12

Solving Fluid Dynamics Problems with Matlab

Rui M. S. Pereira[1] and Jitesh S. B. Gajjar[2]
[1]*University of Minho*
[2]*University of Manchester*
[1]*Portugal*
[2]*United Kingdom*

1. Introduction

MATLAB (short for Matrix Laboratory) was created by Cleve Moler and Jack Little in the 1970's. It is a programming language for technical computing. Its environment is easy to work with, the syntax is very simple and intuitive, it has powerful toolboxes to treat many different problems in engineering, and it allows us to produce fantastic graphics as the programme runs. It also allows us to create a graphical interface (via graphical user interfaces - GUIs) that gives our programme a look that is very close to professional software.
Because of many of the mentioned features, a MATLAB code can be very compact, allowing anyone to have "the big picture" of any code without have to look at all its details. Another great advantage of Matlab is that, if the code is written in a vectorized form, the code can run much faster than if it was written in the traditional form (*'a la C/fortran'*). The fact that MATLAB allows us to use a powerful toolbox for sparse matrices, is also a great advantage since, many traditional linear algebra operations can be highly improved, allowing the codes to run much faster than it would run with the traditional linear algebra functions.
In our work we have made extensive use of MATLAB to do 'proof of concept' studies, especially when developing new algorithms and techniques for solving systems of coupled nonlinear partial differential equations, such as those which arise in fluid dynamics. This includes, for instance, codes for investigating instabilities in lid-driven cavities, Boppana and Gajjar (2010), instabilities in flow past circular cylinders, Boppana and Gajjar (2011), and transonic flow past aerofoils Pereira and Gajjar (2010). In some cases MATLAB is used in its own right for solving small problems, but the fact that MATLAB is an interpreted language means that for increasing problem sizes, the MATLAB version of the code can be much slower than equivalent versions in other languages especially when one is dealing with very large sparse matrices. On the other hand the beauty of MATLAB is that much of the hard work is buried in the simple syntax and hidden from the user. An example of this is the use of the backslash operator for solving linear systems. Whether the system is sparse or full, the manner in which the equations are solved is hidden from the user and this greatly facilitates code development. In the equivalent fortran versions of the code the replacement for the '\' operation requires considerable work and the code translation process is no longer a trivial exercise.
In this chapter we will discuss the use of hybrid spectral methods to solve two and three-dimensional problems using MATLAB. There is an excellent book by Trefethen (2000) which discusses the application of spectral methods using MATLAB to solve ordinary and

partial differential equations, and which provides the foundation for the techniques described below.

To motivate the ideas we first consider the solution of a model elliptic equation of the form

$$a(x,y)\psi_{xx} + b(x,y)\psi_{yy} + c(x,y)\psi_x + e(x,y)\psi_y + f(x,y)\psi = g(x,y),$$

say with Dirichlet boundary conditions in a rectangular domain. To obtain a numerical solution to this problem the first step is to choose an appropriate method and discretization. In our work we have used a combination of spectral methods in one or two dimensions and high order finite difference methods in another dimension. The main reasons for this choice are that a hybrid approach combines the accuracy of spectral methods together with flexibility in comparison to using spectral methods on their own. One restriction with the use of spectral methods is that one needs to use somewhat simplified geometries. A hybrid approach gives more flexibility in this respect. Another important reason stems from consideration of the type of matrix patterns which arise. With finite differences in say the x direction and spectral collocation in the y direction, the coefficient matrix with the unknowns ordered in terms of increasing y values for a fixed x value, has a particular sparsity pattern dependent on the order of the finite-differencing used. Second order finite-differencing leads to a a block tridiagonal matrix whilst with fourth-order finite differences, the matrix is block-pentadiagonal of the form:

$$\mathbf{A}_q\mathbf{\Psi}_{q-2} + \mathbf{B}_q\mathbf{\Psi}_{q-1} + \mathbf{C}_q\mathbf{\Psi}_q + \mathbf{D}_q\mathbf{\Psi}_{q+1} + \mathbf{E}_q\mathbf{\Psi}_{q+2} = \mathbf{R}_q, \quad q = 0, 1, ..., M. \tag{1}$$

Here $\mathbf{\Psi_q}$ is the vector of unknowns at location $x = x_q$, $M+1$ is typically the number of points in the x direction and the block matrices are of size $N+1$ by $N+1$ where $N+1$ spectral points are used.

Using MATLAB it is not too difficult to generate a short code to solve the above discrete system and the book by Trefethen (2000) gives plenty of such examples. The problem becomes more challenging when N and M become large, as for example in some fluid flow applications where large N, M values are needed to resolve regions of the flow where the solution changes very rapidly. When using a large number of points the sparse matrix facilities of MATLAB come into their own. The whole coefficient matrix does not need to be stored and by declaring this as a sparse matrix, only the non-zero entries of the block matrices are calculated. This avoids having to store a very large and sparse matrix which can quickly lead to memory problems.

Increasing the order of the scheme leads to increased bandwidths. This sparsity pattern can be exploited for 2nd, 4th or even 6th finite-differencing with a *direct* solver. However, with spectral methods in two directions, unless the differential operator involved has a special form, it is not immediately possible to utilize the sparse nature of the matrix. Whilst this does not pose any intrinsic difficulties if one is coding in MATLAB, with increased number of points the solution phase can become very memory intensive and requires a lot of processor time. The use of the hybrid approach in our work is motivated in part by the observation that the sparse matrix structure can be exploited to write efficient solvers, which not only work well with MATLAB, but can be coded directly in other languages. MATLAB provides for an excellent environment in which one can test and develop solvers of this type.

The above techniques have been successfully applied to investigate a whole range of different flow problems governed by the Navier-Stokes and related equations. In the first example we consider the onset of instability in the lid-driven cavity flow. MATLAB was used to generate results on coarse grids and do preliminary eigenvalue computations. For very fine grids, the

computations were performed in Fortran 95. The problem is described in detail in Boppana and Gajjar (2010).
The second problem concerns the onset of instability in the flow past a row of circular cylinders. Again the same technqiues have been used but for a more complicated geometry. This problem is described in detail in Boppana and Gajjar (2011).
The third problem we discuss concerns the inviscid transonic flows past thin airfoils. Here the governing equations are nonlinear and of mixed type and the flow can contain shock wave discontinuities for certain parameter values. The full details are given in Pereira and Gajjar (2010). The same methods as described above are used except now type differencing needs to be incorporated to allow for the different flow behaviours in regions of subsonic and supersonic flow. The method is fast and very robust and we are able to compute steady flows with strong shocks. The code was written in MATLAB, using vectorization when possible, and, in order to produce a good interface with the user, we used GUIs (graphical user interfaces) from MATLAB. The result was a fast and accurate code, with the extra bonus of a very good interface with the user, without a lot of effort in terms of programming. The fact that graphical results can be shown immediately, saves us a lot of work, both on the analysis of the results and on its presentation.

2. Instabilities in lid-driven cavities.

To motivate the techniques used in our work we first consider a model elliptic equation of the form

$$a(x,y)\psi_{xx} + b(x,y)\psi_{yy} + c(x,y)\psi_x + e(x,y)\psi_y + f(x,y)\psi = g(x,y) \tag{2}$$

with say Dirichlet boundary conditions in a square domain $0 \leq x, y \leq 1$. It is assumed that the functions a, b, c, e, f, g are smooth functions of x and y.
The discrete solution to the equations will be obtained at a set of grid points on an $(M+1)\times(N+1)$ grid say with the nodes, $x = x_j, \quad j = 0, \ldots, M$ with $x_1 = 0, x_M = 1$, and $y = y_k, \quad 0 \leq k \leq N$ with $y_0 = 0, y_N = 1$, and at these points we set $\psi_{j,k} = \psi(x_j, y_k)$.
We will assume that derivatives in the $x-$ and $y-$ directions may be approximated as

$$\begin{aligned}
\frac{\partial \psi}{\partial x}(x_j, y_k) &= \sum_{q=0}^{M} (\mathbf{D}_x)_{j,q}\psi_{q,k}, \qquad \frac{\partial^2 \psi}{\partial x^2}(x_j, y_k) = \sum_{q=0}^{M} (\mathbf{D}_{xx})_{j,q}\psi_{q,k}, \\
\frac{\partial \psi}{\partial y}(x_j, y_k) &= \sum_{q=0}^{M} (\mathbf{D}_y)_{k,q}\psi_{j,q}, \qquad \frac{\partial^2 \psi}{\partial y^2}(x_j, y_k) = \sum_{q=0}^{M} (\mathbf{D}_{yy})_{k,q}\psi_{j,q}.
\end{aligned} \tag{3}$$

Given the type and order of discretisation, the elements of the matrices $\mathbf{D}_x, \mathbf{D}_{xx}, \mathbf{D}_y, \mathbf{D}_{yy}$ are known. For example with second-order central finite differences, at an interior point of a uniform grid in x with $\Delta_x = x_j - x_{j-1}$, we have

$$(\mathbf{D}_x)_{j,j-1} = -\frac{1}{2\Delta_x}, \quad (\mathbf{D}_x)_{j,j+1} = \frac{1}{2\Delta_x},$$

$$(\mathbf{D}_{xx})_{j,j-1} = \frac{1}{\Delta_x^2}, \quad (\mathbf{D}_{xx})_{j,j+1} = \frac{1}{\Delta_x^2}, \quad (\mathbf{D}_{xx})_{j,j} = -\frac{2}{\Delta_x^2}, \quad 1 \leq j \leq M-1,$$

and zero otherwise. If we take Chebychev collocation in the $y-$direction, the $\mathbf{D}_y, \mathbf{D}_{yy}$ are the Chebychev differentiation matrices and are given in a number of places, see for example

Weideman and Reddy (2003) or Trefethen (2000), where MATLAB code to generate the matrices is given.
Discretization of the equation 2 leads to the set of equations

$$a_{j,k}\sum_{q=0}^{M}(\mathbf{D}_{xx})_{j,q}\psi_{q,k}+b_{j,k}\sum_{q=0}^{M}(\mathbf{D}_{yy})_{k,q}\psi_{j,q}+c_{j,k}\sum_{q=0}^{M}(\mathbf{D}_{x})_{j,q}\psi_{q,k}+e_{j,k}\sum_{q=0}^{M}(\mathbf{D}_{y})_{k,q}\psi_{j,q}+f_{j,k}\psi_{j,k}=g_{j,k}, \tag{4}$$

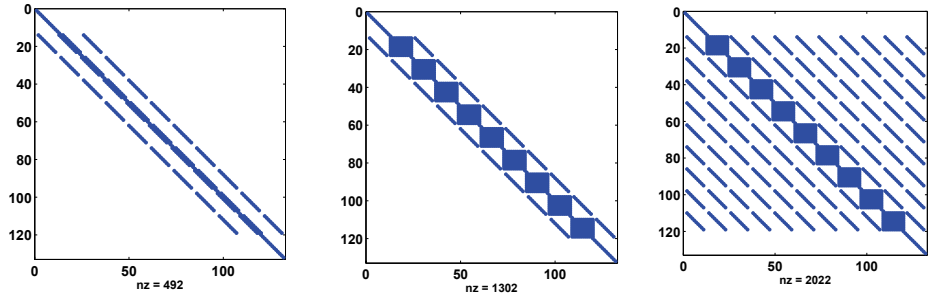
at the interior points $1 \leq j \leq M-1, \quad 1 \leq k \leq N-1$. At the boundaries we have $\psi = 0$.
The discrete set (4) can be combined into a compact form as

$$\begin{aligned}&\left[\mathrm{DIAG}(\mathbf{a})(\mathbf{I}_{N+1}\oplus\mathbf{D}_{xx})+\mathrm{DIAG}(\mathbf{c})(\mathbf{I}_{N+1}\oplus\mathbf{D}_{x})\right]\Psi+\\&\left[\mathrm{DIAG}(\mathbf{b})(\mathbf{D}_{yy}\oplus\mathbf{I}_{M+1})+\mathrm{DIAG}(\mathbf{e})(\mathbf{D}_{y}\oplus\mathbf{I}_{M+1})\right]\Psi+\mathrm{DIAG}(\mathbf{f})\Psi=\mathbf{g}\ ,\end{aligned} \tag{5}$$

where Ψ denotes the vector of unknowns, $\mathbf{A}\oplus\mathbf{B}$ is the kronecker tensor product of two matrices $\mathbf{A}, \mathbf{B}$ (which in MATLAB is represented by the **kron(A,B)** operator), and DIAG($\mathbf{v}$) is as in MATLAB the diagonal matrix with entries given by the vector $\mathbf{v}$, and $\mathbf{I}_{M+1}$ is the identity matrix of order $M+1$. Certain rows of the matrix operator in (5) are also modified when the boundary conditions are incorporated.
The linear sytem may be represented as

$$\mathbf{S}\Psi=\mathbf{R}. \tag{6}$$

Once the coefficient matrix **S** and the right hand sides have been computed, the solution just involves the use of the \ operator with $\Psi = \mathbf{S}\backslash\mathbf{R}$. From the user perspective other than declaring that the matrices involved are sparse matrices, no additional special treatment is required to obtain the solution of the linear systems.
Given the discretization matrices, the above system is easy to code in MATLAB. For certain discretizations however, the linear systems outlined above can be huge and highly sparse. The bandwidth of the coefficient matrix increases with increasing order of differences used, and with spectral methods in two directions, the coefficient matrix has the sparsity pattern similar to that in figure 1(c), obtained using the **spy** function in MATLAB. On the other hand by taking second-order finite differences in the $x-$direction and spectral collocation in the $y-$direction, with a particular ordering of grid points, the matrices can be written in block tridiagonal form as shown in figure 1(a),1(b), or with fourth order finite-differencing in x the linear system is block pentadiagonal. With 2nd, 4th or even 6th finite-differencing the linear systems can be solved with a *direct* solver but with spectral methods in two directions, unless the differential operator involved has a special form, it is not immediately possible to utilize the sparse nature of the matrix. Whilst this does not pose any intrinsic difficulties if one is coding in MATLAB, with increased number of points the solution phase can become very memory intensive and requires a lot of processor time. The use of the hybrid approach in our work is motivated in part by the observation that the sparse matrix structure can be exploited to write efficient solvers, which not only work well with MATLAB, but can be coded directly in other languages. MATLAB provides for an excellent environment in which one can test and develop solvers of this type. In our work we have written our own direct block solvers as well as making direct use of the \ operator with a sparse matrix.
In the MATLAB codes when constructing the coefficient matrices, we use the functions **speye** or **spalloc** for initially creating the sparse matrices. After this only the non-zero elements of the matrices are assigned.



(a) (b) (c)

Fig. 1. Sparsity patterns for model problem with (a) second order finite-differences in both x and y with $N = 10, M = 11$, (b) second-order finite-difference in y and chebychev collocation in x with $N = 10, M = 11$, (b) chebychev collocation in both x and y with $N = 10, M = 11$.

2.1 Lid-driven cavity flow.

In this section we discuss the steps needed to go from the continuous model described by the set of coupled nonlinear partial differential equations, the Navier-Stokes equations, to solution using MATLAB. To focus attention we will consider the classic problem of flow in a lid-driven cavity which has been extensively studied in the literature. The governing equations for the problem are

$$\left.\begin{aligned} \nabla^2\psi &= \omega, \\ \text{and} \quad \psi_y\omega_x - \psi_x\omega_y &= \tfrac{1}{Re}\nabla^2\omega. \end{aligned}\right\} \tag{7}$$

Here Re is the Reynolds number defined as $\frac{Uw}{\nu}$, where U is the velocity of the lid, ν is the kinematic viscosity of the fluid, and w is the width of the cavity that are used to non-dimensionalize the velocity and length-scale variables respectively. The boundary conditions (see figure 2(a)) are given by

$$\left.\begin{aligned} \psi = 0 \quad \& \quad \psi_x = 0 \quad \text{for} \quad x = 0, \quad 0 \le y \le A, \\ \psi = 0 \quad \& \quad \psi_x = 0 \quad \text{for} \quad x = 1, \quad 0 \le y \le A, \\ \psi = 0 \quad \& \quad \psi_y = 0 \quad \text{for} \quad y = 0, \quad 0 \le x \le 1, \\ \psi = 0 \quad \& \quad \psi_y = 1 \quad \text{for} \quad y = A, \quad 0 \le x \le 1, \end{aligned}\right\} \tag{8}$$

where A is the aspect ratio of the cavity. The discrete solution to the equations will be obtained at a set of grid points on an $(M+1)\times(N+1)$ grid say with the nodes, $x = x_j, \quad j = 0, \ldots, M$ with $x_1 = 0, x_M = 1$, and $y = y_k, \quad 0 \le k \le N$ with $y_0 = 1, y_N = 1$, and at these points we set $\psi_{j,k} = \psi(x_j, y_k), \quad \omega_{j,k} = \omega(x_j, y_k)$.
We will assume that derivatives in the $x-$ and $y-$ directions may be approximated as in (3). The discretization leads to a set of nonlinear algebraic equations for the unknowns $\psi_{j,k}, \omega_{j,k}$, $0 \le j \le M, \quad 0 \le k \le N$. To solve these we use Newton linearization and this will lead to linear system of equations which are solved using MATLAB. Linearization of the nonlinear equations is performed by setting $\psi_{j,k} = \bar{\psi}_{j,k} + G_{j,k}, \quad \omega_{j,k} = \bar{\omega}_{j,k} + H_{j,k}$, the barred quantities representing a current iterate and $G_{j,k}, H_{j,k}$ begin small corrections, and substituting into the

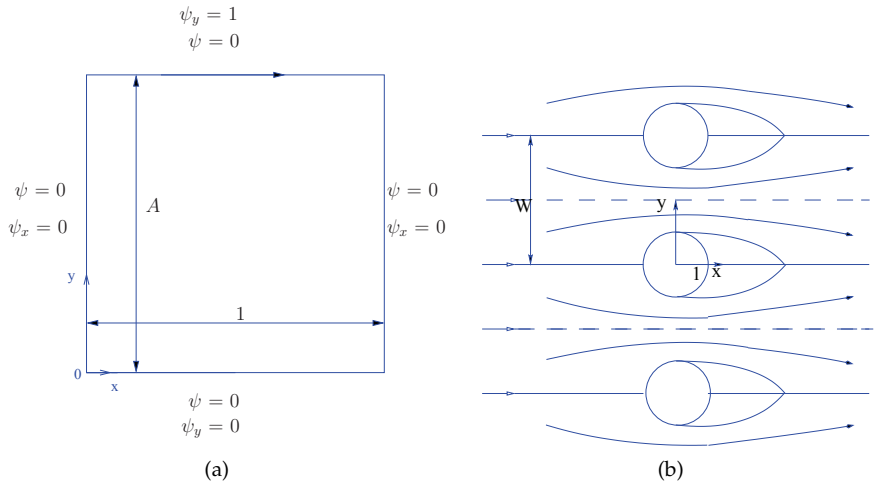


Fig. 2. Sketch of (a) the lid–driven cavity with boundary conditions, and (b) flow past a row of circular cylinders.

nonlinear equations and neglecting second order small terms. This leads to

$$(\mathbf{I}_{N+1} \oplus \mathbf{D}_{xx})\mathbf{G} + (\mathbf{D}_{yy} \oplus \mathbf{I}_{M+1})\mathbf{G} - \mathrm{DIAG}(\mathbf{H}) = -(\mathbf{I}_{N+1} \oplus \mathbf{D}_{xx})\bar{\psi} - (\mathbf{D}_{yy} \oplus \mathbf{I}_{M+1})\bar{\psi} + \mathrm{DIAG}(\bar{\omega}),$$

$$\begin{aligned} &\mathrm{DIAG}(\bar{\psi}_y)(\mathbf{I}_{N+1} \oplus \mathbf{D}_x)\mathbf{H} + \mathrm{DIAG}(\bar{\omega}_x)(\mathbf{D}_y \oplus \mathbf{I}_{M+1})\mathbf{G} \\ &- \mathrm{DIAG}(\bar{\psi}_x)(\mathbf{D}_y \oplus \mathbf{I}_{M+1})\mathbf{H} - \mathrm{DIAG}(\bar{\omega}_y)(\mathbf{I}_{N+1} \oplus \mathbf{D}_x)\mathbf{G} \\ &- \frac{1}{Re}\left[(\mathbf{I}_{N+1} \oplus \mathbf{D}_{xx})\mathbf{H} + (\mathbf{D}_{yy} \oplus \mathbf{I}_{M+1})\mathbf{H}\right] = -\mathrm{DIAG}(\bar{\psi}_y)(\mathbf{I}_{N+1} \oplus \mathbf{D}_x)\bar{\omega} \\ &+ \mathrm{DIAG}(\bar{\psi}_x)(\mathbf{D}_y \oplus \mathbf{I}_{M+1})\bar{\omega} + \frac{1}{Re}\left[(\mathbf{I}_{N+1} \oplus \mathbf{D}_{xx})\bar{\omega} + (\mathbf{D}_{yy} \oplus \mathbf{I}_{M+1})\bar{\omega})\right]. \end{aligned}$$

Here $\mathbf{G}, \mathbf{H}$ are the vector of unknown corrections.
Depending on the discretization, the above can be coded directly in MATLAB by constructing the coefficient matrix multiplying the vector of unknowns $(\boldsymbol{G}, \boldsymbol{H})^T$. Note that the size of the coefficient matrix is $2(N+1)(M+1) \times 2(N+1)(M+1)$ and even for modest N, M the above procedure leads to very large matrices and is not efficient. The approach we have adopted is to make use of the sparsity patterns for particular types of discretizations.

2.2 Use of Matlab for the solution of the discrete system

For the case when we use second order finite-differeces in the $x-$direction and chebychev collocation in the $y-$direction, the linear system may be written as

$$\mathbf{S}\boldsymbol{\Phi} = (\mathbf{S}_0, \mathbf{S}_1, \ldots, \mathbf{S}_M)^T = \mathbf{r},$$

where

$$\mathbf{S}_p = \mathbf{A}_p\boldsymbol{\Phi}_{p-1} + \mathbf{B}_p\boldsymbol{\Phi}_p + \mathbf{C}_p\boldsymbol{\Phi}_{p+1}, \quad 0 \le p \le M \tag{9}$$

$$\boldsymbol{\Phi}_p = (G_{p0}, G_{p1}, \ldots, G_{pN}, H_{p0}, H_{p1}, \ldots H_{pN})^T, \quad \boldsymbol{\Phi} = (\boldsymbol{\Phi}_0, \boldsymbol{\Phi}_1, \ldots, \boldsymbol{\Phi}_M)^T,$$

with $\mathbf{A}_0 = \mathbf{C}_M = \mathbf{0}$. This represents a particular ordering of unknowns and gives rise to the block tridiagonal system in (9). With 4th order finite-differencing in x the linear system is block pentadiagonal.

The coefficient matrices $\mathbf{A}_p, \mathbf{B}_p, \mathbf{C}_p$ can be extracted from the discrete equations above and

$$\mathbf{A}_p = \begin{pmatrix} \frac{1}{\Delta_x^2}\mathbf{I}_{N+1} & \mathbf{O} \\ \frac{1}{2\Delta_x}\mathrm{DIAG}(\mathbf{D}_y\bar{\boldsymbol{\omega}}_p) & -\frac{1}{2\Delta_x}\mathrm{DIAG}(\mathbf{U}_p) - \frac{1}{Re\Delta_x^2}\mathbf{I}_{N+1} \end{pmatrix},$$

$$\mathbf{B}_p = \begin{pmatrix} \mathbf{D}_{yy} - \frac{2}{\Delta_x^2}\mathbf{I}_{N+1} & -\mathbf{I}_{N+1} \\ \mathrm{DIAG}(\boldsymbol{\Omega}_{xp})\mathbf{D}_y & \mathrm{DIAG}(\mathbf{V}_p)\mathbf{D}_y + \frac{1}{Re}\left[\frac{2}{\Delta_x^2}\mathbf{I}_{N+1} - \mathbf{D}_{yy}\right] \end{pmatrix},$$

$$\mathbf{C}_p = \begin{pmatrix} \frac{1}{\Delta_x^2}\mathbf{I}_{N+1} & \mathbf{O} \\ \frac{1}{2\Delta_x}\mathrm{DIAG}(\mathbf{D}_y\bar{\boldsymbol{\omega}}_p) & \frac{1}{2\Delta_x}\mathrm{DIAG}(\mathbf{U}_p) - \frac{1}{Re\Delta_x^2}\mathbf{I}_{N+1} \end{pmatrix},$$

with

$$\mathbf{U}_p = \mathbf{D}_y\bar{\boldsymbol{\psi}}_p, \quad \boldsymbol{\Omega}_{xp} = \frac{\bar{\boldsymbol{\omega}}_{p+1} - \bar{\boldsymbol{\omega}}_{p-1}}{2\Delta_x}, \quad \mathbf{V}_p = -\frac{\bar{\boldsymbol{\psi}}_{p+1} - \bar{\boldsymbol{\psi}}_{p-1}}{2\Delta_x}.$$

The above excludes the boundary conditions, but these just alter certain rows of the matrices. In MATLAB the individual entries of the block matrices are easily computed and the **S** matrix is updated via

$$\mathbf{S}(1+2p(N+1) : 2(p+1)(N+1), 1+2(p-1)(N+1) : 2p(N+1)) = \mathbf{A}_p,$$
$$\mathbf{S}(1+2p(N+1) : 2(p+1)(N+1), 1+2p(N+1) : 2(p+1)(N+1)) = \mathbf{B}_p,$$
$$\mathbf{S}(1+2p(N+1) : 2(p+1)(N+1), 1+2(p+1)(N+1) : 2(p+2)(N+1)) = \mathbf{C}_P,$$

for $1 \leq p \leq M-1$.

2.3 Results for lid-driven cavity

The method described above was used to compute the flow in a lid-driven cavity. The same techniques used were adapted to firstly compute the steady flow, and then to investigate the instability of the flow via simulations as well as solving the linear eigenvalue problem assuming normal mode disturbances proportional to $e^{\lambda t}$. Further details of the techniques may be found in Boppana and Gajjar (2010). MATLAB was also used in the eigenvalue analysis. In fact the eigenvalue problem required to be solved takes the form

$$S\boldsymbol{\Phi} = \lambda\mathbf{T},$$

which is a generalised eigenvalue problem. The matrix **S** is the same as the Jacobian matrix of the linear system after Newton linearization, and **T** is a singular diagonal matrix. The eigenvalue problem is solved in MATLAB using the **eig** function. In other related problems the routine **sptarn** available in the PDE toolbox was used for the solution of the generalised eigenvalue problem.

In figures 3, 4 we have shown the real and imaginary parts of eigenfunctions for the disturbance streamfunction ψ for aspect ratios of $A = 1$ and $A = 2$ at the onset of instability, obtained from a solution of the eigenvalue problem. The advantage of working with MATLAB is that such plots can be generated at the same time as the computation is in progress.

More extensive results and details are documented in Boppana and Gajjar (2010).

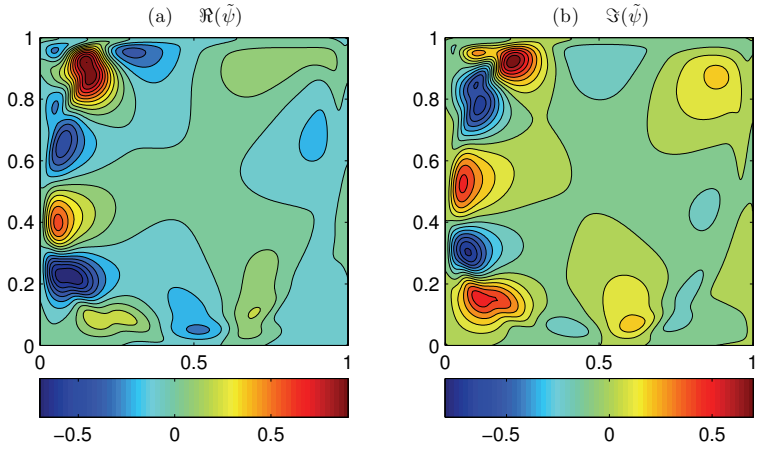


Fig. 3. Eigenfunctions of the streamfunction for lid-driven cavity at a critical Reynolds number of $Re = 8026.7$ and aspect ratio $A = 1$.

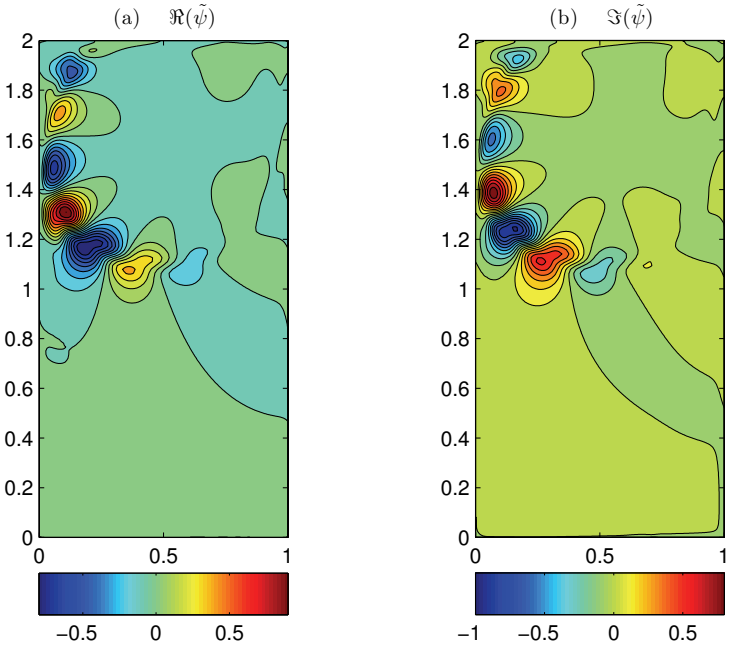
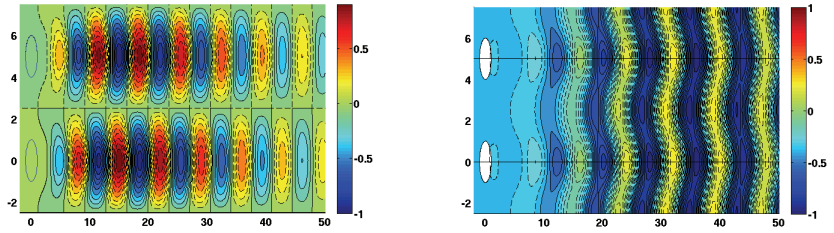


Fig. 4. Eigenfunctions of the streamfunction for lid-driven cavity at a critical Reynolds number of $Re = 5861$ and aspect ratio $A = 2$.

3. Flow past circular cylinders

The techniques described above have also been used to solve for the uniform flow past a row of circular cylinders, see figure 2(b). Here the Navier-Stokes equations are solved with boundary conditions of no slip on the cylinder surfaces and uniform flow far upstream. The two important parameters are the gap-width between the cylinder centres W (normalised with respect to cylinder radius) and the Reynolds number Re. Of particular interest is to ascertain when the flow first becomes unstable and the mode of instability. Experiments, see for example Mizushima & Ino (2008), for the flow past two such cylinders show a complicated dynamics as the parameters are varied. For large Reynolds numbers, and for large gap widths the flow is like that past an isolated cylinder and the observed shedding frequencies also similar. For $Re >> 1$ and intermediate gap widths (of 0.5 to 1 cylinder diameters), the flow can become deflected to one side and becomes asymmetric. For small gap widths, the flow is similar to that past a single bluff body. At low Reynolds numbers the flow dynamics is also complicated and with conflicting results.
In our work, MATLAB was used to first compute the base flows, and then for studies of the onset of instability, by solving the generalised eigenvalue problem. Full details of the numerical methods and results can be found in Boppana and Gajjar (2011). In figures 5, 6 we have shown the results for the streamfunction eigenfunctions. Modes which are symmetric and asymmetric with respect to the cylinder centreline, can be observed and it found that the critical Reynolds numbers for the onset of instabilties are very similar for both modes. However the symmetric modes as shown in figures 5(b), 6(b) for instance, have much lower critical Strohal frequencies as compared to asymmetric mode. The latter is the one which tends to that for the flow past an isolated cylinder as the gap width increases.

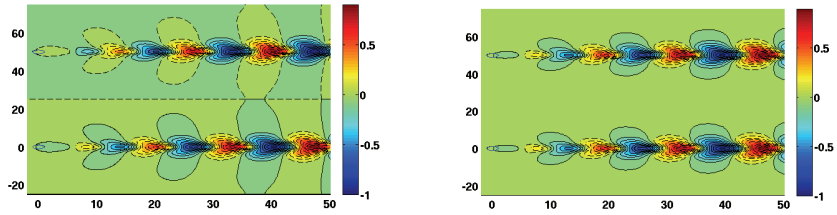


(a) (b)

Fig. 5. Perturbation eigenfunctions the flow past a circular cylinder for a gap width $W = 5$ with (a) anti-phase oscillatory mode, (b) in phase oscillatory mode.

4. Transonic flows past airfoils.

The study of transonic flows is motivated in part by the observation that many modern airplane carriers operate most efficiently when cruising at speeds which fall in the transonic range, that is close to the speed of sound. Mathematically the study of transonic flows is fascinating because the governing equations are nonlinear and of mixed type. The methods described above work have been applied to partial differential equations of mixed type containing shocks.



(a) (b)

Fig. 6. Perturbation eigenfunctions the flow past a circular cylinder for a gap width $W = 50$ with (a) anti-phase oscillatory mode, (b) in phase oscillatory mode.

4.1 The mathematical model

In this subsection we describe briefly a numerical method to deal with transonic flows past thin airfoils, based on the Kárman-Guderley equations. This method is discussed in detail in Pereira and Gajjar (2010). However the aim of this section in not to repeat once again the same work, but rather to focus on the use of the graphical interface which is extremely easy to program in MATLAB. First we will give an overview of how the mathematical model was built, and then we will explain how to use MATLAB graphics user interfaces (GUIs) in the context of our problem.

The governing equation may be obtained using asymptotic methods as described in Cole and Cook (1986). The starting point to our model was the full potential equation:

$$(a^2 - U^2)\Phi_{xx} - 2UV\Phi_{xy} + (a^2 - V^2)\Phi_{yy} = 0, \tag{10}$$

$$\frac{1}{2}(\Phi_x^2 + \Phi_y^2) + \frac{a^2}{\gamma - 1} = \frac{U_\infty}{2} + \frac{a_\infty^2}{\gamma - 1}, \tag{11}$$

where Φ represents the velocity potential, a is the local speed of sound, U_∞ is the velocity in the far field, a_∞ is the speed of sound in the far field, and $M_\infty = U_\infty / a_\infty$ is the free-stream Mach number. The velocity components (U, V) are defined as follows,

$$U = \Phi_x, \quad V = \Phi_y.$$

The density ρ and pressure p can be determined via the relationships,

$$\rho^{\gamma-1} = M_\infty^2 a^2; \quad p = \frac{\rho^\gamma}{\gamma M_\infty^2},$$

where γ is the ratio of specific heats.

In order to define the boundary conditions we assume that the flow is uniform in the far field and that the flow is tangent to the airfoil on its surface. To construct this theory we also assume that we have a thin aerofoil with width ($\delta \to 0$) and that the air flow speed is close to sonic so $M_\infty^2 = 1 - k\mu(\delta), \quad \mu(\delta) \to 0$ where k is the transonic similarity parameter and μ is a function of the airfoil width (δ). The oncoming flow is assumed to be aligned with the x-direction.

In order to define the boundary conditions we assume that the flow is uniform in the far field and that the flow is tangent to the airfoil on its surface. To construct this theory we also assume that we have a thin aerofoil with width ($\delta \to 0$) and that the air flow speed is close to sonic so $M_\infty^2 = 1 - k\mu(\delta), \quad \mu(\delta) \to 0$ where k is the transonic similarity parameter and μ is a function

of the airfoil width (δ). The oncoming flow is assumed to be aligned with the x-direction. The airfoil is defined by,

$$y = \delta F(x).$$

Introducing non dimensional variables,

$$U = uU_\infty, V = vU_\infty,$$

then the full potential equation becomes,

$$(\frac{a^2}{U_\infty^2} - u^2)\Phi_{xx} - 2uv\Phi_{xy} + (\frac{a^2}{U_\infty^2} - v^2)\Phi_{yy} = 0.$$

The expansion for Φ is described in Cole and Cook (1986) and is given by,

$$\Phi(x, y, M_\infty, \delta) = U_\infty(x + \epsilon(\delta)\phi(x, y, k) + ...).$$

It is well known that as $M_\infty \to 1$, the perturbations extend in the y direction significantly. Because of this, stretched coordinates were used and the governing equations and boundary conditions reduce to,

$$\phi_{xx}(k - \phi_x(\gamma + 1)) + \phi_{YY} = 0, \tag{12}$$

$$\phi_x = \phi_Y = 0, \qquad x^2 + Y^2 \to \infty, \tag{13}$$

$$\phi_Y(Y = 0) = F'(x), \tag{14}$$

where (12) is the so called Kárman-Guderley equation. Equation (12) in conservative form is written as follows,

$$\frac{\partial}{\partial x}(k\phi_x - \frac{\gamma + 1}{2}\phi_x^2) + \phi_{YY} = 0, \tag{15}$$

and, if we denote,

$$\psi = kx - (\gamma + 1)\phi \tag{16}$$

then, we may rewrite (15) as,

$$(\frac{\psi_x^2}{2})_x + \psi_{YY} = 0. \tag{17}$$

The boundary conditions become,

$$\psi_x(x^2 + Y^2 \to \infty) = k, \quad \psi_Y(Y = 0) = -(\gamma + 1)F'(x). \tag{18}$$

When considering the non symmetric case, one has to introduce a new boundary condition - the Kutta condition. The Kutta condition is used at the trailing edge to ensure that the jump obtained in the integration of ψ_x along the lower and upper surfaces at the tail is zero.
Next we give a brief overview of the numerical method used to solve the above problem. Ee used finite differences for the derivatives in the x direction and a Chebyshev collocation method to describe the derivatives in the Y direction. As described in Cole and Cook (1986) each point of the domain may be either subsonic, sonic, supersonic or a shock point. Let,

$$P = (\frac{\psi_x^2}{2}),$$

then equation (17) becomes,

$$P_x + \psi_{YY} = 0. \tag{19}$$

Let $(\psi_x)_{i+1/2,j}$ represent the derivative with respect to x of ψ at the point $(x_{i+1/2}, Y_j)$, and let $h_i = x_i - x_{i-1}$. Using central differences for the x derivatives we may write (19) as,

$$\frac{(\psi_x)^2_{i+1/2,j} - (\psi_x)^2_{i-1/2,j}}{h_i + h_{i+1}} + (\psi_{YY})_{i,j} = 0,$$

or,

$$\frac{1}{h_i + h_{i+1}}((\psi_x)_{i+1/2,j} - (\psi_x)_{i-1/2,j})((\psi_x)_{i+1/2,j} + (\psi_x)_{i-1/2,j}) + (\psi_{YY})_{i,j} = 0. \tag{20}$$

As discussed in Cole and Cook (1986) if we have a subsonic point, the equation is elliptic and central differences should be used to calculate both $(\psi_x)_{i+1/2,j}$ and $(\psi_x)_{i-1/2,j}$. If we have a supersonic point, then equation (20) becomes hyperbolic and backwards differences should be used to calculate both $(\psi_x)_{i+1/2,j}$ and $(\psi_x)_{i-1/2,j}$. In the first case we can rewrite (20) as,

$$p_{i,j} + (\psi_{YY})_{i,j} = 0. \tag{21}$$

In the second case, we can rewrite (20) as,

$$p_{i-1,j} + (\psi_{YY})_{i,j} = 0. \tag{22}$$

where,

$$p_{i,j} = \frac{A_{i,j}}{h_i + h_{i+1}}\psi^c_x,$$

$$A_{i,j} = \frac{\psi_{i+1,j} - \psi_{i,j}}{h_{i+1}} - \frac{\psi_{i,j} - \psi_{i-1,j}}{h_i}. \tag{23}$$

and,

$$\psi^c_x = \frac{\psi_{i+1,j} - \psi_{i,j}}{h_{i+1}} + \frac{\psi_{i,j} - \psi_{i-1,j}}{h_i}. \tag{24}$$

Two other cases are considered, the case when the flow accelerates from subsonic to supersonic in which case we define a sonic point, and the opposite, when the flow decelerates form supersonic to subsonic in which case we call it a shock point. To deal with these cases we use artificial viscosity ($\mu_{i,j}$) Murman and Cole (1971), and equations (21) and (22) can be condensed as,

$$p_{i,j}(1 - \mu_{i,j}) + p_{i-1,j}\mu_{i-1,j} + \psi_{YY} = 0$$

where the values of $\mu_{i,j}$ and $\mu_{i-1,j}$ are taken from the following table:

TYPE OF POINT	$\psi_x^{central}$	ψ_x^{backWs}	$\mu_{i-1,j}$	$\mu_{i,j}$
ELLIPTIC	> 0	> 0	0	0
HYPERBOLIC	< 0	< 0	1	1
SONIC	< 0	> 0	0	1
SHOCK	> 0	< 0	1	0

Note that a shock point can be seen as an addition of both elliptic and hyperbolic x difference operators.
In the Y direction, the physical domain was first truncated to y_∞ and mapped into the Chebyshev space, as in Canuto et. all (1998),

$$Y \in [0, y_\infty] \rightarrow z \in [-1, 1]$$

where,

$$z_j = \cos(\frac{j\pi}{N}), j = 0, 1, ..., N$$

and,

$$Y_j = y_\infty(\frac{z_j + 1}{2}).$$

First and second derivatives in the Y directions were calculated as described earlier.
After applying the above discretizations we obtain a set of coupled nonlinear algebraic equations. These are linearized using Newton-Raphson linearization by setting

$$\psi_{i,j} = \overline{\psi_{i,j}} + G_{i,j},$$

where $\overline{\psi_{i,j}}$ represents the value of $\psi_{i,j}$ in a previous iteration and $G_{i,j}$ represents the update for $\psi_{i,j}$. This results in a linear system of equations for the $G_{i,j}$ of the form

$$A_{i,j}G_{i-2,j} + B_{i,j}G_{i-1,j} + C_{i,j}G_{i,j} + H_{i,j}G_{i+1,j} + E_{i,j}G_{i+2,j} = F_{i,j}. \quad (25)$$

Details on how to calculate the coefficients $A_{i,j}, B_{i,j}..., F_{i,j}$ can be found in Pereira and Gajjar (2010). The block pentadiagonal system of equations was solved directly using routines described in Korolev et all. (2002).

4.2 The use of GUIs

In this subsection we will focus on how to generate a good graphics interface using MATLAB GUIs in the context of the problem described in the previous subsection. In order to obtain a graphics interface to our programme, the first thing we have to do is to type **guide** in MATLAB's command window. The result is that MATLAB opens a window that has a graphics interface working environment(GIWE). The next thing to do, is to save it as **name.fig**. This action has as a result not only of saving the work done so far, but also to generate a file **name.m**, that has the MATLAB corresponding instructions.
Suppose next that we want to introduce a title on top of the graphics interface window. We select the *"**Static text**"* button in the GIWE. With the mouse one selects the location and the size of the text to input. Then after double clicking on it a new window appears, where one can choose options for the static text we want to introduce.

Another important feature on the design of an interface, is how to read data from this interface. To do so, one option is to select the *"**Edit text**"* button in the GIWE. With the mouse one selects the location and the size of the text to be read. Then, after a double click a new window appears where one can chose options for the text to be read from the keyboard. It is possible to choose a default text that can be attributed to a certain variable in the MATLAB code. If we fill in the field **"Tag"** with a name, that can be used to attribute the edited text to a variable. Suppose we fill the field "Tag" with **A1**. By doing this and saving it, one automatically obtains

in the **.m file** two new functions **A1CreateFcn** and **A1Callback**. The first function executes during object creation, after setting all properties. The second function allows for the edited text to be attributed to a variable when the **enter key** is pressed. Say we want to attribute the edited text to the variable **"x"**, all you have to do is to write in the **A1Callback** function the following instruction:

x=str2double(get(handles.A1,'String'));

Here, the text read via the keyboard, once the **enter key** is pressed, is converted to double and attributed to a variable **"x"**. This feature is very important for the user to be able to set the values of any variables to be used in the program.

In many codes, it is also very important to define a **push button**. This can be used to set up certain actions once it is pushed. An example is when one wants to read data from the keyboard, before starting a simulation. To implement this idea, we click the **push button** in the GIWE. Then one selects the size and place where to put it in the GIWE. Next, if one double clicks in it, a new window opens and the options for this **push button** are defined. Once this is done and it is saved, a new function is created in the **.m file** called **pushbuttonCallback**. All the actions that are to happen once the button is pushed are to appear in this function. For instance, one can attribute a set of values to a set of variables. This is done exactly in the same way as before. The difference is that this can be done to a set of many variables at once. The next instructions we put in this function are the ones that compose the **main code** of our programme.

Suppose that in the course of the simulation, we want to show a graphic object. This is done by choosing the **Axes button** in the GIWE. Then one selects the size and place where to put the graphic in the GIWE. Next, a double click on it and a new window opens and the options for this **Axes button** are defined. If we fill in the field **"Tag"** with a name, say **"graphics1"**, in order to plot a graphic $(x, f(x))$ in that window, we would write:

axes(handles.graphics1);
plot(x,f)

Using these features, we were able to define a useful graphics interface for the program to solve the transonic flow past an airfoil using the mathematical model presented in the previous section. The interface was built for the NACA0012 airfoil and is shown in figure 7.
The parameters considered were: Angle of attack (α), Mach number(M_∞), number of points in the x direction over the wing (nx), number of points in the Y direction (ny), maximum value for Y ($Ymax$), and relaxation factor (w).
In the figures 8, 9, we present the results obtained by our code for two classic examples extensively studied in the literature. The first example is for $M_\infty = 0.75$, $\alpha = 2.0$, $nx = 40$, $ny = 40$, $Ymax = 1.25$, $w = 0.5$.
The second example is for $M_\infty = 0.8$, $\alpha = 1.25$, $nx = 40$, $ny = 40$, $Ymax = 1.25$, $w = 0.6$.
The results of both examples agree with the literature, see for example Cole and Cook (1986) and Camilo (2003). As we can see, by using GUIs, we obtained a user friendly professional looking graphics interface, which allowed any other user to perform simulations and input their parameters for the run. The other users of the software did not need to be familiar with the underlying code and equations.

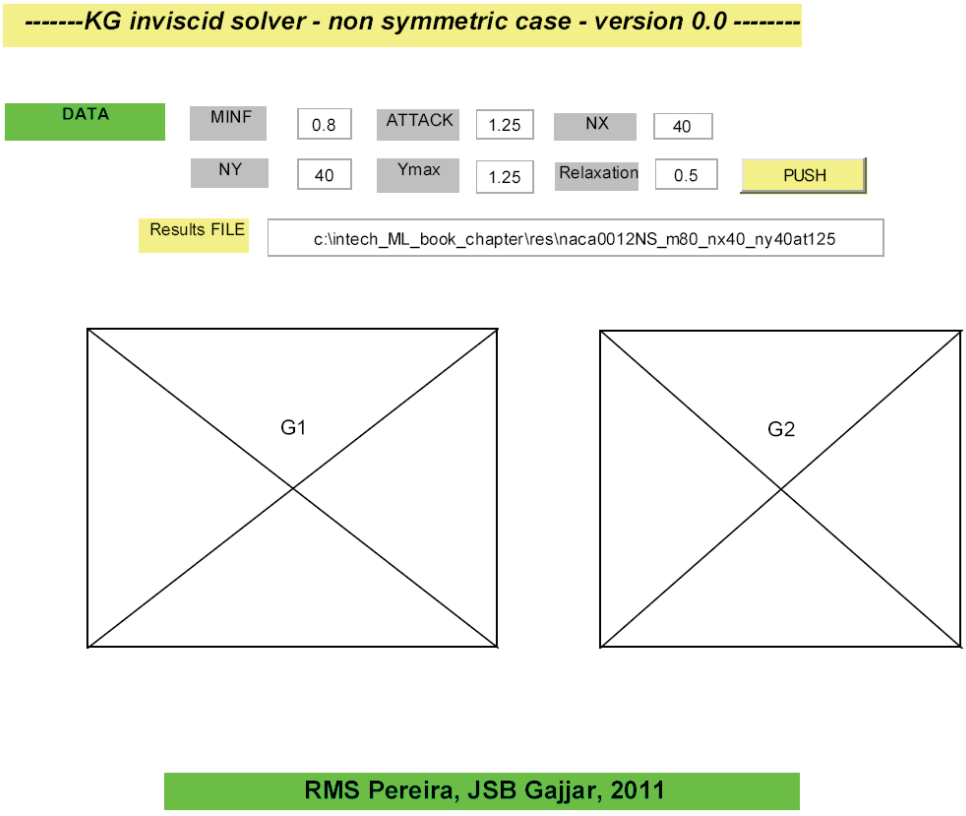


Fig. 7. The ".fig" file built for the example considered.

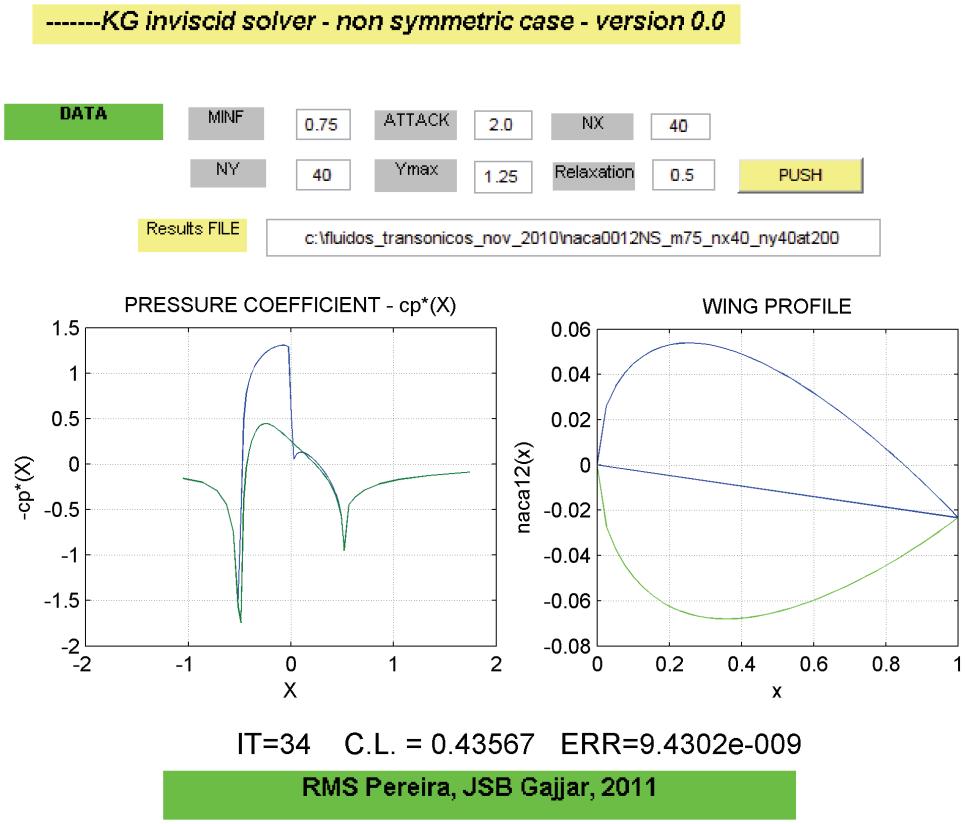


Fig. 8. Example of results for KG code for Transonic flows where $M_{inf} = 0.75; \alpha = 2.0; Nx = 40, Ny = 40, relaxation = 0.5$.

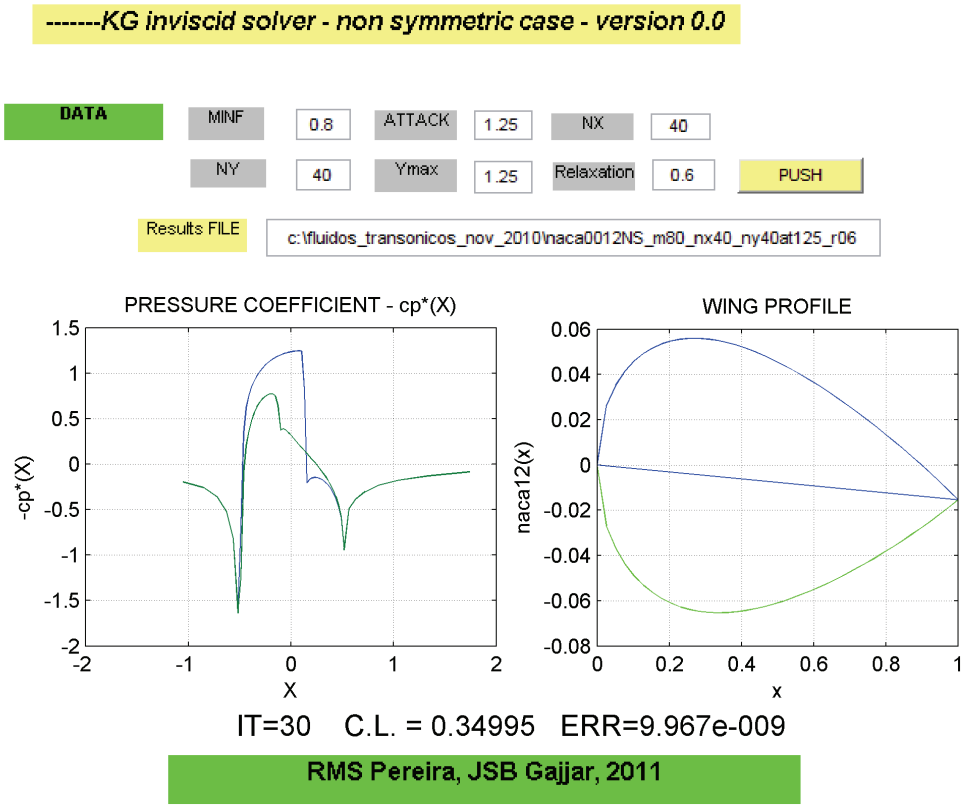


Fig. 9. Example of results for KG code for Transonic flows where $M_{inf} = 0.8; \alpha = 1.25; Nx = 40, Ny = 40, relaxation = 0.6$.

5. Conclusions

- The environment of MATLAB is easy to work, the syntax is very simple and intuitive, it has powerful toolboxes to treat many different problems in engineering, and it allows us to produce fantastic graphics as the program runs.
- A MATLAB code can be very compact, allowing anyone to have "the big picture" of any code without having to look at all its details.
- Another great advantage of Matlab is that, if the code is written in a vectorized form, the code can run much faster than if it was written in the traditional form (*'a la C/fortran'*).
- The fact that MATLAB allows us to use a powerful toolbox for sparse matrices, is also a great advantage since, many traditional linear algebra operations can be highly improved, allowing the codes to run much faster than it would run with the traditional linear algebra functions.
- The main drawback with the use of MATLAB is that when the number of points used in the discretization grows, which is necessary for finely resolved computations, the execution time and memory requirements can grow substantially. On multicore machines, the use of the *parallel toolbox* is recommended and this is supposed to provide superior performance making use of parallel linear algebra routines with minimal code changes. However we have not been able to test this with our codes.

6. References

Boppana, V.B.L. and Gajjar, J.S.B. (2010). *Global flow instability in a lid-driven cavity.* Int J. for Num Methods in Fluids, 62, 827-853.

Boppana, V.B.L. and Gajjar, J.S.B. (2011). *Onset of instability in the flow past a circular cylinder cascade.* J. Fluid Mech. 668, 303-334.

Camilo E. (2003). *Solução numérica das equações de Euler para representação do escoamento transónico de aerofólios.*M.Sc thesis, University of São Paulo, Brazil.

Canuto, C., Hussaini, M. Y., Quarteroni, A. and Zhang, T.A., (1998). *Spectral Methods in Fluid Mechanics.* Springer series in Comp. Phys., Springer Verlag.

Cole, J. D. and Cook, L. P., (1986). *Transonic Aerodynamics*, Elsevier Science Publishers B.V. .

Korolev G.L, Gajjar J.S.B., and Ruban A.I., (2002). *Once again on the supersonic flow separation near a corner*. J. Fluid Mech., 463, 173-199.

Mizushima, J. and Ino, Y., (2008). *Stability of flows past a pair of circular cylinders in a side-by-side arrangement.* J. Fluid Mech., 595, 491-507.

Murman, E. M. and Cole, J. D.,(1971). *Calculation of plane steady transonic flows*. Boeing Scientific Research Laboratories.

Pereira, R. M. S. and Gajjar, J. S. B. (2010). *Transonic Inviscid Flows Past Thin Airfoils: A New Numerical Method and Global Stability Analysis using MatLab.* International Journal of Mathematical Models and Methods in Applied Sciences, ISSN 1998-0140.

Trefethen L.N., (2000). *Spectral Methods in Matlab.* SIAM.

Weideman, J.A.C and Reddy, S.C (2003). *http://dip.sun.ac.za/∼weideman/research/differ.html* .