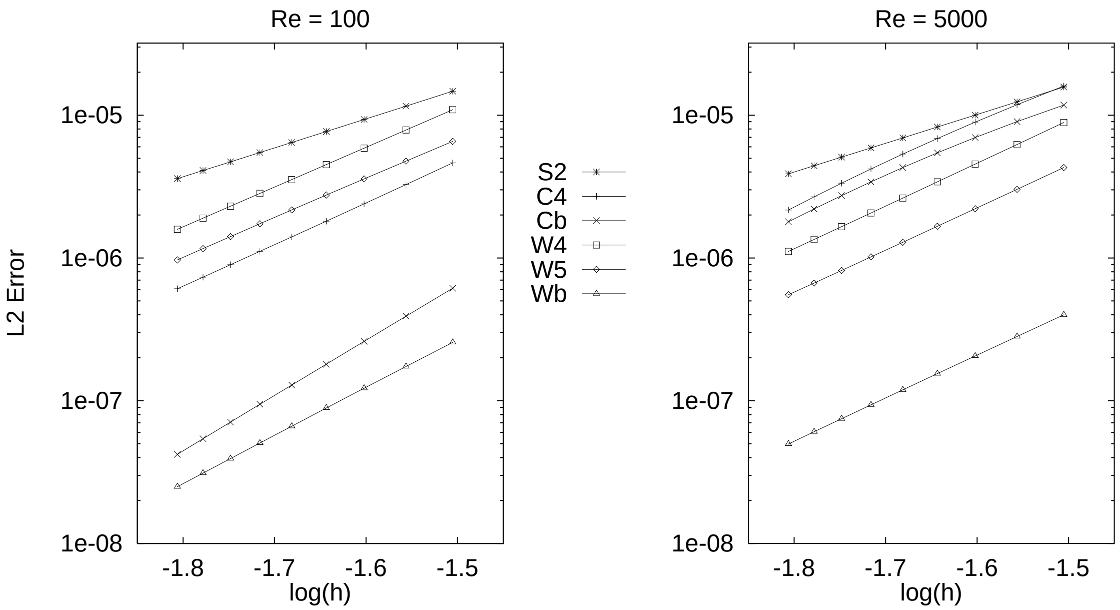

Figure 3.5: L2 Errors for Test Problem 2

3.3.2 Effects of Numerical Boundary Conditions

Comparing the schemes using exact boundary conditions (i.e. W_b and C_b) to those using approximate boundary conditions (i.e. W_4, W_5 and C_4) in Figures 3.4 and 3.5 shows that the approximate boundary conditions generate a great deal of error. For the compact schemes and Re $= 5000$, the errors for C_4 and C_b are nearly equal, indicating that the largest contribution to the error is due to the scheme; but for the wide schemes, it may be seen that the methods using approximate boundary conditions produce an error from 10 to 500 times as large as those using exact boundary conditions. We note here that this effect is sometimes observed when row-scaling is not used on the Jacobian matrix. This is not the case as the dependence of the scheme points on the boundary points is eliminated before decomposition of the matrix. The large error resulting from the boundary conditions becomes significant when one considers that $O(n^2)$ points are determined by the scheme while only $O(n)$ points are determined by the boundary conditions. This observation illustrates the potential importance of boundary conditions relative to the schemes. To further emphasize this point, all results display the expected order of accuracy with the exception of W_5 for Problems 1 and 3, where it appears to be an $O(h^5)$ method. A reasonable explanation for this behavior is that the coefficient of the fifth-order error term corresponding to the boundary conditions is much larger than the coefficient of the fourth-order error term for the scheme. The actual fourth-order behavior of the method would then not become apparent until h was very small. In light of this result we conclude that for the wide scheme, any improvement in the accuracy of the boundary conditions will give a corresponding improvement in the global accuracy of the computation. For the compact scheme this effect will only be noticed for low Re .

3.3.3 Efficiency of Methods

In using Newton's method, the most expensive step is the LU-decomposition of the Jacobian, requiring $O(n^4)$ operations, where n is the number of grid points along the side of the square domain. The C.P.U. time required by this step is primarily dependent upon n, and the bandwidth of the Jacobian, so the decomposition should require the same time for W_4,W_5, and W_b. In Figure 3.6, C.P.U. times are plotted against n for S_2, C_b, and W_b, and verify the fourth-order dependence of the C.P.U. time on n. The times as a function of n for

S_2 and C_b are nearly equal, as their bandwidths are $4n - 3$ and $4n + 1$ respectively, and only differ by a constant quantity. The wide scheme, W_b, with a bandwidth of $6n - 5$ takes approximately 2.5 times the C_b C.P.U. time for a given n. Combining this information with that of the two previous subsections tells us that it is efficient to use C_4 when Re is small, but for larger Re, W_5 is the best method, making up for any inefficiency by a significant increase in accuracy.

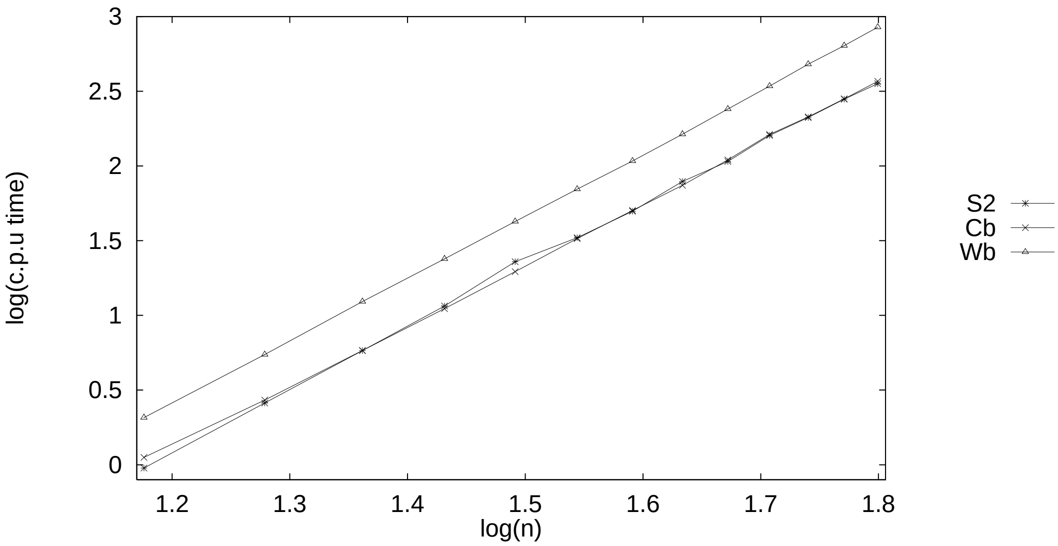

Figure 3.6: C.P.U times for S_2,C_b,and W_b

3.4 Results for Driven Cavity Problem

For the driven cavity problem results are presented for Re = 1000, 3200, and 5000. Termination of Newton's method is achieved when the L_2 residual drops below 10^{-10}. The computed results are compared with those of Ghia et al. [13], Schreiber and Keller [28], and Nishida and Satofuka [25].

Table 3.1: Vortex data for Re =1000

source	primary vortex location	secondary vortex 1 location	secondary vortex 2 location
W_5 65×65	0.11698 0.4690,0.5651	-1.7579(-3) 0.1386,0.1103	-2.2276(-4) 0.9178,0.0789
W_4 65×65	0.11753 0.4682,0.5638	-1.7113(-3) 0.1351,0.1073	-2.3260(-4) 0.9137,0.0797
C_4 65×65	0.11672 0.4697,0.5678	-1.7424(-3) 0.1410,0.1123	-2.0886(-4) 0.9184,0.0783
Ghia *et al* 129×129	0.11816 0.4695,0.5658	-1.7510(-3) 0.1406,0.1094	-2.3113(-4) 0.9141,0.0781
Schreiber *et al* 141×141	0.11603 0.4714,0.5643	-1.7000(-3) 0.1357,0.1071	-2.1700(-4) 0.9143,0.0714
Nishida *et al* 129×129	0.11882 0.4687,0.5625	-1.7238(-3) 0.1406,0.1094	-2.3153(-4) 0.9141,0.0781

In Table 1 we present the vortex data for Re =1000 for W_4,W_5, and C_4 using 65×65 grids. The results compare well with the known benchmark solutions, with the primary vortex and its co-ordinates for these methods being within 2% of the benchmark values. The streamlines for W_5 and C_4 are presented in Figure 3.7, and it is observed that they are graphically indistinguishable. Other calculations using 97×97 grids indicate a 0.2% deviation from each other.

Table 3.2: Vortex data for Re =3200

source	primary vortex location	secondary vortex 1 location	secondary vortex 2 location	secondary vortex 3 location
W_5 97×97	0.11870 0.4819,0.5408	-2.8906(-3) 0.1783,0.0848	-1.0739(-3) 0.9166,0.1165	-6.3025(-4) 0.9493,0.8927
W_4 97×97	0.11980 0.4805,0.5394	-2.7837(-3) 0.1714,0.0858	-1.1277(-3) 0.9153,0.1190	-6.1421(-4) 0.9499,0.8924
C_4 97×97	0.11793 0.4833,0.5456	-2.8750(-3) 0.1892,0.0863	-9.3701(-4) 0.9176,0.1142	-5.1456(-4) 0.9520,0.8898
Ghia *et al* 129×129	0.12038 0.4835,0.5496	-3.1396(-3) 0.1875,0.0859	-9.7823(-4) 0.9141,0.1094	-7.2768(-4) 0.9453,0.8984
Nishida *et al* 129×129	0.12072 0.4844,0.5391	-2.7897(-3) 0.1719,0.0859	-1.0859(-4) 0.9219,0.1250	-7.2887(-4) 0.9453,0.1016

For Re =3200, 97×97 grids were used to obtain the data in Table 2. In this case the W_4 and W_5 results are within 1% of the benchmark solutions , while the C_4 results have slightly larger deviation. This is expected as for high Re the compact scheme was shown to have greater error for the test problems. The streamlines for W_5 and C_4 are shown in Figure 3.8. It may be observed that the general shape of the two contour plots differ little, but the primary difference is present in the value of ψ at the vortices.

Table 3.3: Vortex data for Re =5000

source	primary vortex location	secondary vortex 1 location	secondary vortex 2 location	secondary vortex 3 location
W_5 97×97	0.11799 0.4838,0.5350	-3.1365(-3) 0.1900,0.0738	-1.3587(-3) 0.9257,0.1372	-1.2655(-3) 0.9406,0.9023
W_4 97×97	0.12053 0.4810,0.5327	-2.8381(-3) 0.1744,0.0701	-1.4770(-3) 0.9284,0.1428	-1.2292(-3) 0.9413,0.9023
C_4 97×97	0.11531 0.4870,0.5487	-3.1914(-3) 0.2234,0.0842	-9.3084(-4) 0.9224,0.1296	-9.3058(-4) 0.9447,0.8969
Ghia *et al* 257×257	0.11897 0.4883,0.5352	-3.0836(-3) 0.1914,0.0742	-1.3612(-3) 0.9297,0.1367	-1.4564(-3) 0.9375,0.9102

For Re =5000, 97×97 grids were used to obtain the data in Table 3. The wide schemes were calculated using continuation on 65×65 grids, with continuation steps at Re =3200,4000, and 5000. A solution for the compact scheme could not be obtained past Re =4600 on this coarse mesh, and the continuation behavior, in combination with the work of Schreiber and Keller in [29], leads us to suspect a spurious bifurcation is present. Refinement eliminated this difficulty allowing us to obtain a 97×97 solution. The primary vortex data for the wide schemes is within 2% of the benchmark data, while the compact scheme is slightly less accurate. Reasonably good agreement with Ghia's results using less than $\frac{1}{6}$ the grid points indicates the value of these schemes. The streamlines for W_5 and C_4 with 97×97 grids are shown in Figure 3.9. There is a noticeable difference in the levels of the streamlines for the primary vortex as well as the positions of the secondary vortices. Secondary vortex 1 is too large for C_4, but is much closer to the benchmark solution for W_5. Secondary vortex 2 and 3 are too small for the compact scheme, and again the wide scheme is closer to the benchmark streamlines.

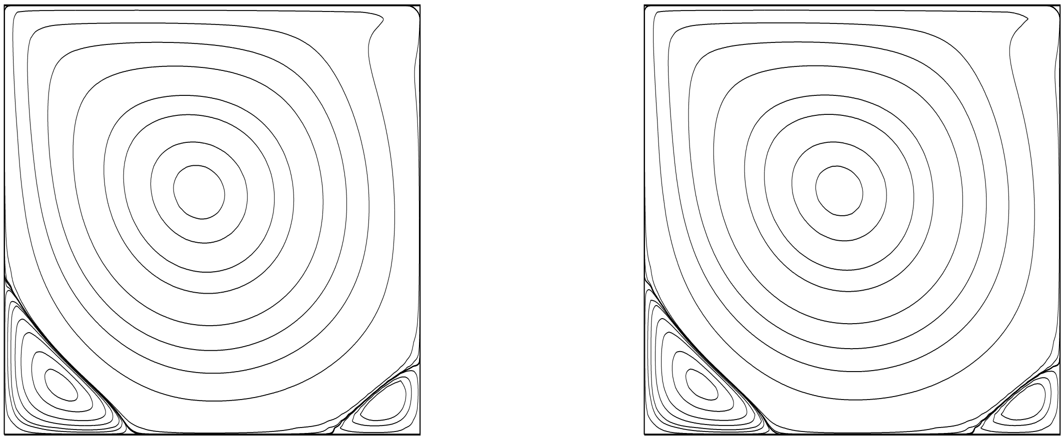

Figure 3.7: Streamlines for W_5(left) and C_4(right) at Re $= 1000$

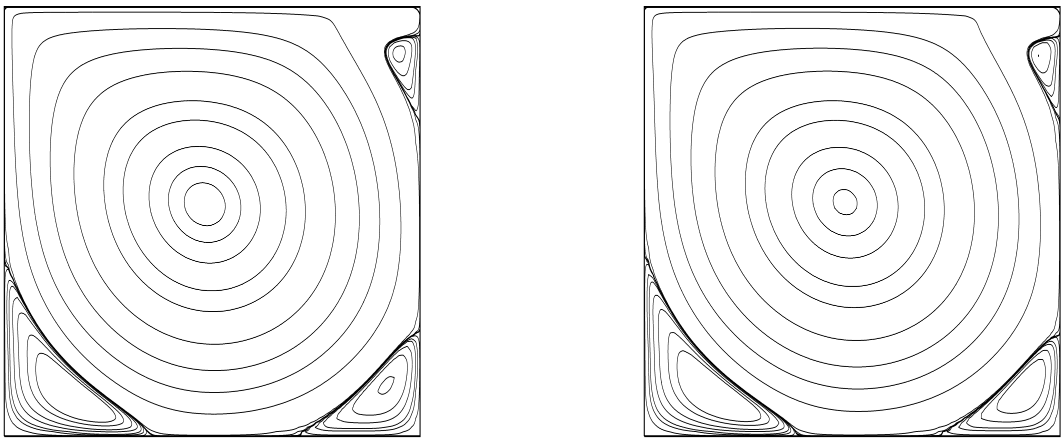

Figure 3.8: Streamlines for W_5(left) and C_4(right) at Re $= 3200$

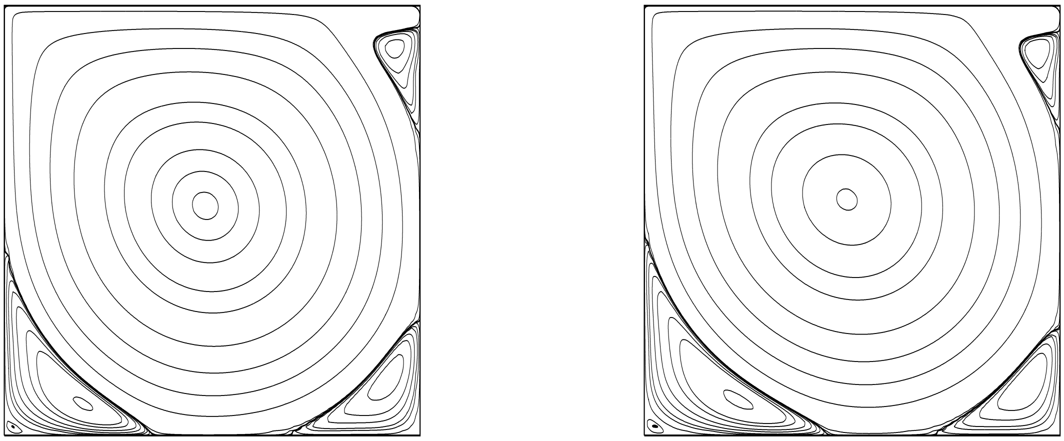

Figure 3.9: Streamlines for W_5(left) and C_4(right) at Re $= 5000$

Chapter 4

Conclusions

In this thesis a fourth-order wide scheme and a fourth-order compact scheme for the two-dimensional steady state Navier-Stokes equations were derived using both analytic techniques and the symbolic programming language Maple. Although these methods require direct solution techniques, which are considered inefficient, they have a number of factors working in their favor. Application of boundary conditions is greatly simplified, as we are only dealing with the streamfunction ψ. This may also have advantages when working with problems for which the vorticity is singular. We expect good agreement with the benchmark solutions for the schemes, as the truncation errors are fourth order, and the Reynolds number dependence of the truncation errors may also provide some insight for general wide and compact schemes.

Our exact problem calculations show that in some cases with low Re, the compact scheme has less error than the wide scheme, but for high Re the wide scheme is orders of magnitude better. The driven cavity calculations support this result, as the wide scheme gives good agreement with the benchmark solutions for all Re tested, while the compact scheme agrees only for lower Re. This behavior may be easily predicted from the form of the truncation errors, being $O(h^4\,\mathrm{Re})$ for the wide scheme, and $O(h^4\,\mathrm{Re}^2)$ for the compact scheme. Both schemes compare well with their second-order counterpart on the exact problems, having notably less error for even relatively large h.

Boundary conditions were found to have a significant impact on the error of the approximate solutions. Using fifth-order boundary conditions with a fourth-order scheme (i.e. W_5)

clearly demonstrated *fifth-order* error. Comparison of the solutions using exact values in the place of boundary conditions to those using approximate boundary conditions verified this, in some cases indicating that the scheme error was negligible in comparison with the boundary condition error. The significance of this result may only be fully appreciated when it is observed that only $O(n)$ values are determined by the boundary conditions, while $O(n^2)$ values are determined by the scheme, where n is the number of grid points along the boundary.

Some possible extensions of this research are given below.

- Application of a transformation to the equation, and discretization using a wide-type scheme to further investigate the error resulting from boundary condition approximations.
- Analysis of the $O(h^6\,\mathrm{Re})$ method briefly mentioned in Section 3.1.3.
- Possible improvements in dealing with point singularities which occur in many problems such as driven cavity problems and sudden compression problems (see, e.g. [9, 21]).
- Extension of the wide and compact schemes to the time-dependent Navier-Stokes equations.
- Extension of compact schemes to three dimensional problems.

Appendix: 1

```
findif:=proc(ord,pt) local pts,inds,eqn,seqns,x,i,j;
#
#  This code uses values of U at the points in the list 'pt' to obtain the
#  highest order finite difference approximation to the 'ord' derivative of
#  U with respect to x. A third arguement(optional) is the number of Taylor
#  series terms to be retained for output.
#
   if type(pt,list) then pts:=pt;
   elif type(pt,integer) then pts:=[seq(i,i=-pt..pt)];
   else ERROR(`Incorrect input parameter for second arguement`) fi;
   if nargs>2 then keep:=args[3]: else keep:=0 fi:
   inds:=[seq(cat(c,i),i=1..nops(pts))];
   seqns:=inds;
   i:=0;  eqn:=0;  rem:=0;
   while eqn=0 and i<100 do
      if i=0 then
         eqn:=sum(seqns['j'],'j'=1..nops(pts)):
      else
         eqn:=sum(seqns['j']*(pts['j'])^i,'j'=1..nops(pts)):
      fi;
      if i=ord then eqn:=eqn-1; fi;
      eqn:=simplify(eqn);
      for j to nops(inds) do
         if has(eqn,inds[j]) then break; fi;
      od;
      if j<=nops(inds) then
         eqn:=inds[j]=solve(eqn,inds[j]);
         seqns:=subs(eqn,seqns);
         eqn:=0;
      fi;
      if eqn<>0 and keep>0 then
         rem:=rem-eqn*cat(U,x$i)*h^(i-ord)*ord!/i!:
         eqn:=0:
         keep:=keep-1:
      fi:
      i:=i+1;
   od;
   eqn:=1;
   for j to nops(seqns) do eqn:=ilcm(eqn,denom(normal(seqns[j]*ord!))); od;
   inds:=0;
```

```
   for j to nops(pts) do
      inds:=inds+eqn*ord!*seqns[j]*u(pts[j]);
   od;
   inds:=cat(U,x$ord)=inds/(eqn*h^ord)+rem+O(h^(i-ord-1));
   RETURN(inds);
end:
```

Appendix: 2

```
tay:=proc(ord,var) local expr,i,j:
#
# This routine generates an 'ord' order taylor series in V(x,y)
#
   clist:=NULL:
   expr:=0:
   for i from 0 to ord do
      for j from 0 to i do
         clist:=clist,V([var,i-j,j],x,y):
         expr:=expr+X^(i-j)*Y^(j)*V([var,i-j,j],x,y)/((i-j)!*(j)!):
      od:
   od:
   clist:=[clist]:
   expr;
end:


`diff/V`:=proc() local l:
#
# This routine tells maple how to differentiate V(x,y)
#
   l:=args[1]:
   if args[nargs]=x then V(subsop(2=(l[2]+1),l),x,y):
   elif args[nargs]=y then V(subsop(3=(l[3]+1),l),x,y):
   fi:
end:


buildeqns:=proc(nn,pts)
#
# This routine builds a structured list of linear equations which must be
# solved to obtain difference approximations to the linear 2-D p.d.e. 'eqn'.
# Equations are generated to give an error term of O(h^nn).
#
local t1,t2,cc,eqn,numl,expr,vars,dexpr,dvars,points,i,j,k;
#
   eqn:=V([1,2,0],x,y)+V([1,0,2],x,y)+V([2,0,0],x,y):
   num:=150:
#
#
```

```
   lprint(`Building Taylor Series:`);
   expr:=0: vars:=NULL:
   t1:=collect(subs(X=i1*h,Y=j1*h,tay(nn+1,1)),h):
   t2:=op(clist):
   for i to nops(pts) do
      expr:=expr+cat(b,i-1)*subs(i1=pts[i][1],j1=pts[i][2],t1):
      vars:=vars,cat(b,i-1):
   od:
   t1:=collect(h^2*subs(X=i1*h,Y=j1*h,tay(nn-1,2)),h):
   clist:=[t2,op(clist)]:
   for i to nops(pts) do
      expr:=expr+cat(c,i-1)*subs(i1=pts[i][1],j1=pts[i][2],t1):
      vars:=vars,cat(c,i-1):
   od:
   expr:=collect(expr,h):
#
# Here the contribution and differential extension of the PDE is
# taken into account
#
   lprint(`Adding contribution of PDE to system`):
   dexpr:=eqn: dvars:=NULL:
   for i from 1 to nn-1 do
      lprint(cat(`At order `,i));
      for j from 0 to i do
         cc:=x$(i-j),y$j:
         dexpr:=dexpr+cat(a,cc)*diff(eqn,cc)*h^i:
         dvars:=dvars,cat(a,cc):
      od:
   od:
   dexpr:=collect(dexpr,h):
#
# Here the final system is put into output format.
#
   expr:=collect(expr+h^2*dexpr,h):
   lprint(`Combining System into Final Array`);
   _eqns:=array(1..num):
   _coeff:=array(1..num):
   points:=NULL:
   j:=1:
   for k from 0 to nn+1 do
      coeff(expr,h,k):
      collect(",clist,distributed):
      cc:=[coeffs(",clist,'t2')]:
      t2:=[t2]:
      for i to nops(cc) do
         _eqns[j]:=cc[i]:
         _coeff[j]:=t2[i]:
         j:=j+1:
      od:
      points:=points,j-1:
   od:
   [[points],[vars],[dvars]]
end:
```

```
solveme:=proc(ptl,orrd)
#
#  This routine solves the system output by ’buildeqns’, and saves it
#  as each order of accuracy greater than or equal to ’orrd’ is passed.
#  These systems are stored in ’ERROR(order).m’ where (order) refers to
#  the local truncation error order. This is continued until an
#  inconsistent equation is found, or all relations are satisfied.
#
local neqns,unks,incons,ceqn,seqn,cord,tonext,eqn,unnum,var,sol,flag,i:

   for i to 1000 do
      if not(assigned(_eqns[i])) then break: fi:
   od:
   neqns:=i-1:
   unks:=[op(ptl[2]),op(ptl[3])]:
   _seqns:=array(1..1000):
   incons:=NULL:
   ceqn:=1: seqn:=0: cord:=0:
   tonext:=ptl[1][cord+1]:
   flag:=false:
   while not(flag) and ceqn<=neqns do
      if not(_eqns[ceqn]=0) then
         eqn:=_eqns[ceqn]:
         unnum:=findnum(eqn,unks):
         if unnum=0 then
            incons:=incons,[ceqn,_coeff[ceqn],eqn]:
            print(‘inconsistent equation:‘,eqn);
            print(‘inconsistent coefficient:‘,_coeff[ceqn]);
         else
            var:=unks[unnum]:
            sol:=var=solve(eqn=0,var):
            lprint(cat(‘Solved #‘,ceqn,‘ for ‘,var,‘   Total of ‘,seqn+1,
                       ‘ solved ‘,_coeff[ceqn]));
            if seqn>0 then
               for i to seqn do
                  if has(_seqns[i],var) then
                     simplify(subs(sol,_seqns[i])):
                     _seqns[i]:=":
                  fi:
               od:
            fi:
            seqn:=seqn+1:
            _seqns[seqn]:=sol:
            for i from ceqn to neqns do
               if has(_eqns[i],var) then
                  simplify(subs(sol,_eqns[i])):
                  _eqns[i]:=":
               fi:
            od:
         fi:
      fi:
```

```
      if ceqn=tonext then
            lprint(cat(`***Equations for order `,cord-2,` completed`));
            incons:=[incons]:
            cord:=cord+1:
            if cord-2>=orrd then
               lprint(`Saving ...`);
               save _seqns,_eqns,_coeff,incons,seqn,ceqn,`ERROR`.cord.`.m`:
            fi:
            if incons<>[] then flag:=true: fi:
            incons:=op(incons):
            if nops(ptl[1])>cord then
               tonext:=ptl[1][cord+1]:
            else
               flag:=true:
            fi:
      fi:
      ceqn:=ceqn+1:
   od:
   if incons=NULL then
      lprint(cat(`Number of solved equations:`,seqn));
      lprint(cat(`Number of free parameters:`,nops(unks)-seqn));
   fi:
   cord:=NULL:
   for i to seqn do
      cord:=cord,_seqns[i]:
   od:
   [cord]:
end:


findnum:=proc(eqn,unks) local i:
   for i to nops(unks) do
      if has(eqn,unks[i]) then break; fi;
   od;
   if i>nops(unks) then 0;
   else i;
   fi;
end:
#
#  The following are the procedure calls which derive the scheme
#
pts:=[[0,0],
[1,0],[0,1],[-1,0],[0,-1],
[1,1],[-1,1],[-1,-1],[1,-1]]:
ptl:=buildeqns(4,pts):
save _eqns,_coeff,ptl,`builtsys.m`:
solsys:=solveme(ptl,2);
```

Appendix: 3

```
tay:=proc(ord) local expr,i,j:
#
#  This routine generates an 'ord' order taylor series in V(x,y)
#
   clist:=NULL:
   expr:=0:
   for i from 0 to ord do
      for j from 0 to i do
         clist:=clist,V([i-j,j],x,y):
         expr:=expr+X^(i-j)*Y^(j)*V([i-j,j],x,y)/((i-j)!*(j)!):
      od:
   od:
   clist:=[clist]:
   expr;
end:


`diff/V`:=proc() local l:
#
#  This routine tells maple how to differentiate V(x,y)
#
   l:=args[1]:
   if args[nargs]=x then V(subsop(1=(l[1]+1),l),x,y):
   elif args[nargs]=y then V(subsop(2=(l[2]+1),l),x,y):
   fi:
end:


buildeqns:=proc(nn,pts,oeq,leq,nleq)
#
#  This routine builds a structured list of linear equations which must be
#  solved to obtain difference approximations to a 'oeq' order equation
#  with linear part 'leq', and quadratic part 'nleq', using the grid points
#  in the list 'pts'. The taylor series will be calculated up to 'nn' order.
#
local tp1,tp2,t1,t2,cc,lu,nlu,uu,pn,pl,numn,numl,nnon,nlin,lexpr,nlexpr,i,j;
#
#  These variables define the maximum number of equations which may be derived
#
   numl:=150:  numn:=800:
```

```
#
#  Here the grid taylor series for the linear and quadratic terms are obtained.
#
   lprint(`Building Taylor Series:`);
   tp1:=tay(nn):
   t1:=collect(subs(X=i1*h,Y=j1*h,tp1),h):
   t2:=collect(subs(X=i2*h,Y=j2*h,tp1),h):
   tp1:=collect(subs(X=i0*h,Y=j0*h,tp1),h):
   tp2:=collect(t1*t2,h):
#
#  Here the quadratic taylor series is split into the terms which will
#  become our equations, and the pointer list is constructed.
#
   lprint(`Seperating Taylor Series:`);
   _points:=array(0..nn,1..2):
   _nleqns:=array(1..numn):
   _tnleqns:=array(1..numn):
   _nlcoeff:=array(1..numn):
   j:=1:
   for i from 0 to nn do
      t2:=coeff(tp2,h,i):
      t2:=collect(t2,clist,distributed):
      cc:=[coeffs(t2,clist,'t2')]:
      t2:=[t2]:
      for k to nops(t2) do
         _nleqns[j]:=0:
         _tnleqns[j]:=cc[k]:
         _nlcoeff[j]:=t2[k]:
         j:=j+1:
      od:
      _points[i,2]:=j-1:
   od:
   nnon:=j-1:
#
#  Here the linear taylor series is split into the terms which will
#  become our equations, and the pointer list is constructed.
#
   _leqns:=array(1..numl):
   _tleqns:=array(1..numl):
   _lcoeff:=array(1..numl):
   j:=1:
   for i from 0 to nn do
      t1:=coeff(tp1,h,i):
      t1:=collect(t1,clist,distributed):
      cc:=[coeffs(t1,clist,'t1')]:
      t1:=[t1]:
      for k to nops(t1) do
         _leqns[j]:=0:
         _tleqns[j]:=cc[k]:
         _lcoeff[j]:=t1[k]:
         j:=j+1:
      od:
      _points[i,1]:=j-1:
```

```
   od:
   nlin:=j-1:
   lprint(cat(`There are `,nlin,` linear and `,nnon,` non-linear equations`));
#
#  Here the points are looped though, and the linear equations are constructed.
#
   lprint(`Combining Linear Components`);
   lu:=NULL:
   for i to nops(pts) do
      lprint(cat(`At point #`,i-1)):
      teqns:=subs(i0=pts[i][1],j0=pts[i][2],copy(_tleqns)):
      for k to nlin do
         _leqns[k]:=_leqns[k]+cat(c_,i-1)*teqns[k]:
      od:
      lu:=lu,cat(c_,i-1):
   od:
   lprint(`Combining Non-Linear Components`);
   nlu:=NULL:
   for i to nops(pts) do
      teqns:=subs(i1=pts[i][1],j1=pts[i][2],copy(_tnleqns)):
      for j to i do
         lprint(cat(`At point #`,i-1,`x`,j-1)):
         teqns2:=subs(i2=pts[j][1],j2=pts[j][2],copy(teqns)):
         for k to nnon do
            _nleqns[k]:=_nleqns[k]+cat(c_,i-1,_,j-1)*teqns2[k]:
         od:
         nlu:=nlu,cat(c_,i-1,_,j-1):
      od:
   od:
   lu:=[lu]: nlu:=[nlu]:
#
#  Here the PDE is used, and differential consequences of the PDE are taken
#  into account.
#
   uu:=NULL:
   if oeq<=nn then
      lprint(`Adding contribution of PDE to system`):
      for i from oeq to nn do
         lprint(cat(`At order `,i));
         if i=oeq then
            t1:=-leq:
            t2:=-nleq:
         else
            t1:=0: t2:=0:
            for j from 0 to i-oeq do
               cc:=x$(i-oeq-j),y$j:
               t1:=t1-cat(a,cc)*diff(leq,cc):
               t2:=t2-cat(a,cc)*diff(nleq,cc):
               uu:=uu,cat(a,cc):
            od:
         fi:
         t1:=collect(t1,clist):
         t2:=collect(t2,clist,distributed):
```

```
         cc:=[coeffs(t1,clist,'inds')]:
         inds:=[inds]:
         for j from _points[i-1,1]+1 to _points[i,1] do
            if member(_lcoeff[j],inds,'k') then
               _leqns[j]:=_leqns[j]+cc[k]:
               cc:=subsop(k=NULL,cc):
               inds:=subsop(k=NULL,inds):
            fi:
         od:
         if inds<>[] then ERROR(`Not all linear terms present for PDE`); fi;
         cc:=[coeffs(t2,clist,'inds')]:
         inds:=[inds]:
         for j from _points[i-1,2]+1 to _points[i,2] do
            if member(_nlcoeff[j],inds,'k') then
               _nleqns[j]:=_nleqns[j]+cc[k]:
               cc:=subsop(k=NULL,cc):
               inds:=subsop(k=NULL,inds):
            fi:
         od:
         if inds<>[] then ERROR(`Not all non-linear terms present for PDE`); fi;
      od;
   fi;
   uu:=[uu]:
#
#  Here the final system is put into output format.
#
   lprint(`Combining System into Final Array`);
   _eqns:=array(1..(numl+numn)):
   _coeff:=array(1..(numl+numn)):
   points:=NULL:
   j:=1:
   pn:=0:
   pl:=0:
   for k from 0 to nn do
      for i from pn+1 to _points[k,2] do
         _eqns[j]:=_nleqns[i]:
         _coeff[j]:=_nlcoeff[i]:
         j:=j+1:
      od:
      pn:=i-1:
      t2:=j-1:
      for i from pl+1 to _points[k,1] do
         _eqns[j]:=_leqns[i]:
         _coeff[j]:=_lcoeff[i]:
         j:=j+1:
      od:
      pl:=i-1:
      points:=points,[t2,j-1]:
   od:
   [[points],lu,nlu,uu];
end:
```

```
solveme:=proc(ptl,orrd)
#
#  This routine solves the system output by 'buildeqns', and saves it
#  as each order of accuracy greater than or equal to 'orrd' is passed.
#  These systems are stored in 'ERROR(order).m' where (order) refers to
#  the local truncation error order. This is continued until an
#  inconsistent equation is found, or all relations are satisfied.
#
local neqns,unks,incons,ceqn,seqn,cord,tonlin,tolin,eqn,unnum,var,sol,flag,i:

   for i to 1000 do
      if not(assigned(_eqns[i])) then break: fi:
   od:
   neqns:=i-1:
   unks:=[op(ptl[3]),op(ptl[2]),op(ptl[4])]:
   _seqns:=array(1..1000):
   incons:=NULL:
   ceqn:=1:  seqn:=0:  cord:=0:
   tonlin:=ptl[1][cord+1][1]:
   tolin:=ptl[1][cord+1][2]:
   flag:=false:
   while not(flag) and ceqn<=neqns do
      if not(_eqns[ceqn]=0) then
         eqn:=_eqns[ceqn]:
         unnum:=findnum(eqn,unks):
         if unnum=0 then
            incons:=incons,[ceqn,_coeff[ceqn],eqn]:
            print(`inconsistent equation:`,eqn);
            print(`inconsistent coefficient:`,_coeff[ceqn]);
         else
            var:=unks[unnum]:
            sol:=var=solve(eqn=0,var):
            lprint(cat(`Solved #`,ceqn,` for `,var))
            lprint(cat(`Total of `,seqn+1,` solved `,_coeff[ceqn]));
            if seqn>0 then
               for i to seqn do
                  if has(_seqns[i],var) then
                     simplify(subs(sol,_seqns[i])):
                     _seqns[i]:=":
                  fi:
               od:
            fi:
            seqn:=seqn+1:
            _seqns[seqn]:=sol:
            for i from ceqn to neqns do
               if has(_eqns[i],var) then
                  simplify(subs(sol,_eqns[i])):
                  _eqns[i]:=":
               fi:
            od:
         fi:
      fi:
      if ceqn=tonlin then
```

```
            lprint(cat(`***Non-Linear equations for order `,cord,` completed`));
      elif ceqn=tolin then
            lprint(cat(`***Linear equations for order `,cord,` completed`));
            incons:=[incons]:
            cord:=cord+1:
            if cord>=orrd then
               lprint(`Saving ...`);
               save _seqns,_eqns,_coeff,incons,seqn,ceqn,`ERROR`.cord.`.m`:
            fi:
            if incons<>[] then flag:=true: fi:
            incons:=op(incons):
            if nops(ptl[1])>cord then
               tonlin:=ptl[1][cord+1][1]:
               tolin:=ptl[1][cord+1][2]:
            else
               flag:=true:
            fi:
      fi:
      ceqn:=ceqn+1:
   od:
   if incons=NULL then
      lprint(cat(`Number of solved equations:`,seqn));
      lprint(cat(`Number of free parameters:`,nops(unks)-seqn));
   fi:
   cord;
end:


findnum:=proc(eqn,unks) local i:
   for i to nops(unks) do
      if has(eqn,unks[i]) then break; fi;
   od;
   if i>nops(unks) then 0;
   else i;
   fi;
end:
#
#  The following are the procedure calls which derive the scheme
#
pts:=[[0,0],[1,0],[0,1],[-1,0],[0,-1],
[1,1],[-1,1],[-1,-1],[1,-1],
[2,0],[0,2],[-2,0],[0,-2],
[2,-1],[2,1],[1,2],[-1,2],[-2,1],[-2,-1],[-1,-2],[1,-2],
[2,2],[-2,2],[-2,-2],[2,-2],
[3,0],[0,3],[-3,0],[0,-3]]:
leq:=V([4,0],x,y)+2*V([2,2],x,y)+V([0,4],x,y):
nleq:=V([1,0],x,y)*(V([2,1],x,y)+V([0,3],x,y))
-V([0,1],x,y)*(V([3,0],x,y)+V([1,2],x,y)):
ptl:=buildeqns(7,pts,4,leq,nleq):
save _eqns,_coeff,ptl,`builtsys.m`:
solsys:=solveme(ptl,4):
```

Appendix: 4

```
check:=proc(l)
#
#  This procedure constructs second-order or fourth-order accurate central
#  difference schemes for up to sixth order derivatives.
#
local d,cm,il,i,j,k;
#
   d:=array[1..2,0..6];
   d[1,0]:=[   0,   0,   0,   1,   0,   0,   0];
   d[1,1]:=[   0,   0,-1/2,   0, 1/2,   0,   0];
   d[1,2]:=[   0,   0,   1,  -2,   1,   0,   0];
   d[1,3]:=[   0,-1/2,   1,   0,  -1, 1/2,   0];
   d[1,4]:=[   0,   1,  -4,   6,  -4,   1,   0];
   d[1,5]:=[-1/2,   2,-5/2,   0, 5/2,  -2, 1/2];
   d[1,6]:=[   1,  -6,  15, -20,  15,  -6,   1];
   d[2,0]:=[   0,   0,   0,   1,   0,   0,   0];
   d[2,1]:=[   0,1/12,-2/3,   0,2/3,-1/12,   0];
   d[2,2]:=[  0,-1/12, 4/3,-5/2,4/3,-1/12,   0];
   d[2,3]:=[ 1/8,  -1,13/8,  0,-13/8,   1,-1/8];
   d[2,4]:=[-1/6,  2,-13/2,28/3,-13/2,  2,-1/6];
   cm:=array(1..8,1..7);
   for i to 7 do
      for j to 7 do
         cm[i,j]:=0;
      od;
   od;
   for i to 7 do
      for j to 7 do
         for k to nops(l) do
            cm[8-j,i]:=cm[8-j,i]
                       +l[k][1]*d[l[k][2],l[k][3]][i]*d[l[k][2],l[k][4]][j];
         od;
      od;
   od;
   il:=0;
   for i to 7 do
      cm[8,i]:=0:
      for j to 7 do
         il:=il+cat(_temp_,i-4,_,4-j)*cm[j,i];
      od;
```

```
   od;
   normal(il);
   cm[8,1]:=denom(");
   RETURN(copy(cm));
end:


#  Here the difference schemes for the required derivatives are obtained,
#  printed to 'file.g', and stored as global variables for later use in the
#  following procedure calls.
#
writeto(`file.g`);
l1:=[[1,2,4,0],[2,2,2,2],[1,2,0,4]]:
av:=check(l1);
l1:=[[1,2,3,0],[1,2,1,2]]:
bv1:=check(l1);
l1:=[[1,2,2,1],[1,2,0,3]]:
bv2:=check(l1);
l1:=[[1,2,1,0]]:
pvx:=check(l1);
l1:=[[1,2,0,1]]:
pvy:=check(l1);

#  All parameters must be determined here, as the following code is
#  constructed for constant coefficient schemes.

sd:=proc(ind1,ind2)
#
#  This procedure constructs the Jacobian element for the (ind1*h,ind2*h)
#  point using the difference schemes previously derived, and ouputs it
#  in double precision Fortran code.
#
local aa,bb,t1,t2,t3,t4,t5,t6;
#
   aa:=4-ind2;
   bb:=4+ind1;
   t1:=av[aa,bb]:
   t2:=bv1[aa,bb]:
   t3:=bv2[aa,bb]:
   t4:=pvx[aa,bb]:
   t5:=pvy[aa,bb]:
   t6:=t1+Re*(t3*px+t4*b2-t2*py-t5*b1);
   tt:=denom(normal(t6));
   if tt<0 then tt:=-tt; fi;
   t1:=tt*av[aa,bb]:
   if t1<0 then t1:=-cat(-t1,D0);
   elif t1>0 then t1:=cat(t1,D0) fi;
   t2:=tt*bv1[aa,bb]:
   if t2<0 then t2:=-cat(-t2,D0);
   elif t2>0 then t2:=cat(t2,D0) fi;
   t3:=tt*bv2[aa,bb]:
   if t3<0 then t3:=-cat(-t3,D0);
   elif t3>0 then t3:=cat(t3,D0) fi;
```

```
   t4:=tt*pvx[aa,bb]:
   if t4<0 then t4:=-cat(-t4,D0);
   elif t4>0 then t4:=cat(t4,D0) fi;
   t5:=tt*pvy[aa,bb]:
   if t5<0 then t5:=-cat(-t5,D0);
   elif t5>0 then t5:=cat(t5,D0) fi;
   if tt=1 then
      t6:=t1+Re*(t3*px+t4*b2-t2*py-t5*b1);
   else
      t6:=(t1+Re*(t3*px+t4*b2-t2*py-t5*b1))/cat(tt,D0);
   fi;
   if t6<>0 then
      lprint(`   `,jel(ind1,ind2)=t6);
   fi;
end:


makeproc:=proc(nam,gr)
#
#  This procedure inputs a difference grid 'gr', and outputs fortran code
#  required to evaluate the difference scheme.
#
local l,ll,cc,i,j,k,m;

   l:=NULL;
   for m from 1 to 7 do
      for k from 1 to 7 do
         l:=l,abs(gr[k,m]);
      od;
   od;
   l:={l} minus {0};
   while l<>{} do
      cc:=max(op(l));
      l:=l minus {cc};
      ll:=0;
      for m to 7 do
         for k to 7 do
               if abs(gr[k,m])=abs(cc) then
                  if ll=0 and abs(gr[k,m])<>gr[k,m] and cc<>1 then
                     cc:=-cc;
                  fi;
                  ll:=ll+gr[k,m]/cc*psi(i+m-4,j+4-k);
               fi;
         od;
      od;
      cc:=cc*gr[8,1]:
      if cc>0 and cc<>1 then
         if type(ll,`+`) then
            lprint(cat(`   `,nam,`=`,nam,`+`,cc,`D0*(`),ll,`)`);
         else
            lprint(cat(`   `,nam,`=`,nam,`+`,cc,`D0*`),ll);
         fi;
      elif cc<0 then
```

```
         if type(ll,`+`) then
            lprint(cat(`   `,nam,`=`,nam,cc,`D0*(`),ll,`)`);
         else
            lprint(cat(`   `,nam,`=`,nam,cc,`D0*`),ll);
         fi;
      else
         lprint(cat(`   `,nam,`=`,nam,`+`),ll);
      fi;
   od;
   lprint(cat(`   `,nam,`=`,nam,`/`,gr[8,1],`D0`));
   lprint();
end:

writeto(`file.f`);
#
#  Here the difference schemes are constructed and output to `file.f`
#
makeproc(`a`,av);
makeproc(`b1`,bv1);
makeproc(`b2`,bv2);
makeproc(`px`,pvx);
makeproc(`py`,pvy);

#
#  Here the Jacobian elements are constructed and output to `file.f`
#
ord:=[[0,0],
      [1,0],[-1,0],[0,1],[0,-1],
      [1,1],[-1,1],[-1,-1],[1,-1],
      [2,0],[-2,0],[0,2],[0,-2],
      [2,1],[1,2],[-1,2],[-2,1],[-2,-1],[-1,-2],[1,-2],[2,-1],
      [2,2],[-2,2],[-2,-2],[2,-2],
      [3,0],[0,3],[-3,0],[0,-3],
      [3,1],[1,3],[-1,3],[-3,1],
      [-3,-1],[-1,-3],[1,-3],[3,-1],
      [3,2],[2,3],[-2,3],[-3,2],[-3,-2],[-2,-3],[2,-3],[3,-2],
      [3,3],[-3,3],[-3,-3],[3,-3]]:

for i to nops(ord) do
   sd(op(ord[i]));
od;
writeto(terminal);

#
#  Here we present the listing of `file.f` when the above code is executed.
#
c  These are the required differences for the Wide scheme
c
   a=a+2244D0*   psi(i,j)
   a=a-948D0*(   psi(i-1,j)+psi(i,j+1)+psi(i,j-1)+psi(i+1,j)   )
   a=a+256D0*(   psi(i-1,j+1)+psi(i-1,j-1)+psi(i+1,j+1)+psi(i+1,j-1)   )
   a=a+174D0*(   psi(i-2,j)+psi(i,j+2)+psi(i,j-2)+psi(i+2,j)   )
   a=a-16D0*(   psi(i-2,j+1)+psi(i-2,j-1)+psi(i-1,j+2)+psi(i-1,j-2)
```

```
+psi(i+1,j+2)+psi(i+1,j-2)+psi(i+2,j+1)+psi(i+2,j-1)   )
   a=a-12D0*(   psi(i-3,j)+psi(i,j+3)+psi(i,j-3)+psi(i+3,j)   )
   a=a+   psi(i-2,j+2)+psi(i-2,j-2)+psi(i+2,j+2)+psi(i+2,j-2)
   a=a/72D0

   b1=b1+474D0*(   psi(i-1,j)-psi(i+1,j)   )
   b1=b1-174D0*(   psi(i-2,j)-psi(i+2,j)   )
   b1=b1-128D0*(   psi(i-1,j+1)+psi(i-1,j-1)-psi(i+1,j+1)-psi(i+1,j-1)   )
   b1=b1+18D0*(   psi(i-3,j)-psi(i+3,j)   )
   b1=b1+16D0*(   psi(i-2,j+1)+psi(i-2,j-1)-psi(i+2,j+1)-psi(i+2,j-1)   )
   b1=b1+8D0*(   psi(i-1,j+2)+psi(i-1,j-2)-psi(i+1,j+2)-psi(i+1,j-2)   )
   b1=b1-1D0*(   psi(i-2,j+2)+psi(i-2,j-2)-psi(i+2,j+2)-psi(i+2,j-2)   )
   b1=b1/144D0

   b2=b2-474D0*(   psi(i,j+1)-psi(i,j-1)   )
   b2=b2+174D0*(   psi(i,j+2)-psi(i,j-2)   )
   b2=b2+128D0*(   psi(i-1,j+1)-psi(i-1,j-1)+psi(i+1,j+1)-psi(i+1,j-1)   )
   b2=b2-18D0*(   psi(i,j+3)-psi(i,j-3)   )
   b2=b2-16D0*(   psi(i-1,j+2)-psi(i-1,j-2)+psi(i+1,j+2)-psi(i+1,j-2)   )
   b2=b2-8D0*(   psi(i-2,j+1)-psi(i-2,j-1)+psi(i+2,j+1)-psi(i+2,j-1)   )
   b2=b2+   psi(i-2,j+2)-psi(i-2,j-2)+psi(i+2,j+2)-psi(i+2,j-2)
   b2=b2/144D0

   px=px-8D0*(   psi(i-1,j)-psi(i+1,j)   )
   px=px+   psi(i-2,j)-psi(i+2,j)
   px=px/12D0

   py=py+8D0*(   psi(i,j+1)-psi(i,j-1)   )
   py=py-1D0*(   psi(i,j+2)-psi(i,j-2)   )
   py=py/12D0

c
c  These are the Jacobian elements for the Wide scheme
c
      jel(0,0) = 187D0/6D0
      jel(1,0) = (-316D0+Re*(16D0*b2+79D0*py))/24D0
      jel(-1,0) = (-316D0+Re*(-16D0*b2-79D0*py))/24D0
      jel(0,1) = (-316D0+Re*(-79D0*px-16D0*b1))/24D0
      jel(0,-1) = (-316D0+Re*(79D0*px+16D0*b1))/24D0
      jel(1,1) = (32D0+Re*(8D0*px-8D0*py))/9D0
      jel(-1,1) = (32D0+Re*(8D0*px+8D0*py))/9D0
      jel(-1,-1) = (32D0+Re*(-8D0*px+8D0*py))/9D0
      jel(1,-1) = (32D0+Re*(-8D0*px-8D0*py))/9D0
      jel(2,0) = (58D0+Re*(-2D0*b2-29D0*py))/24D0
      jel(-2,0) = (58D0+Re*(2D0*b2+29D0*py))/24D0
      jel(0,2) = (58D0+Re*(29D0*px+2D0*b1))/24D0
      jel(0,-2) = (58D0+Re*(-29D0*px-2D0*b1))/24D0
      jel(2,1) = (-4D0+Re*(-1D0*px+2D0*py))/18D0
      jel(1,2) = (-4D0+Re*(-2D0*px+1D0*py))/18D0
      jel(-1,2) = (-4D0+Re*(-2D0*px-1D0*py))/18D0
      jel(-2,1) = (-4D0+Re*(-1D0*px-2D0*py))/18D0
      jel(-2,-1) = (-4D0+Re*(1D0*px-2D0*py))/18D0
      jel(-1,-2) = (-4D0+Re*(2D0*px-1D0*py))/18D0
```

```
jel(1,-2) = (-4D0+Re*(2D0*px+1D0*py))/18D0
jel(2,-1) = (-4D0+Re*(1D0*px+2D0*py))/18D0
jel(2,2) = (2D0+Re*(1D0*px-1D0*py))/144D0
jel(-2,2) = (2D0+Re*(1D0*px+1D0*py))/144D0
jel(-2,-2) = (2D0+Re*(-1D0*px+1D0*py))/144D0
jel(2,-2) = (2D0+Re*(-1D0*px-1D0*py))/144D0
jel(3,0) = (-4D0+Re*3D0*py)/24D0
jel(0,3) = (-4D0-Re*3D0*px)/24D0
jel(-3,0) = (-4D0-Re*3D0*py)/24D0
jel(0,-3) = (-4D0+Re*3D0*px)/24D0
```

Bibliography

[1] L. M. ADAMS, R. J. LEVEQUE AND D. M. YOUNG, "Analysis of the S.O.R. iteration for the 9 point laplacian", *SIAM J. Numer. Anal.* **25**, 1156 (1988).

[2] G. K. BATCHELOR, *An Introduction to Fluid Dynamics* pp.167, (Springer–Verlag, New York, 1983).

[3] A. S. BENJAMIN AND V. E. DENNY, "On the convergence of numerical solutions for 2-D flows in a cavity at high Re" *J. Comput. Phys.* **33**, 340 (1979).

[4] J. S. BRAMLEY AND D. M. SLOAN, "A comparison of an upwind scheme with a central difference scheme for moderate Reynolds number", Department Report # 3, Department of Mathematics, Univ. of Strathclyde, Glasgow, Scotland, 1988 (unpublished).

[5] G. Q. CHEN, Z. GAO AND Z. F. YANG, "A perturbational h^4 exponential finite difference scheme for the convective diffusion equation", *J. Comput. Phys.* **104**, 129 (1993).

[6] L. COLLATZ, *The Numerical Treatment of Differential Equations*, (Springer–Verlag, Berlin/New York, 1960).

[7] S. C. R. DENNIS AND J. D. HUDSON, "A difference method for solving the Navier-Stokes equations", *Proc 1st Int. Conf. Numer. Methods Laminar and Turbulent Flow* pp.69, (Pentech Press, London, 1978).

[8] S. C. R. DENNIS AND J. D. HUDSON, "Compact h^4 finite-difference approximations to operators of Navier-Stokes type", *J. Comput. Phys.* **85**, 390 (1989).

[9] S. C. R. DENNIS AND F. T. SMITH, "Steady flow through a channel with a symmetrical constriction in the form of a step", *Pro. R. Soc. Lond. A* **372**, 393 (1980).

[10] B. FORNBERG, "Steady incompressable flow past a row of circular cyllinders", *J. Fluid Mech.* **225**, 655 (1991).

[11] B. FORNBERG, "Generation of finite difference formulas on arbitrarily spaced grids", *J. Math Comput.* **51**, 699 (1988).

[12] B. FORNBERG, "A compact fourth order finite difference scheme for the steady incompressable Navier-Stokes equations", Exxon Research and Engineering Co., Annandale, NJ 08801, USA, 1992 (unpublished report).

[13] U. GHIA, K. N. GHIA AND C. T. SHIN, "High-Re solutions for incompressable flow using the Navier-Stokes equations and a multigrid method", *J. Comput. Phys.* **48**, 387 (1982).

[14] J. W. GOODRICH AND W. Y. SOH, "Time-dependent viscous incompressable Navier-Stokes equations: The finite difference Galerkin formulation and streamfunction algorithms", *J. Comput. Phys.* **84**, 207 (1989).

[15] P. M. GRESHO,"Incompressible fluid dynamics: some fundamental formulation issues", *Annu. Rev. Fluid Mech.* **23**, 413 (1991).

[16] M. M. GUPTA, "High accuracy solutions of incompressable Navier-Stokes equations", *J. Comput. Phys.* **93**, 343 (1991).

[17] M. M. GUPTA AND R. P. MANOHAR, "Boundary approximations and accuracy in viscous flow computations", *J. Comput. Phys.* **31**, 265 (1979).

[18] M. M. GUPTA, R. MANOHAR, AND J. W. STEPHENSON,"High-order difference schemes for two-dimensional elliptic equations", *Num. Methods Partial Diff. Equations* **1**, 71 (1985).

[19] T. Y. HOU AND B. T. R. WETTON, "Stable fourth order stream function methods for incompressible flows with boundaries", (1993) Submitted.

[20] H. HUANG AND H. YANG, "The computational boundary method for solving the Navier–Stokes equations", Research Report, Institute of Applied Mathematics, University of British Columbia, Canada, 1990 (unpublished).

[21] R. HUNT, "The numerical solution of the laminar flow in a constricted channel at moderately high Reynolds number using Newton iteration" *J. Numer. Meth. in Fluids* **11**, 247 (1990).

[22] D. B. INGHAM, T. TANG AND B. R. MORTON, "Steady 2-D flow through a row of normal flat plates" *J. Fluid Mech.* **210**, 281 (1990).

[23] M. LI, T. TANG AND B. FORNBERG, "A compact fourth order finite difference scheme for the steady incompressible Navier-Stokes equations", Department Report # 93-13, Department of Mathematics and Statistics, Simon Fraser University, Burnaby, B.C., Canada.

[24] H. K. MOFFAT, "Viscous and resistive eddies near a sharp corner", *J. Fluid Mech.* **18**, 1 (1964).

[25] H. NISHIDA AND N. SATOFUKA, "Higher-order solutions of square driven cavity flow using a variable-order multi-grid method", *Int. J. Numer. Methods Eng.* **34**, 637 (1992).

[26] R. PEYRET AND T. D. TAYLOR, *Computational Methods for Fluid Flow* pp.182, (Cambridge University Press, Great Britain, 1967).

[27] C. W. RICHARDS AND C. M. CRANE, "The accuracy of finite difference schemes for the numerical solution of the Navier-Stokes equations", *Appl. Math. Modeling* **3**, 205 (1979).

[28] R. SCHREIBER AND H. B. KELLER, "Driven cavity flows by efficient numerical techniques", *J. Comput. Phys.* **49**, 310 (1983).

[29] R. SCHREIBER AND H. B. KELLER, "Spurious solutions in driven cavity calculations", *J. Comput. Phys.* **49**, 165 (1983).

[30] J. C. STRIKWERDA, *Finite Difference Schemes and Partial Differential Equations* pp.293, (Wadsworth & Brooks, Belmont, California, 1989).

[31] L. C. WOODS, “A note on the numerical solution of fourth order differential equations”, *Aeronaut. Q.* **5**, 176 (1954).