A Block Preconditioning Techniques for the Streamfunction-Vorticity Formulation of the Navier-Stokes Equations

A Report on

The British Council Summer Research Program

25 June 2007 – 24 August 2007

By

Dr. Faisal A. Fairag

King Fahd University of Petroleum and Minerals
College of Sciences
Department of Mathematics and Statistics
Dhahran 31261, Saudi Arabia
ffairag@kfupm.edu.sa
http://faculty.kfupm.edu.sa/math/ffairag

With the Collaboration of
Dr. Andy Wathen
The Computing Laboratory
University of Oxford
United Kingdom

September 2007

Contents

Abstract

This report is for a 9 weeks summer research visit to join the Numerical Analysis Group at the University of Oxford Computing Laboratory. This research project has been carried out with collaboration of Dr. Andy Wathen during the period 25 June- 24 August 2007.

In this research work, we describe and test block preconditioner techniques based on a Schur Complement approximation which uses multigrid methods for finite element approximations of the linearized incompressible Navier-Stokes equations in streamfunction-vertocity formulation. By using a Picard iteration, we use this techniques to solve fully nonlinear Navier-Stokes problems. One feature of this technique is that the preconditioning can be realized using any multigrid solver designed for the preconditioned system having an eigenvalue distribution consisting of a tightly clustered set together with a small number of outliers. Computations indicate that convergence for the preconditioned system is only mildly dependent on the viscosity and, most importantly, that the number of iterations does not grow as the mesh size is reduced.

The research study involves writing finite element computer codes, and testing them on problems which have a known solution. The outcomes of this research is this report, submitted paper to the Journal of Numerical Methods for Partial Differential Equations and a Graphical User interface Matlab codes.

1 Introduction

In this paper, we study the performance of a preconditioning technique based on the simple result of Murphy, Golub, and Wathen [8]for the streamfunction and vorticity formulation of the Navier-Stokes problem. This preconditioning technique is designed for systems arising from mixed finite element approximation.

We consider the steady-state Navier-Stokes equations describing the motion of an incompressible fluid in a convex polygonal domain $\Omega \in R^2$:

$$-Re^{-1} \triangle u + u \cdot \bigtriangledown u + \bigtriangledown p = f \qquad \text{in } \Omega \tag{1}$$

$$\bigtriangledown \cdot u = 0 \qquad \text{in } \Omega \tag{2}$$

subject to suitable boundary conditions on $\partial\Omega$, where u is the velocity and p the pressure of the fluid, f denotes the given body force, $\partial\Omega$ denote the boundary of Ω and $Re > 0$ the Reynolds number.

It is well-known [5] that the divergence-free condition (2) can be expressed by introducing the streamfunction ψ of u defined by $u = \text{ curl } \psi = [\psi_y, -\psi_x]^T$. The pressure can be eliminated by taking the curl of the equation (1), but this leads to a nonlinear fourth-order pde for the streamfunction ψ. This pde can be reduced into a system of second-order pdes by introducing the vorticity $\omega = \triangle\psi$. The resulting streamfunction-vorticity formulation is given by

$$-\triangle \psi = \omega \qquad \text{in } \Omega, \tag{3}$$

$$-Re^{-1} \triangle \omega = -curl\ \psi \cdot \bigtriangledown\ \omega + curl\ f \quad \text{in } \Omega, \tag{4}$$

subject to suitable boundary conditions on $\partial\Omega$. Note that as $Re \to 0$, equations (3-4) tend to a Stokes problems in the streamfunction-vorticity form. An efficient numerical technique for (3-4) can be obtained by a Picard iteration. More precisely, for $n = 1, 2, 3, \cdots$, one constructs iterates $\{\omega_n, \psi_n\}$ satisfying the linearized version of (3-4), known as the Oseen equations in the streamfunction-vorticity form

$$-\omega_n - \triangle \psi_n = 0 \qquad \text{in } \Omega, \tag{5}$$

$$-\varepsilon \triangle \omega_n + curl\ \psi_{n-1} \cdot \bigtriangledown\ \omega_n = curl\ f \quad \text{in } \Omega, \tag{6}$$

where $\varepsilon = Re^{-1}$. The Oseen equations (5-6) can be rewritten in a matrix form as

$$\begin{bmatrix} -1 & -\triangle \\ -\varepsilon \triangle + curl\ \psi_{n-1} \cdot \bigtriangledown & 0 \end{bmatrix} \begin{bmatrix} \omega_n \\ \psi_n \end{bmatrix} = \begin{bmatrix} 0 \\ curl\ f \end{bmatrix}. \tag{7}$$

The initial guess ψ_0 can be taken to be the solution to the associated Stokes-like problem

$$-\triangle \psi_0 = \omega_0 \quad \text{in} \quad \Omega, \tag{8}$$
$$-\varepsilon \triangle \omega_0 = curl\ f \quad \text{in} \quad \Omega, \tag{9}$$

A mixed finite element approximation of the Oseen problem (5-6) is obtained by constructing finite-dimensional subspaces $\Phi^h \subset \Phi$ and $W^h \subset W$, where Φ and W are appropriate Sobolev Spaces (see for example [6]). The discrete Oseen problem is then to

$$\text{Find } \psi_n^h \in \Phi^h \text{ and } w_n^h \in W^h \text{ such that}$$
$$-(\omega_n^h, \phi) + (\bigtriangledown \psi_n^h, \bigtriangledown \phi) = 0 \quad \forall\ \phi \in \Phi^h, \tag{10}$$
$$Adv(\omega_n^h, s) = (curl\ f, s) \quad \forall\ s \in W^h. \tag{11}$$

where $Adv(\omega_n^h, s) = \varepsilon(\bigtriangledown \omega_n^h, \bigtriangledown s) + (curl\ \psi_{n-1} \cdot \bigtriangledown\ \omega_n^h, s)$. The discrete constraint space $Z_h = \{s \in W^h : (\bigtriangledown s, \bigtriangledown \phi) = 0\ \forall \phi \in \Phi^h\}$ is not a subspace of the continuous analogue hence well-posedness of (10-11) does not automatically follow from the well-posedness of the continuous analogue. This can be treated by introducing an appropriate mesh dependent norms on $W^h \times \Phi^h$ (see [1]) if the space W^h consists of C^0 piecewise polynomials of degree $k \geq 1$ and $\Phi^h = W^h \cap H_0^1(\Omega)$. These mesh dependent norms will satisfy Brezzi's abstract stability conditions [2].

For any choice of Φ^h and W^h, the discrete system (10-11) can be written in block matrix form as

$$\begin{bmatrix} M & B^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \underline{\omega} \\ \underline{\psi} \end{bmatrix} = \begin{bmatrix} 0 \\ g \end{bmatrix}. \tag{12}$$

where $\underline{\omega}$ is a vector of the discrete vorticity variables, $\underline{\psi}$ a vector of the discrete streamfunction variables with B the discrete laplacian, $C = \varepsilon B + G$, G resulted from the term $-curl\ \psi_{n-1} \cdot \bigtriangledown\ \omega_n$, and M the mass matrix with a negative sign.

The focus of this study is the efficient solution to the linear system (12). This linear system (12) is nonsymmetric and indefinite. For large problem with a sparse coefficient matrix, iterative methods are preferable. Iterative solutions of systems of the form (12) can be found using a variety of methods. In particular, Krylov subspace methods (see for example [3]) such as GMRES

are applicable in such nonsymmetric systems (12). It is known that the performance of Krylov subspace methods is usually sensitive to the conditioning of the coefficient matrix and thus the idea of preconditioning must be implemented. Here we seek a preconditioning matrix P which is easy to construct and invert such that the preconditioned system is solved efficiently via an appropriate iterative method. Also, the role of P is to reduce the number of iterations required for convergence and at the same time not increase significantly the amount of computation at each iteration. A sufficient condition for a good preconditioner is that the preconditioner matrix $T = P^{-1}A$ has a few distinct eigenvalues where A denotes the coefficient matrix of the system.

Murphy, Golub and Wathen [8] have proposed preconditioners P_s and P_{ns} for matrices of the form (4) which give a preconditioned matrices T_s with at most three distinct eigenvalues and T_{ns} with exactly two distinct eigenvalues, respectively. This guarantees convergence of an appropriate iterative solver in at most three iterations. This result can be stated as:

- If the system (12) is preconditioned by $P_s = \begin{bmatrix} M & 0 \\ 0 & S \end{bmatrix}$, then the preconditioned matrix has at most three distinct eigenvalues when $S = CM^{-1}B^T$.
- If the system (12) is preconditioned by $P_{ns} = \begin{bmatrix} M & B^T \\ 0 & S \end{bmatrix}$, then the preconditioned matrix has exactly two distinct eigenvalues.

Hence, an exact inversion of the mass matrix M and formation and exact inversion of the Schur complement S or equivalently solutions of linear systems with these matrices would be needed: this would be very insufficient. So, we need to approximate the Schur complement S and the mass matrix M. A good choice of approximations for these matrices can lead to a very effective preconditioner as we show here. Approximation of these matrices will be discussed in the next sections and we will see that there will be two clusters of eigenvalues and Krylov subspace iterative convergence will still be very rapid whilst solutions of the approximate systems will be efficient.

2 Approximations of the Schur Complement

To determine an optimal approximation for S, it is useful to express (12) in the following way. We follow [10], by introducing the finite element basis set,

$$W_h = \text{Span } \{\phi_i\}_{i=1}^{m+nb}, \quad \Psi_h = \text{Span } \{\phi_j\}_{j=1}^{m}; \tag{13}$$

so that we have m interior and nb boundary degrees of freedom respectively. We then associate the functions $\omega_h,\ \psi_h,\ f_h$ with the vectors $\underline{\omega} \in R^{m+nb},\ \underline{\psi} \in R^m$ and $\underline{f} \in R^m$ of generalized coefficients,

$$\omega_h = \sum_{i=1}^{m+nb} \omega_i \phi_i, \quad \psi_h = \sum_{i=1}^{m} \psi_i \phi_i, \quad f_h = \sum_{i=1}^{m} f_i \phi_i. \tag{14}$$

This gives the linear system

$$\begin{bmatrix} M & B^T \\ C & 0 \end{bmatrix} \begin{bmatrix} \underline{w} \\ \underline{\psi} \end{bmatrix} = \begin{bmatrix} \underline{0} \\ \underline{f} \end{bmatrix}. \tag{15}$$

where
M is an $n \times n$ mass matrix defined by $M_{ij} = -(\phi_i, \phi_j)$.
B is an $m \times n$ matrix defined by $B_{ij} = (\bigtriangledown \phi_i, \bigtriangledown \phi_j)$.
C is an $m \times n$ matrix defined by $C_{ij} = \varepsilon B_{ij} + (curl\ \psi^* \bigtriangledown \phi_i, \bigtriangledown \phi_j)$.
f is an $m \times 1$ vector defined by $f_j = (curl\ f, \phi_j)$.

To see how we approximate the Schur complement S, we start by labeling the m interior nodes then the nb boundary nodes. This labeling will yield the decomposition of ω_n into the sum of interior and boundary contributions

$$\sum_{i=1}^{m+nb} \omega_i \phi_i = \sum_{i=1}^{m} \omega_i \phi_i + \sum_{i=1}^{nb} \omega_{m+i} \phi_{m+i}. \tag{16}$$

This decomposition induces a partitioning of the matrices $B,\ C$, and M into $B = [K_I \quad K_B],\ C = [\widetilde{K}_I \quad \widetilde{K}_B]$ and $M = [M_I,\ M_{BI};\ M_{BI}^T,\ M_B]$ where $K_I,\ \widetilde{K}_I$ and M_I are $m \times m$ matrices and $K_B,\ \widetilde{K}_B,\ M_{BI}$ are $m \times nb$ matrices and M_B is an $nb \times nb$ matrix. If the matrix M is lumped, e.g., using appropriate quadrature as in [7], then M is replaced by a block diagonal matrix $M_l = \text{diag}\ (M_I,\ M_B)$ and (15) can be written as

$$\begin{bmatrix} M_I & 0 & K_I \\ 0 & M_B & K_B \\ \widetilde{K}_I & \widetilde{K}_B & 0 \end{bmatrix} \begin{bmatrix} \underline{\omega}_I \\ \underline{\omega}_B \\ \underline{\psi} \end{bmatrix} = \begin{bmatrix} \underline{0} \\ \underline{0} \\ \underline{f} \end{bmatrix}. \tag{17}$$

where $\underline{\omega}_I = [\omega_1, \omega_2, \cdots, \omega_m]^T$ and $\underline{\omega}_B = [\omega_{m+1}, \omega_{m+2}, \cdots, \omega_{m+nb}]^T$.
The Schur complement in (17) can be expressed as

$$S_l = \widetilde{K}_I M_I^{-1} K_I + \widetilde{K}_B M_B^{-1} K_B^T. \tag{18}$$

The matrix S_l can be approximated by the matrix

$$S_{MG} = \widetilde{K}_{MG} M_I^{-1} K_{MG}, \tag{19}$$

where K_{MG} and $\widetilde{K}_{MG}$ represent the action of multigrid cycles applied to the Poisson problem and the action of multigrid cycles applied to the matrix $\widetilde{K}_I$ respectively.

3 Approximations of the Mass Matrix M

In this section, we are considering iterative methods for solving the system $Mz = r$, where M represents the mass matrix. To come up with an efficient preconditioner P while not increasing significantly the amount of computations, we propose two techniques to approximate the matrix M. The first one is the use of the diagonal matrix $D = \text{diag}(M)$. Wathen [11] provides a realistic upper and lower bounds on the eigenvalues of the preconditioned matrix $D^{-1}M$ and show that:

$$\lambda(D^{-1}M) \subset [\frac{1}{2}, 2] \tag{20}$$

The second technique is to apply few iterations of relaxed Jacobi method with a relaxation parameter $\alpha = \frac{4}{5}$ (i.e.)

$$Dz^{(k+1)} = (D - \alpha M)z^{(k)} + \alpha\ r. \tag{21}$$

With the choice of $\alpha = \frac{4}{5}$, the eigenvalues of the iteration matrix $D^{-1}(D - \alpha M)$ satisfies

$$\lambda(D^{-1}[D - \alpha M]) \subset [-\frac{3}{5}, \frac{3}{5}]. \tag{22}$$

This implies that

$$\| z - z^{(k)} \| \leq (\frac{3}{5})^k \| z - z^{(0)} \| . \tag{23}$$

4 Four Block-Preconditioners

In this section we consider four block preconditioning approaches that has been found to be effective. They are

$$P_1 = \begin{bmatrix} diag(M) & 0 \\ 0 & S_{MG} \end{bmatrix} \qquad P_2 = \begin{bmatrix} M_{Jk} & 0 \\ 0 & S_{MG} \end{bmatrix} \tag{24}$$

$$P_3 = \begin{bmatrix} diag(M) & B^T \\ 0 & S_{MG} \end{bmatrix} \qquad P_4 = \begin{bmatrix} M_{Jk} & B^T \\ 0 & S_{MG} \end{bmatrix} \tag{25}$$

where M_{Jk} represents the action of k relaxed Jacobi steps applied on the matrix M described in the previous section.

To see the distribution of the eigenvalues of the preconditioned matrices $T_1 = P_1^{-1}A$ and $T_3 = P_3^{-1}A$, eigenvalues are computed using $K_{MG} = K_I$, $Re = 1$ and plotted in Figure (4) and Figure (4), respectively. Observe that there is an eigenvalue cluster on each side of the origin, both of which expand with increasing mesh refinement. Figure (3) shows the eigenvalues of the preconditioned matrix $P_1^{-1}A$ for $Re = 1, 25, 50, 100$. From Figure (3),

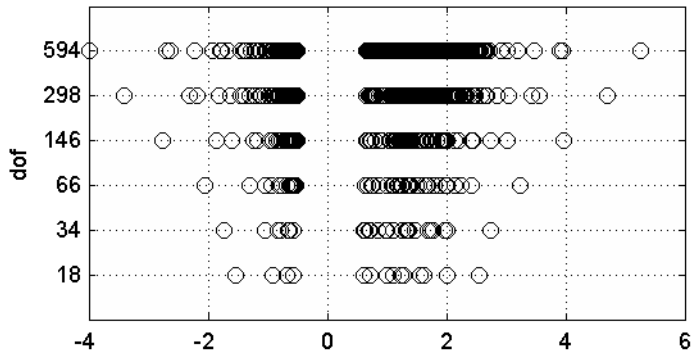

Figure 1: Computed eigenvalues for the preconditioned matrix $P_1^{-1}A$

the ratio of the maximum to the minimum eigenvalues is seen clearly to be growing very slowly as Re increases.

5 Numerical Results

In this section, results of some numerical experiments for the four types of preconditioner (24), (25) are presented. The test problem is a Navier-

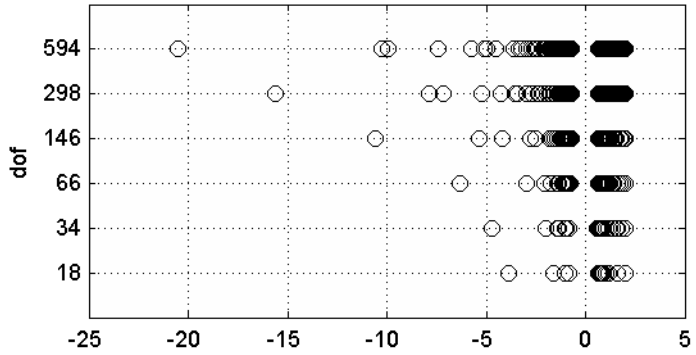

Figure 2: Computed eigenvalues for the preconditioned matrix $P_3^{-1}A$

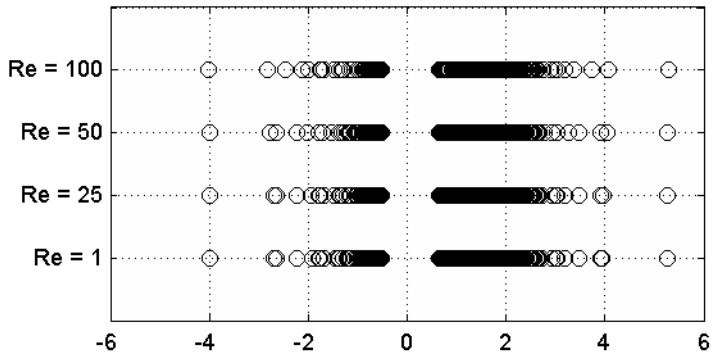

Figure 3: Computed eigenvalues for the preconditioned matrix $P_1^{-1}A$ for different values of Re

Stokes problem in streamfunction-vorticity with known solution. For this section, the domain Ω is the unit square $[0,1] \times [0,1]$ and the exact solution is $\psi^*(x,y) = (x\ (x-1)\ y\ (y-1))^2$. This test problem has been studied in [4, 9]. All computations were done using Matlab 7 on a Compaq nc8230 with Intel Mobile CPU 1.86 GHz running Windows XP. The discretization is performed on meshes generated by the Matlab command *initmesh*. Linear triangular elements are used with different values of *Re* and different mesh sizes. The Matlab command *refinemesh* is used to refine the initial mesh. The refinement method is regular refinement, where all triangles are divided into four triangles of the same shape.

The linear systems are solved using Matlab's function *gmres.m* with tolerance $1.0e-4$. In our computations, we used multigrid method for solving the systems that involve the discrete Laplacian matrix K_I and the discrete Laplacian plus advection matrix $\widetilde{K}_I$. K_I and $\widetilde{K}_I$ are replaced by a multigrid approximation K_{MG} and $\widetilde{K}_{MG}$ where one V-cycle is performed.

Solving the test problem using the matrices P_1, P_2, P_3, and P_4 as preconditioners gave the iteration counts in Table (1) with different mesh sizes. We use the following notation: 'no. elem' denotes the number of triangles, and 'no. dof' is the totla dimension of a discrete linear system (no. dof $= 2m + nb$). GMRES *iter* denotes the number of GMRES iterations needed. For example, in Table (1) where we consider the case of matrix P_1, the GMRES 5,7,7 means that 5 GMRES iterations are required for the first Stokes-like Oseen problem, 7 GMRES iterations are required for the second Stokes-like Oseen problem and 7 GMRES are required for the third and final Oseen problem at which point the nonlinear iteration has converged. Notice that P_4 has the smallest number of iterations and cpu time. Table (2) shows a comparison between MJ4 and MJ26 as an approximation to the mass matrix M. Table (3) displays the number of the GMRES iterations for each value of the Reynolds number.

	no. elem	16	32	64	144	296	592
	no. dof	18	34	66	146	298	594
P_1	GMRES iter.	5,7,7	8,10,10,10	12,14,14	21,21,21	35,35,35	43,43
	$\| e \|_\infty$	5.3e-5	5.2e-5	2.2e-5	1.3e-6	1.5e-6	4.2e-7
	cpu time	0.391	0.813	1.11	2.25	9.609	18.875
P_2	GMRES iter.	5,7,7	7,9,9,9	10,12,12	16,16,16	26,26,26	30,30
	$\| e \|_\infty$	5.3e-5	5.2e-5	2.2e-5	1.2e-6	1.5e-6	4.2e-7
	cpu time	0.422	0.766	1.032	1.938	7.734	14.735
P_3	GMRES iter.	4,5,5	9,10,10	12,14,14,14	23,25,25	34,34,34	39,39
	$\| e \|_\infty$	5.5e-5	5.2e-5	2.2e-5	1.2e-6	1.5e-6	4.2e-7
	cpu time	0.36	0.625	1.453	2.468	9.375	17.5
P_4	GMRES iter.	3,4,4	6,7,7	7,9,9	12,13,13	20,20,20	23,23
	$\| e \|_\infty$	5.3e-5	5.2e-5	2.2e-5	1.2e-6	1.5e-6	4.2e-7
	cpu time	0.391	0.547	0.922	1.734	6.437	12.078

Table 1: P_1, P_2, P_3, P_4 Comparisons

#elem	#dof	P_4 with iter	$k=4$ $\| e \|_\infty$	CPU	P_4 with iter	$k=26$ $\| e \|_\infty$	CPU
16	18	3,4,4	5.3e-5	0.391	2,3,3	5.3e-5	0.312
32	34	6,7,7,7	5.2e-5	0.547	3,4,4	5.2e-5	0.5
64	66	7,9,9	2.2e-5	0.922	5,5,5	2.2e-5	0.734
144	146	12,13,13	1.2e-6	1.734	7,7,7	1.2e-6	1.375
296	298	20,20,20	1.5e-6	6.437	12,12,12	1.5e-6	4.734
592	594	23,23	4.2e-7	12.078	15,15	4.2e-7	9.125

Table 2: M_{J4} .vs. M_{J26}

Re	iter	$\| e \|_\infty$	CPU
1	15,15	4.2e-7	8.75
10	19,19,19	6.1e-7	15.141
20	19,20,20	9.6e-7	15.344
30	20,20,20,20	1.3e-6	20.844
40	20,20,20,20	1.9e-6	20.906
50	20,20,20,20,20	2.6e-6	25.86
60	20,20,20,20,20	3.6e-6	26.281

Table 3: Number of GMRES iterations and cpu time for different values of Re.

6 MATLAB Program

Two MATLAB programs are written to solve the Navier-Stokes equations in the streamfunction vorticity formulation. The first program is written to compute and plot the eigenvalues of the preconditioned matrix. While the second program is written using finite element method to solve the Navier-Stokes equations with different preconditioning techniques. Graphical user interface is created using GUI MATLAB which ease exploring different values of the parameters of the problems and different type of preconditioners. Figure (6) shows the first window of the first MATLAB program. Figure (5) shows the first window of the second MATLAB program. 2D and 3D graphing utilities in MATLAB are used to view the main features of the flow and the eigenvalues distribution.

The code consists of several MATLAB FUNCTIONS. Each MATLAB

FUNCTION do a specific task. The MATLAB programs are included in the Appendix.

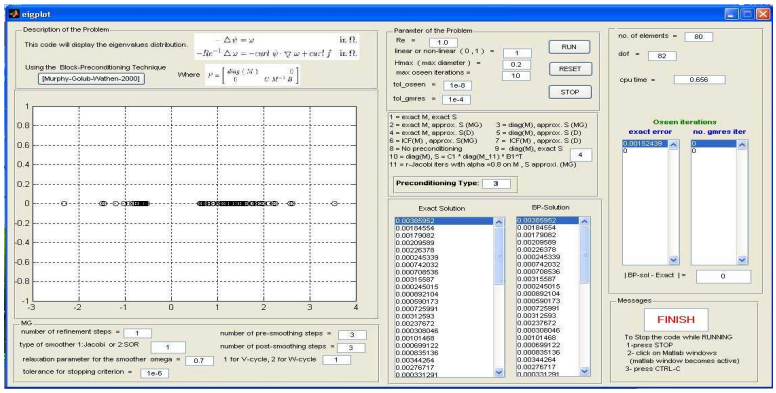

Figure 4: The first window of the MATLAB program which computes and plots the eigenvalues of the preconditioned matrix

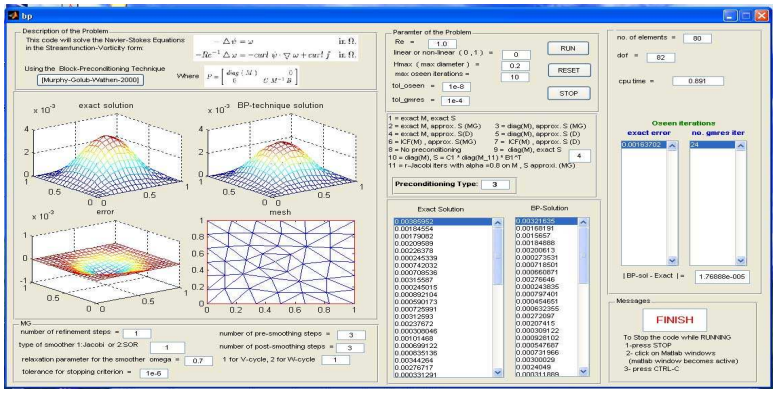

Figure 5: The first window of the MATLAB program which uses different preconditioning techniques

7 Conclusion

We have derived and tested a class of preconditioned iterative solution method for the Navier-Stokes equations in streamfunction and vorticity form which has almost linear complexity in terms of the number of degrees of freedom,

and which apparently requires a fixed number of iterations over a range of Reynolds numbers. This work represents a generalization of some of the results of Silvester and Mihajlovic of the Stokes problem to the Oseen problem and the fully nonlinear Navier-Stokes equations.

8 Acknowledgment

This research work was undertaken while the first author was visiting the Computing Laboratory at Oxford University supported by BAE-British Council grant number DAM/BAE/07/07. He would like to extend his gratitude to King Fahd University of Petroleum and Minerals for help and support. He would also like to express thanks to the British Council in Saudi Arabia for providing the financial support for this research and to the Computing Laboratory at Oxford University for their hospitality and support.

9 References

References

[1] I. Babuska,J. Osborn and J. Pitkaranta. Analysis of mixed methods using mesh dependant norms, *Math. comput.*, 35:1039–1062, 1980.

[2] F. Brezzi and M. Fortin. *Mixed and Hybrid finite element methods*, Springer-Verlag, New York, 1991.

[3] H. Elman, D. Silvester and A. Wathen. *Finite Elements and Fast Iterative Solvers*, Numerical Mathematics and Scientific Computation, Oxford University Press, Oxford, 2005.

[4] F. Fairag and M. S. Sahimi. The alternating group explicit (age) iterative method for solving a ladyzhenskaya model for stationary incompressible viscous flow, *to appear in Int. J. Comput. Math.*.

[5] V. Girault and P. A. Raviart. *Finite Element Approximation of the Navier-Stokes Equations*, volume 749. Springer, Berlin, 1979.

[6] M. Gunzburger. *Finite Element Method for Viscous Incompressible Flow : A Guide to Theory Practice and Algorithms.* Academic Press, Boston, 1989.

[7] M. Hanisch. Multigrid preconditioning for the biharmonic Dirichlet problem, *SIAM J. Numer. Anal.*, 30:184-214, 1993.

[8] M. Murphy, G. Golub and A. Wathen. A note on preconditioning for indefinite linear systems, *SIAM J. Sci. Comput.*, 21(6):1969–1972, 2000.

[9] F. Schieweck. On the order of two noncomforming finite element approximations of upwind type for the Navier-Stokes equations. Proceeding of the international Workshop on Numerical Methods for the Navier-Stokes Equations, Heidelberg, October 25-28, 1993, series Notes on Numerical Fluid Mechanics, Vieweg-Verlag.

[10] D. Silvester and M. Mihajlovic. A black-box multigrid preconditioner for the biharmonic equation, *BIT Numerical Mathematics*, 44:151–163, 2004.

[11] A. Wathen. Realistic Eigenvalue Bounds for the Galerkin Mass Matrix, *SIAM J. Numer. Anal.*, 7:449-457, 1987.

Appendix

MATLAB
PROGRAMS

Afun.m

```
function [z]=afun(x,nn)
global a_mat
z=a_mat*x;
```

Bp.m

```
function varargout = bp(varargin)
% BP M-file for bp.fig
%      BP, by itself, creates a new BP or raises the existing
%      singleton*.
%
%      H = BP returns the handle to a new BP or the handle to
%      the existing singleton*.
%
%      BP('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in BP.M with the given input arguments.
%
%      BP('Property','Value',...) creates a new BP or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before bp_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to bp_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help bp

% Last Modified by GUIDE v2.5 16-Aug-2007 16:24:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @bp_OpeningFcn, ...
                   'gui_OutputFcn',  @bp_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT



% --- Executes just before bp is made visible.
function bp_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to bp (see VARARGIN)

% Choose default command line output for bp
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes bp wait for user response (see UIRESUME)
% uiwait(handles.figure1);

gray=imread('gray.jpg');
axes(handles.axes6);  image(gray);  axis off;
axes(handles.axes7);  image(gray);  axis off;
axes(handles.axes8);  image(gray);  axis off;
axes(handles.axes9);  image(gray);  axis off;

psi_omega =imread('psi_omega.jpg');
axes(handles.axes10);  image(psi_omega);  axis off;

p_matrix =imread('p_matrix.jpg');
axes(handles.axes11);  image(p_matrix);  axis off;

% --- Outputs from this function are returned to the command line.
function varargout = bp_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



function Re_Callback(hObject, eventdata, handles)
% hObject    handle to Re (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Re as text
%        str2double(get(hObject,'String')) returns contents of Re as a double


% --- Executes during object creation, after setting all properties.
function Re_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Re (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function gam_Callback(hObject, eventdata, handles)
% hObject    handle to gam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of gam as text
%        str2double(get(hObject,'String')) returns contents of gam as a double


% --- Executes during object creation, after setting all properties.
function gam_CreateFcn(hObject, eventdata, handles)
% hObject    handle to gam (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function Hmax_Callback(hObject, eventdata, handles)
% hObject    handle to Hmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Hmax as text
%        str2double(get(hObject,'String')) returns contents of Hmax as a double


% --- Executes during object creation, after setting all properties.
function Hmax_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Hmax (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function max_oseen_iter_Callback(hObject, eventdata, handles)
% hObject    handle to max_oseen_iter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of max_oseen_iter as text
%        str2double(get(hObject,'String')) returns contents of max_oseen_iter as a double
```

```
% --- Executes during object creation, after setting all properties.
function max_oseen_iter_CreateFcn(hObject, eventdata, handles)
% hObject    handle to max_oseen_iter (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function tol_oseen_Callback(hObject, eventdata, handles)
% hObject    handle to tol_oseen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tol_oseen as text
%        str2double(get(hObject,'String')) returns contents of tol_oseen as a double


% --- Executes during object creation, after setting all properties.
function tol_oseen_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tol_oseen (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function tol_gmres_Callback(hObject, eventdata, handles)
% hObject    handle to tol_gmres (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of tol_gmres as text
%        str2double(get(hObject,'String')) returns contents of tol_gmres as a double


% --- Executes during object creation, after setting all properties.
function tol_gmres_CreateFcn(hObject, eventdata, handles)
% hObject    handle to tol_gmres (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
```

```
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on button press in run.
function run_Callback(hObject, eventdata, handles)
% hObject    handle to run (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

main_gui

% --- Executes on button press in stop.
function stop_Callback(hObject, eventdata, handles)
% hObject    handle to stop (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

reset2


% --- Executes on selection change in exact_sol.
function exact_sol_Callback(hObject, eventdata, handles)
% hObject    handle to exact_sol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns exact_sol contents as cell array
%        contents{get(hObject,'Value')} returns selected item from exact_sol


% --- Executes during object creation, after setting all properties.
function exact_sol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to exact_sol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in BP_sol.
function BP_sol_Callback(hObject, eventdata, handles)
% hObject    handle to BP_sol (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns BP_sol contents as cell array
%        contents{get(hObject,'Value')} returns selected item from BP_sol


% --- Executes during object creation, after setting all properties.
function BP_sol_CreateFcn(hObject, eventdata, handles)
% hObject    handle to BP_sol (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in error.
function error_Callback(hObject, eventdata, handles)
% hObject    handle to error (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns error contents as cell array
%        contents{get(hObject,'Value')} returns selected item from error


% --- Executes during object creation, after setting all properties.
function error_CreateFcn(hObject, eventdata, handles)
% hObject    handle to error (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function er2_Callback(hObject, eventdata, handles)
% hObject    handle to er2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of er2 as text
%        str2double(get(hObject,'String')) returns contents of er2 as a double


% --- Executes during object creation, after setting all properties.
function er2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to er2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function msg1_Callback(hObject, eventdata, handles)
% hObject    handle to msg1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of msg1 as text
%        str2double(get(hObject,'String')) returns contents of msg1 as a double


% --- Executes during object creation, after setting all properties.
function msg1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to msg1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on button press in reset_b.
function reset_b_Callback(hObject, eventdata, handles)
% hObject    handle to reset_b (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

reset2




function msg2_Callback(hObject, eventdata, handles)
% hObject    handle to msg2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of msg2 as text
%        str2double(get(hObject,'String')) returns contents of msg2 as a double


% --- Executes during object creation, after setting all properties.
function msg2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to msg2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

```
% --- Executes on selection change in listbox4.
function listbox4_Callback(hObject, eventdata, handles)
% hObject    handle to listbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns listbox4 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from listbox4


% --- Executes during object creation, after setting all properties.
function listbox4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to listbox4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: listbox controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function dof_Callback(hObject, eventdata, handles)
% hObject    handle to dof (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of dof as text
%        str2double(get(hObject,'String')) returns contents of dof as a double


% --- Executes during object creation, after setting all properties.
function dof_CreateFcn(hObject, eventdata, handles)
% hObject    handle to dof (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function nelem_Callback(hObject, eventdata, handles)
% hObject    handle to nelem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of nelem as text
%        str2double(get(hObject,'String')) returns contents of nelem as a double


% --- Executes during object creation, after setting all properties.
function nelem_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nelem (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function precond_type_Callback(hObject, eventdata, handles)
% hObject    handle to precond_type (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of precond_type as text
%        str2double(get(hObject,'String')) returns contents of precond_type as a double


% --- Executes during object creation, after setting all properties.
function precond_type_CreateFcn(hObject, eventdata, handles)
% hObject    handle to precond_type (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function cpu_time_Callback(hObject, eventdata, handles)
% hObject    handle to cpu_time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of cpu_time as text
%        str2double(get(hObject,'String')) returns contents of cpu_time as a double


% --- Executes during object creation, after setting all properties.
function cpu_time_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cpu_time (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function edit14_Callback(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit14 as text
%        str2double(get(hObject,'String')) returns contents of edit14 as a double


% --- Executes during object creation, after setting all properties.
function edit14_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end



function smooth_i_Callback(hObject, eventdata, handles)
% hObject    handle to smooth_i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of smooth_i as text
%        str2double(get(hObject,'String')) returns contents of smooth_i as a double


% --- Executes during object creation, after setting all properties.
function smooth_i_CreateFcn(hObject, eventdata, handles)
% hObject    handle to smooth_i (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function omega_Callback(hObject, eventdata, handles)
% hObject    handle to omega (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of omega as text
%        str2double(get(hObject,'String')) returns contents of omega as a double


% --- Executes during object creation, after setting all properties.
function omega_CreateFcn(hObject, eventdata, handles)
% hObject    handle to omega (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function nu1_Callback(hObject, eventdata, handles)
% hObject    handle to nu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nu1 as text
%        str2double(get(hObject,'String')) returns contents of nu1 as a double


% --- Executes during object creation, after setting all properties.
function nu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function nu2_Callback(hObject, eventdata, handles)
% hObject    handle to nu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of nu2 as text
%        str2double(get(hObject,'String')) returns contents of nu2 as a double


% --- Executes during object creation, after setting all properties.
function nu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to nu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
```

```
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function mu_Callback(hObject, eventdata, handles)
% hObject    handle to mu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mu as text
%        str2double(get(hObject,'String')) returns contents of mu as a double


% --- Executes during object creation, after setting all properties.
function mu_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


function edit20_Callback(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit20 as text
%        str2double(get(hObject,'String')) returns contents of edit20 as a double


% --- Executes during object creation, after setting all properties.
function edit20_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit20 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in MGW2000.
function MGW2000_Callback(hObject, eventdata, handles)
% hObject    handle to MGW2000 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)

open('MGW2000.htm')

function rjacobi_Callback(hObject, eventdata, handles)
% hObject    handle to rjacobi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rjacobi as text
%        str2double(get(hObject,'String')) returns contents of rjacobi as a double


% --- Executes during object creation, after setting all properties.
function rjacobi_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rjacobi (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```

compute_e.m

```
function [e] = compute_e(f11,f12,f21,f22)
e=zeros(8,1);
e(1)=-(f22^2*f12 + f22*f12^2);
e(2)=f21^2*f11 + f21*f11^2;
e(3)=-(f21^2*f12 + f11^2*f22);
e(4)=f22^2*f11 + f12^2*f21;
e(5)=f12*f22*f21 + f22*f12*f11;
e(6)=f11*f21*f22 + f11*f21*f12;
e(7)=f22*f12*f21 + f22*f12*f11;
e(8)=-(f21*f11*f22 + f21*f11*f12 );
```

compute_error.m

```
function [er1,er2,er3,exact_sol,p_sol,direct_sol] = compute_error(n,nn,nb,sol,sol_direct,exsol,ds,p,t,e)

exact_sol  = [exsol(n+1:nn,1);zeros(nb,1)];   %(n+1:nn);ds(m+1:n)];
p_sol      = [sol(n+1:nn)      ;zeros(nb,1)];
direct_sol = [sol_direct(n+1:nn);zeros(nb,1)];

format long;
%fprintf('    direct solution      our solution     exact solution\n')
%full([direct_sol,p_sol,exact_sol]);
%fprintf('    ------------      ---------------     --------------\n')
%fprintf('    direct solution      our solution     exact solution\n')
%fprintf('    ------------      ---------------     --------------\n')

% -------------  error computaions ------------------
er1=norm(p_sol - direct_sol)/norm(direct_sol);
fprintf('| sol - direct | =  %.4e\n', er1)
er2=norm(p_sol-exact_sol)/nn;
fprintf('| sol - exact | =  %.4e\n', er2)
er3=norm(direct_sol-exact_sol)/norm(exact_sol);
```

```
fprintf('| exact - direct | =  %.4e\n', er3)
% ------------  error computaions -----------------
```

compute_exact.m

```
function [exsol,exsol_psi_bodry] = compute_exact(p,m,nb)
% compute exact solution
xx = p(1,:)'; yy = p(2,:)';
ex_psi = inline(' ( (x-1).*x .*(y-1).*y ).^2 ','x','y');
ex_omega = inline( '-2.*x.^2 + 4.*x.^3 - 2.*x.^4 + 12.*x.^2.*y - 24.*x.^3.*y + 12.*x.^4.*y - 2.*y.^2 + 12.*x.*y.^2 -
24.*x.^2.*y.^2 + 24.*x.^3.*y.^2 - 12.*x.^4.*y.^2 + 4.*y.^3 - 24.*x.*y.^3 + 24.*x.^2.*y.^3 - 2.*y.^4 + 12.*x.*y.^4 -
12.*x.^2.*y.^4','x','y');
ex1= ex_psi(xx,yy);
ex2= ex_omega(xx,yy);
exsol = [ex2;ex1(1:m,1)];
exsol_psi_bodry = ex1(m+1:m+nb,1);
```

Curlf.m

```
function [value]=curlf(x,y,Re,gam);
mu = 1/Re;
% gam = 0 for biharmonic equation ; gam = 1 for Navier-Stokes Equations
value = ...
8.*(mu.*(1 - 6.*x.^3 + 3.*x.^4 + ...
    9.*x.^2.*(1 - 2.*y).^2 - 6.*y + 9.*y.^2 - ...
    6.*y.^3 + 3.*y.^4 - 6.*x.*(1 - 6.*y + 6.*y.^2)) ...
   + gam.*(-1 + x).*x.*(-1 + y).*y.* ...
  (-9.*x.^5.*(1 - 2.*y).^2 + ...
    3.*x.^6.*(1 - 2.*y).^2 - ...
    3.*(-1 + y).^3.*y.^3 + ...
    x.^3.*(-3 + 16.*y - 16.*y.^2) + ...
    2.*x.*(-1 + y).^2.*y.^2.* ...
     (1 - 6.*y + 6.*y.^2) + ...
    x.^4.*(9 - 38.*y + 38.*y.^2) - ...
    2.*x.^2.*(y - 8.*y.^3 + 19.*y.^4 - 18.*y.^5 + ...
      6.*y.^6)));
```

Eig_plot.m

```
Re=10;
[v10]=give_eigc(10,Re); w10=zeros(length(v10));
[v15]=give_eigc(15,Re); w15=zeros(length(v15))+1;
[v20]=give_eigc(20,Re); w20=zeros(length(v20))+2;
[v25]=give_eigc(25,Re); w25=zeros(length(v25))+3;
[v30]=give_eigc(30,Re); w30=zeros(length(v30))+4;
%[v40]=give_eigc(40,Re); w40=zeros(length(v40))+1;
%[v50]=give_eigc(50,Re); w50=zeros(length(v50))+1;
plot(v10,w10,'ok',v15,w15,'ok',v20,w20,'ok',v25,w25,'ok',v30,w30,'ok')  %,'o',v40,w40,'o',v50,w50,'o')
```

get_a.m

```
function [a_mat]=get_a(C,B,M,p,e,t);

[nelem,nb,nn,m,n] = get_parameters(p,e,t);

a_mat=sparse(nn,nn);

z=sparse(m,m);

a_mat=[M,B';C,z];
```

get_advection.m

```
function [advec] = get_advection(r,s,t,e,wq,phix,phiy);

   int1 = phix(t,:).*phix(s,:).*phix(r,:)*e(1);
   int2 = phiy(t,:).*phiy(s,:).*phiy(r,:)*e(2);
   int3 = phix(t,:).*phiy(s,:).*phiy(r,:)*e(3);
   int4 = phiy(t,:).*phix(s,:).*phix(r,:)*e(4);
   int5 = phix(t,:).*phix(s,:).*phiy(r,:)*e(5);
   int6 = phiy(t,:).*phiy(s,:).*phix(r,:)*e(6);
   int7 = phix(t,:).*phiy(s,:).*phix(r,:)*e(7);
   int8 = phiy(t,:).*phix(s,:).*phiy(r,:)*e(8);

advec = (int1+int2+int3+int4+int5+int6+int7+int8) * wq;
```

get_BM.m

```
function [B,M]=get_BM(p,e,t);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Re = the Reynolds number
%  np = number of nodes on the x-axis
%  ds = an approximation of the streamfunction at the interior nodes
%     =  a vector of size  n with zeros at the last nb entries
%        which corresponds to the streamfunction values at the boundary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  n = m + nb;       % # of all nodes = # of DEF for the vorticity
%  nn = n + m;       % # of all DEF for vorticity and streamfunction

% give me the quadrature information i.e
% nqpt = number of quadrature points, wq = the wight, qpx, qpy = the
% coordinate of thsee points

[nqpt,wq,xq,yq]= give_quad_info();

% give me the the x-der, y-der value of the basis functions
% at all quadrature points i.e
% phix, phiy = are matrices of size 4 x nqpt
% phix(i,q) = the x-der value of the local ref basis function phi_i
%             at the point ( xi_q , eta_q )

[phi,phix,phiy] = phi_info(nqpt,xq,yq);

%ds=zeros(nn,1);
%[ds]=exact_ds(m,nelem,elnode,coord,n);

[nelem,nb,nn,m,n] = get_parameters(p,e,t);
[elnode,coord] = get_elnode(p,e,t);
BB=sparse(n,n);
MM=sparse(n,n);
z=sparse(m,m);

%b = 'squareb1';
%[MM, r1] = assempde(b, p, e, t, 0, 1, 0); % assemble the mass matrix M
%[BB, r1] = assempde(b, p, e, t, 1, 0, 0); % assemble the laplacian matrix
%B
%M = MM(order,order);
%B = -BB(int,order);

%a_mat=[M,B';B,z];
```

```
%%%%%%%%%%%   START: form the rhs  %%%%%%%%%%%%%%%%%%%%
for k=1:nelem
   x = coord( elnode(k,:) , 1);
   y = coord( elnode(k,:) , 2);
   f11=x(2)-x(1);f22=y(3)-y(1);f12=x(3)-x(1);f21=y(2)-y(1);
   [e] = compute_e(f11,f12,f21,f22);
   Dt=f11*f22-f12*f21;E1t=f12^2+f22^2;E2t=f11*f12+f21*f22;
   E3t=f11^2+f21^2;
   b1=x(1);b2=y(1);
   for r=1:3
      j=elnode(k,r);
      for s=1:3
         at=1;qt=-1;
         i=elnode(k,s);
         term1=phix(r,:).*phix(s,:);
         term2=phiy(r,:).*phix(s,:)+phix(r,:).*phiy(s,:);
         term3=phiy(r,:).*phiy(s,:);
         cont = at*(term1*wq*E1t - term2*wq*E2t + term3*wq*E3t);
         BB(i,j)=BB(i,j)+ cont/Dt;     % laplacian matrix
         mass1 = qt*phi(r,:).*phi(s,:);
         mass2 = mass1*wq;
         MM(i,j) = MM(i,j) + mass2*Dt; % mass matrix
      end
   end
end
%%%%%%%%%%%   END:  form the rhs  %%%%%%%%%%%%%%%%%%%%
B=sparse(m,n);
M=sparse(n,n);
B=BB(1:m,1:m+nb);
M=MM(1:n,1:n);
```

get_BsMs.m

```
function [Bs,Ms]=get_BsMs(ps,es,ts);

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

Bs = cell(steps+1,1);
Ms = cell(steps+1,1);

for i=1:steps+1
   [Bs{i},Ms{i}]=get_BM(ps{i},es{i},ts{i});
end
```

get_C.m

```
function [C]=get_C(Re,gam,ds,p,e,t,B,M);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Re = the Reynolds number
%  np = number of nodes on the x-axis
%  ds = an approximation of the streamfunction at the interior nodes
%     =  a vector of size  n with zeros at the last nb entries
%        which corresponds to the streamfunction values at the boundary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mu = 1/Re;
%  n = m + nb;       % # of all nodes = # of DEF for the vorticity
%  nn = n + m;       % # of all DEF for vorticity and streamfunction

[nelem,nb,nn,m,n] = get_parameters(p,e,t);
[elnode,coord] = get_elnode(p,e,t);
```

```
% give me the quadrature information i.e
% nqpt = number of quadrature points, wq = the wight, qpx, qpy = the
% coordinate of thsee points

[nqpt,wq,xq,yq]= give_quad_info();

% give me the the x-der, y-der value of the basis functions
% at all quadrature points i.e
% phix, phiy = are matrices of size 4 x nqpt
% phix(i,q) = the x-der value of the local ref basis function phi_i
%             at the point ( xi_q , eta_q )

[phi,phix,phiy] = phi_info(nqpt,xq,yq);

HH=sparse(n,n);

%%%%%%%%%%%   START: form the rhs  %%%%%%%%%%%%%%%%%%%
for k=1:nelem
   x = coord( elnode(k,:) , 1);
   y = coord( elnode(k,:) , 2);
   f11=x(2)-x(1);f22=y(3)-y(1);f12=x(3)-x(1);f21=y(2)-y(1);
   [e] = compute_e(f11,f12,f21,f22);
   Dt=f11*f22-f12*f21;
   b1=x(1);b2=y(1);
   for r=1:3;  j=elnode(k,r);
   for s=1:3;  i=elnode(k,s);
   for t=1:3;  l = elnode(k,t);
      % we have fix here elnode is on old labeling
      % while ds in the new lableing i mean according to
      % order vector
      [adv] = get_advection(r,s,t,e,wq,phix,phiy);
      HH(i,j) = HH(i,j) + ds(l)*adv/(Dt^2);  % advection matrix
   end
   end
   end
end
%%%%%%%%%%%   END:  form the rhs  %%%%%%%%%%%%%%%%%%%
H=HH(1:m,1:n);
C= mu * B + gam * H;
```

get_Cs.m

```
function [Cs]=get_Cs(Re,gam,ds,ps,es,ts,Bs,Ms);

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

Cs = cell(steps+1,1);
ds_s = cell(steps+1,1);

ds_s{steps+1} = ds;
for i=steps:-1:1
   [ds_s{i}] = restrict_v(ds_s{i+1},i+1);
end

for i=1:steps+1
   p=ps{i};e=es{i};t=ts{i};
   [nelem,nb,nn,m,n] = get_parameters(p,e,t);
   [Cs{i}]=get_C(Re,gam,ds_s{i},p,e,t,Bs{i},Ms{i});
end
```

get_edges.m

```
function edges = get_edges(p, t)
% Locates the edges of a finite element mesh.
% Especially useful for the implementation of transfer operators in
% the context of multigrid methods, since the endpoints of an edge are
% exactly the two fathers of a newly created FE node.
%
% INPUT:   p  2 by np matrix containing the coordinates of the
%            finite element nodes
%          t  4 by nt matrix containing the indices (with
%            respect to p) of the vertices of the triangulation
%
% OUTPUT:  edges  2 by ne matrix, ne the number of edges. The i-th
%                 column contains the indices (with respect to p) of the
%                 edge endpoints
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net

  np = size(p,2);
  nt = size(t,2);
  it = (1:nt);
  ip1 = t(1,it);
  ip2 = t(2,it);
  ip3 = t(3,it);
  A = sparse(ip1, ip2, -1, np, np);
  A = A + sparse(ip2, ip3, -1, np, np);
  A = A + sparse(ip3, ip1, -1, np, np);
  A = -((A + A.') < 0);
  [i1,i2] = find(A == -1 & A.' == -1);
  i = find(i2 > i1);
  i1 = i1(i)';
  i2 = i2(i)';
  edges = [i1; i2];
```

get_elnode.m

```
function [elnode,coord] = get_elnode(p,e,t);

elnode=t(1:3,:)';
coord=p';
```

get_grids.m

```
function [ps,es,ts,pms,ems,tms] = get_grids(Hmax);

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

ps = cell(steps+1,1);
es = cell(steps+1,1);
ts = cell(steps+1,1);
pms = cell(steps+1,1);
ems = cell(steps+1,1);
tms = cell(steps+1,1);

hc = (2^(steps+1))*Hmax;

[pi, ei, ti] = initmesh(geom, 'Hmax', hc); % generate initial mesh
pms{1}=pi;ems{1}=ei;tms{1}=ti;
```

```
[ps{1},es{1},ts{1},pi,ei,ti] = relabel(pi,ei,ti);

for i = 1:steps;
   [pi, ei, ti] = refinemesh(geom, pi, ei, ti);
   pms{i+1}=pi;ems{i+1}=ei;tms{i+1}=ti;
   [ps{i+1},es{i+1},ts{i+1},pi,ei,ti] = relabel(pi,ei,ti);
end
```

get_MG_info.m

```
function [steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
global smooth1
global mg_par

steps = mg_par(1);        % number of refinement steps
smooth = smooth1;% type of smoother,
              % 'Jacobi' for damped Jacobi or 'SOR' for SOR
omega = mg_par(2);      % relaxation parameter for the smoother
nu1 = mg_par(4);          % number of pre-smoothing steps
nu2 = mg_par(5);          % number of post-smoothing steps
mu = mg_par(6);           % number of recursive multigrid step calls,
              % 1 for V-cycle, 2 for W-cycle
tol = mg_par(3);       % tolerance for stopping criterion
maxit = 100;      % maximum number of multigrid iterations
exact = 0;        % flag whether exact solution is available
prec = 0;         % flag whether multigrid is used as preconditioner
geom = 'squareg'; % Matlab geometry description
geomb = 'squareb1'; % Matlab bondary description
```

get_parameters.m

```
function [nelem,nb,nn,m,n] = get_parameters(p,e,t);

m = size(p, 2)- size(e, 2); % number of degrees of freedom for the streamfunction
n =  size(p, 2); % number of degrees of freedom for vorticity
nb = n - m;  % no of boundary nodes
nn = m + nb + m ;
nelem=size(t,2);
```

get_rhs.m

```
function [rhs]=get_rhs(Re,gam,p,e,t);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Re = the Reynolds number
%  np = number of nodes on the x-axis
%  ds = an approximation of the streamfunction at the interior nodes
%     =  a vector of size  n with zeros at the last nb entries
%        which corresponds to the streamfunction values at the boundary
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  n = m + nb;       % # of all nodes = # of DEF for the vorticity
%  nn = n + m;       % # of all DEF for vorticity and streamfunction

% give me the quadrature information i.e
% nqpt = number of quadrature points, wq = the wight, qpx, qpy = the
% coordinate of thsee points

[nelem,nb,nn,m,n] = get_parameters(p,e,t);
[nqpt,wq,xq,yq]= give_quad_info();
```

```
% give me the the x-der, y-der value of the basis functions
% at all quadrature points i.e
% phix, phiy = are matrices of size 4 x nqpt
% phix(i,q) = the x-der value of the local ref basis function phi_i
%            at the point ( xi_q , eta_q )

[phi,phix,phiy] = phi_info(nqpt,xq,yq);

%ds=zeros(nn,1);
%[ds]=exact_ds(m,nelem,elnode,coord,n);


[elnode,coord] = get_elnode(p,e,t);
rhs = sparse(nn,1);
g = sparse(n,1);

%b = 'squareb1';

%%%%%%%%%%%   START: form the rhs  %%%%%%%%%%%%%%%%%%%%
for k=1:nelem
   x = coord( elnode(k,:) , 1);
   y = coord( elnode(k,:) , 2);
   f11=x(2)-x(1);f22=y(3)-y(1);f12=x(3)-x(1);f21=y(2)-y(1);
   [e] = compute_e(f11,f12,f21,f22);
   Dt=f11*f22-f12*f21;E1t=f12^2+f22^2;E2t=f11*f12+f21*f22;
   E3t=f11^2+f21^2;
   b1=x(1);b2=y(1);
   xqt = f11*xq +f12*yq + b1; %(1-xq-yq)*x(1) + xq*x(2) + yq*x(3);
   yqt = f21*xq +f22*yq + b2; %(1-xq-yq)*y(1) + xq*y(2) + yq*y(3);
   for r=1:3
      j=elnode(k,r);
%      here we compute the rhs using quadratures
%      c1 = is the curlf function values at the quadrature points
%      c2 = is the local basis functions at the quadrature points
%      wq = is the quadrature weights
      c1=curlf( xqt , yqt , Re, gam);
      c2=phi(r,:)';
      c3=c1 .* c2;
      contr1 = wq' * c3;
      g(j) = g(j) +  contr1*Dt;
   end
end
%%%%%%%%%%%   END:  form the rhs  %%%%%%%%%%%%%%%%%%%%

g1=sparse(n,1);
g2=g(1:m);
rhs=[g1;g2];
```

give_eig.m

```
function [v]=give_eig(np,Re);
%clear
%np=16;                % # of nodes on the x-axis
%Re=1000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mu = 1/Re;
h=1/(np-1);           % mesh size
nix= np-2;            % # of interior nodes on each horizontal line
m = nix^2;            % # of interior nodes = # of DEF for the streamfunction
nb = 4*nix + 4;       % # of boundary nodes
```

```
n = m + nb;            % # of all nodes = # of DEF for the vorticity
nelem = ( np - 1)^2;   % # of elements
%
% here we call the m-function to get all information about the grid
% bo = boundary nodes
% int = interior nodes
% elnode = local labeling vs global labeling
% coord = nodes xy-coordinates
%
[bo,int,elnode,coord] = grid_info(np,n,h,nelem);
%
%  load all local matrices comes from each operator
%
[loc_mass,loc_lap,loc_adv] = phi_matrices();
%
% initial condition for streamfuction
%
%ds=zeros(n,1);
[ds]=exact_ds(m,nelem,elnode,coord,n);
%
%
%
%
a=sparse(n,n);
aa=sparse(n,n);
bb=sparse(n,n);
cc=sparse(n,n);
adv=sparse(n,n);
for k=1:nelem
   for t=1:4
      j=elnode(k,t);
      for s=1:4
         i=elnode(k,s);
         aa(i,j) = aa(i,j) + h^2 * loc_mass(s,t);
         bb(i,j) = bb(i,j) + loc_lap(s,t);
         cc(i,j) = cc(i,j) + mu * loc_lap(s,t);
         %
         sum=0;
         for ik=1:4
            kk=elnode(k,ik);
            sum = sum + ds(kk)*loc_adv(s,t,ik);
         end
         adv(i,j)=adv(i,j)+sum;
      end
   end
end
%z=zeros(n,m);
%a1=aa(1:m,1:n);
%b1=bb(1:m,1:m);
%c1=cc(1:n,1:n);
%a=[a1,b1;c1,z];
e1=aa(int,int);
e2=aa(int,bo);
e3=bb(int,int);
e4=aa(bo,int);
e5=aa(bo,bo);
e6=bb(bo,int);
e7=cc(int,int);
e8=cc(int,bo);
e9=sparse(m,m);
```

```
a=[e1,e2,e3;e4,e5,e6;e7,e8,e9];
zz=inv(e3)*e1*inv(e7)*e8*inv(e5)*e6;
I=speye(size(zz));
T=I+zz;
v=eig(full(T));



function [ds]=exact_ds(m,nelem,elnode,coord,n);
% this function comute the exact solution of psi ( streamfunction )
% at all the interior node and return thevalues at the vector ds
%
for k=1:n
   x=coord(k,1);
   y=coord(k,2);
   ds(k)= 1000*( x * (x-1) * y * (y-1) )^2;
end
```

give_eigc.m

```
function [v]=give_eig(np,Re);
%clear
%np=16;              % # of nodes on the x-axis
%Re=1000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mu = 1/Re;
h=1/(np-1);          % mesh size
nix= np-2;           % # of interior nodes on each horizontal line
m = nix^2;           % # of interior nodes = # of DEF for the streamfunction
nb = 4*nix + 4;      % # of boundary nodes
n = m + nb;          % # of all nodes = # of DEF for the vorticity
nelem = ( np - 1)^2;  % # of elements
%
% here we call the m-function to get all information about the grid
% bo = boundary nodes
% int = interior nodes
% elnode = local labeling vs global labeling
% coord = nodes xy-coordinates
%
[bo,int,elnode,coord] = grid_info(np,n,h,nelem);
%
%  load all local matrices comes from each operator
%
[loc_mass,loc_lap,loc_adv] = phi_matrices();
%
% initial condition for streamfuction
%
%ds=zeros(n,1);
[ds]=exact_ds(m,nelem,elnode,coord,n);
%
%
%
%
a=sparse(n,n);
aa=sparse(n,n);
bb=sparse(n,n);
cc=sparse(n,n);
adv=sparse(n,n);
for k=1:nelem
   for t=1:4
```

```
      j=elnode(k,t);
      for s=1:4
         i=elnode(k,s);
         aa(i,j) = aa(i,j) + h^2 * loc_mass(s,t);
         bb(i,j) = bb(i,j) + loc_lap(s,t);
         cc(i,j) = cc(i,j) + mu * loc_lap(s,t);
         %
         sum=0;
         for ik=1:4
            kk=elnode(k,ik);
            sum = sum + ds(kk)*loc_adv(s,t,ik);
         end
         adv(i,j)=adv(i,j)+sum;
      end
   end
end
%z=zeros(n,m);
%a1=aa(1:m,1:n);
%b1=bb(1:m,1:m);
%c1=cc(1:n,1:n);
%a=[a1,b1;c1,z];
e1=aa(int,int);
e2=aa(int,bo);
e3=bb(int,int);
e4=aa(bo,int);
e5=aa(bo,bo);
e6=bb(bo,int);
e7=cc(int,int);
e8=cc(int,bo);
e9=sparse(m,m);
a=[e1,e2,e3;e4,e5,e6;e7,e8,e9];
zz=inv(e3)*e1*inv(e7)*e8*inv(e5)*e6;
I=speye(size(zz));
T=I+zz;
v=eig(full(T));



function [ds]=exact_ds(m,nelem,elnode,coord,n);
% this function comute the exact solution of psi ( streamfunction )
% at all the interior node and return thevalues at the vector ds
%
for k=1:n
   x=coord(k,1);
   y=coord(k,2);
   ds(k)= 1000*( x * (x-1) * y * (y-1) )^2;
end
```

give_quad_info.m

```
function [nqpt,wq,xq,yq]= give_quad_info();

nqpt=6;
xy=[  0.659027622374092  0.231933368553031; ...
      0.659027622374092  0.109039009072877;  ...
      0.231933368553031  0.659027622374092; ...
      0.231933368553031  0.109039009072877; ...
      0.109039009072877  0.659027622374092; ...
      0.109039009072877  0.231933368553031  ];
xq = xy(:,1);
```

```
yq = xy(:,2);
w1=0.16666666666666666667/2;
wq=[w1;w1;w1;w1;w1;w1];

% quadrature for three points
%nqpt=3;
%xq=[0.5;0.5;0];
%yq=[0;0.5;0.5];
%wq1=1/6;
%wq=[wq1;wq1;wq1];
```

graph_sol.m

```
% -------------  Graph the Solutions -----------------
%ti=-1:0.1:1;
ti=0:0.05:1;
[xi,yi]=meshgrid(ti,ti);

[elnode,coord] = get_elnode(p,e,t);
xxx=coord(:,1);
yyy=coord(:,2);

axes(handles.axes8)
z2=griddata(xxx,yyy,exact_sol,xi,yi);
mesh(xi,yi,z2)
title('exact solution')

axes(handles.axes6)
z1=griddata(xxx,yyy,p_sol,xi,yi);
mesh(xi,yi,z1)
title('BP-technique solution')

axes(handles.axes9)
error=p_sol-exact_sol;
z3=griddata(xxx,yyy,error,xi,yi);
mesh(xi,yi,z3)
title('error')

% -------------  Graph the Solutions -----------------
```

gs.m

```
function x = gs(A, b, x0, steps, omega)
% SOR smoother
%
% INPUT:   A      matrix
%          b      right hand side
%          x0     initial guess
%          steps  number of iteration steps
%          omega  relaxation parameter
%
% OUTPUT:  x      approximate solution
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net

  n = size(A, 1);
  if omega == 1
    N = tril(A);
    M = -triu(A,1);
```

```
  else
    D = spdiags(spdiags(A,0),0,n,n);
    L = tril(A,-1);
    N = D + omega*L;
    M = (1 - omega)*D - omega*triu(A,1);
    b = omega*b;
  end
  x = x0;
  for k = 1:steps
    x = N\(b + M*x);
  end
```

interpolate.m

```
function w_l = interpolate(w_lm1, parents)
% Interpolation from coarse to fine.
% Uses the algorithm from Jung/Langer: Methode der finiten Elemente
% fuer Ingenieure, Teubner, 2001, p. 279.
%
% INPUT:   w_lm1    vector on coarse grid
%          parents  father-son relation
%
% OUTPUT:  w_l      vector on fine grid
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net

  n_lm1 = length(w_lm1);
  n_l = n_lm1 + size(parents, 2);
  w_l = zeros(n_l,1);
  w_l(1:n_lm1) = w_lm1;
  for i = n_lm1+1:n_l
    idx1 = parents(1, i-n_lm1);
    idx2 = parents(2, i-n_lm1);
    w_l(i) = 0.5*(w_l(idx1) + w_l(idx2));
  end
```

jacobi.m

```
function x = jacobi(A, b, x0, steps, omega)
% Jacobi smoother
%
% INPUT:   A      matrix
%          b      right hand side
%          x0     initial guess
%          steps  number of iteration steps
%          omega  relaxation parameter
%
% OUTPUT:  x      approximate solution
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net

  d = diag(A);
  dinv = 1./d;
  x = x0;
  for k = 1:steps
     x = x + omega*dinv.*(b - A*x);
  end
```

jacobiM.m

```
function [z] = jacobiM(M,b,r)
% do 4 iterations on the system
%    M * z = b

n = size(b,1);
alpha = 0.8;
d = 1 ./ diag(M);
K = speye(n,n) - alpha*diag(d)*M;
db = alpha * d .* b;
z=sparse(n,1);
for i=1:r
    z = K * z + db;
end
```

main_eig.m

```
global a_mat
global B
global M
global C
global ps
global es
global ts
global Bs
global Ms
global Cs
global order_s
global rev_s
global smooth1
global mg_par
global rjacobi

fprintf('\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n');
set(handles.msg1, 'String', 'RUNNING');
pause(0.01)

Re=str2double(get(handles.Re,'String'));
gam=str2double(get(handles.gam,'String'));
Hmax=str2double(get(handles.Hmax,'String'));
max_oseen_iter=str2double(get(handles.max_oseen_iter,'String'));
tol_oseen=str2double(get(handles.tol_oseen,'String'));
tol_gmres=str2double(get(handles.tol_gmres,'String'));
precond_type=str2double(get(handles.precond_type,'String'));
rjacobi=str2double(get(handles.rjacobi,'String'));
%----------------    start: set parameters for the problem ---------------
steps=str2double(get(handles.steps,'String'));
omega=str2double(get(handles.omega,'String'));
tol_mg=str2double(get(handles.tol_mg,'String'));
nu1=str2double(get(handles.nu1,'String'));
nu2=str2double(get(handles.nu2,'String'));
mu=str2double(get(handles.mu,'String'));
smooth_i=str2double(get(handles.smooth_i,'String'));
if smooth_i == 1 ; smooth1 = 'Jacobi'; end;
if smooth_i == 2 ; smooth1 = 'SOR'; end;
mg_par =[steps;omega;tol_mg;nu1;nu2;mu];
%----------------    end: set parameters for the problem ---------------
```

```
%----------------    start: set parameters for biharmonic ----------------
if gam == 0; max_oseen_iter = 1; end;
%----------------    end: set parameters for biharmonic ----------------


%----------------    start: grid information ---------------------------

[ps,es,ts,pms,ems,tms] = get_grids(Hmax);
[order_s,rev_s] = relabel_vectors(pms,ems,tms);
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
p=ps{steps+1};e=es{steps+1};t=ts{steps+1};


axes(handles.axes7);pdemesh(p,e,t);  title('mesh');
[nelem,nb,nn,m,n] = get_parameters(p,e,t);
istart=nn;
maxit_gmres = nn;
set(handles.nelem, 'String',nelem);
set(handles.dof, 'String',n+m);
%----------------    end: grid information  -----------------------------
if precond_type == 1; precond='precond1';
elseif precond_type == 2; precond='precond2';
elseif precond_type == 3; precond='precond3';
elseif precond_type == 4; precond='precond4';
elseif precond_type == 5; precond='precond5';
elseif precond_type == 6; precond='precond6';
elseif precond_type == 7; precond='precond7';
elseif precond_type == 8; precond='precond8';
elseif precond_type == 9; precond='precond9';
elseif precond_type == 10; precond='precond10';
elseif precond_type == 11; precond='precond11';
elseif precond_type == 12; precond='precond12';
elseif precond_type == 13; precond='precond13';
elseif precond_type == 14; precond='precond14';
elseif precond_type == 15; precond='precond15';
else; precond='precond1'; end;


%----------------    start: compute exact solution ---------------------------
[exsol,exsol_psi_bodry] = compute_exact(p,m,nb);
%----------------    end: compute exact solution ---------------------------


%----------------    start: set initial guiss ---------------------------
            % initial guess for the streamfunction ( the interior nodes )
            % the initial vector guiss satisfies the boundary conditions
            % NOTE: for (cavity-flow-problem) change this one
ds=sparse(n,1);
ds(m+1:m+nb,1)=exsol_psi_bodry;
sol_old= sparse(nn,1);
sol = sol_old;
sol_direct = sol_old;
%----------------    end: set initial guiss ---------------------------

[Bs,Ms]=get_BsMs(ps,es,ts);
B=Bs{steps+1};  M=Ms{steps+1};
[rhs]=get_rhs(Re,gam,p,e,t);

precondxx = 'precond1';
```

```
tic;
for i=1:2

   ds(1:m,1)=sol_old(n+1:n+m,1);

   [Cs]=get_Cs(Re,gam,ds,ps,es,ts,Bs,Ms);
   C = Cs{steps+1};
   [a_mat]=get_a(C,B,M,p,e,t);

   [sol,flag,relres,iter,resvec] =  gmres(@afun,rhs,istart,tol_gmres,maxit_gmres,precond,[],[],nn);

   flag;

   gmres_iter(i) = iter(1,2);

   if i==2;
      ploteig;
   end

   diff_norm(i) = norm( sol - sol_old, inf )/nn;
   exact_error(i) = norm( sol - exsol,inf )/nn;
   set(handles.error, 'String', diff_norm');
   set(handles.gmres_iter, 'String', gmres_iter);
   pause(0.001)

   sol_old = sol;

   if diff_norm(i) < tol_oseen ; break; end;
end
runtime = toc

%----------------    start: print error and plot solution ----------------------------
[er1,er2,er3,exact_sol,p_sol,direct_sol] = compute_error(n,nn,nb,sol,sol,exsol,ds,p,t,e);

graph_sol

%----------------    end: print error and plot solution ----------------------------

set(handles.cpu_time, 'String', runtime);
set(handles.exact_sol, 'String', exact_sol);
set(handles.BP_sol, 'String', p_sol);
set(handles.error, 'String', diff_norm');
format short e
set(handles.er2, 'String',er2);
set(handles.msg1, 'String', 'FINISH');
set(handles.gmres_iter, 'String', gmres_iter);
```

main_gui.m

```
global a_mat
global B
global M
global C
global ps
global es
global ts
global Bs
global Ms
global Cs
```

```
global order_s
global rev_s
global smooth1
global mg_par
global rjacobi

fprintf('\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n');
set(handles.msg1, 'String', 'RUNNING');
pause(0.01)

Re=str2double(get(handles.Re,'String'));
gam=str2double(get(handles.gam,'String'));
Hmax=str2double(get(handles.Hmax,'String'));
max_oseen_iter=str2double(get(handles.max_oseen_iter,'String'));
tol_oseen=str2double(get(handles.tol_oseen,'String'));
tol_gmres=str2double(get(handles.tol_gmres,'String'));
precond_type=str2double(get(handles.precond_type,'String'));
rjacobi=str2double(get(handles.rjacobi,'String'));
%----------------    start: set parameters for the problem ---------------
steps=str2double(get(handles.steps,'String'));
omega=str2double(get(handles.omega,'String'));
tol_mg=str2double(get(handles.tol_mg,'String'));
nu1=str2double(get(handles.nu1,'String'));
nu2=str2double(get(handles.nu2,'String'));
mu=str2double(get(handles.mu,'String'));
smooth_i=str2double(get(handles.smooth_i,'String'));
if smooth_i == 1 ; smooth1 = 'Jacobi'; end;
if smooth_i == 2 ; smooth1 = 'SOR'; end;
mg_par =[steps;omega;tol_mg;nu1;nu2;mu];
%----------------    end: set parameters for the problem ---------------


%----------------    start: set parameters for biharmonic ---------------
if gam == 0; max_oseen_iter = 1; end;
%----------------    end: set parameters for biharmonic ---------------


%----------------    start: grid information ---------------------------

[ps,es,ts,pms,ems,tms] = get_grids(Hmax);
[order_s,rev_s] = relabel_vectors(pms,ems,tms);
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
p=ps{steps+1};e=es{steps+1};t=ts{steps+1};


axes(handles.axes7);pdemesh(p,e,t);  title('mesh');
[nelem,nb,nn,m,n] = get_parameters(p,e,t);
istart=nn;
maxit_gmres = nn;
set(handles.nelem, 'String',nelem);
set(handles.dof, 'String',n+m);
%----------------    end: grid information  -----------------------------
if precond_type == 1; precond='precond1';
elseif precond_type == 2; precond='precond2';
elseif precond_type == 3; precond='precond3';
elseif precond_type == 4; precond='precond4';
elseif precond_type == 5; precond='precond5';
elseif precond_type == 6; precond='precond6';
elseif precond_type == 7; precond='precond7';
elseif precond_type == 8; precond='precond8';
```

```
elseif precond_type == 9; precond='precond9';
elseif precond_type == 10; precond='precond10';
elseif precond_type == 11; precond='precond11';
elseif precond_type == 12; precond='precond12';
elseif precond_type == 13; precond='precond13';
elseif precond_type == 14; precond='precond14';
elseif precond_type == 15; precond='precond15';
else; precond='precond1'; end;


%----------------    start: compute exact solution ---------------------------
[exsol,exsol_psi_bodry] = compute_exact(p,m,nb);
%----------------    end: compute exact solution ---------------------------


%----------------    start: set initial guiss ---------------------------
            % initial guess for the streamfunction ( the interior nodes )
            % the initial vector guiss satisfies the boundary conditions
            % NOTE: for (cavity-flow-problem) change this one
ds=sparse(n,1);
ds(m+1:m+nb,1)=exsol_psi_bodry;
sol_old= sparse(nn,1);
sol = sol_old;
sol_direct = sol_old;
%----------------    end: set initial guiss ---------------------------

[Bs,Ms]=get_BsMs(ps,es,ts);
B=Bs{steps+1};  M=Ms{steps+1};
[rhs]=get_rhs(Re,gam,p,e,t);

tic;
for i=1:max_oseen_iter

   ds(1:m,1)=sol_old(n+1:n+m,1);

   [Cs]=get_Cs(Re,gam,ds,ps,es,ts,Bs,Ms);
   C = Cs{steps+1};
   [a_mat]=get_a(C,B,M,p,e,t);

   [sol,flag,relres,iter,resvec] =  gmres(@afun,rhs,istart,tol_gmres,maxit_gmres,precond,[],[],nn);

   flag;

   gmres_iter(i) = iter(1,2);

   %[sol_direct]=solve_direct(a_mat,rhs);

   diff_norm(i) = norm( sol - sol_old, inf )/nn;
   exact_error(i) = norm( sol - exsol,inf )/nn;
   set(handles.error, 'String', diff_norm');
   set(handles.gmres_iter, 'String', gmres_iter);
   pause(0.001)

   sol_old = sol;

   if diff_norm(i) < tol_oseen ; break; end;
end
runtime = toc

%----------------    start: print error and plot solution ---------------------------
```

```
[er1,er2,er3,exact_sol,p_sol,direct_sol] = compute_error(n,nn,nb,sol,sol,exsol,ds,p,t,e);

graph_sol

%----------------    end: print error and plot solution ----------------------------

set(handles.cpu_time, 'String', runtime);
set(handles.exact_sol, 'String', exact_sol);
set(handles.BP_sol, 'String', p_sol);
set(handles.error, 'String', diff_norm');
format short e
set(handles.er2, 'String',er2);
set(handles.msg1, 'String', 'FINISH');
set(handles.gmres_iter, 'String', gmres_iter);
```

main_help.m

```
% Here we slove the Navier-Sokes equations using
% the streamfunction-vorticity formulation

%   laplace( w )

% we solve the problem using FEM with Q1-Q1
% the resulting linear system are solved using GMRES with
% Block Preconditioning technique descriped in the
% Murphy-Golub-Wathen-2000
% if the coeff matrix is
%       A = [ e1 , e2 , e3 ; e4 , e5 , e6 ; e7 , e8 , e9 ];
%  then
%       P = [ diag(e1) , 0 , 0 ; 0 , diag(e5) , 0 ; 0 , 0 , S* ];
%  where
%       S* =  K_mgc * e1 * K_mg
%
%       K_mg = the action of multigrid method on the Laplacian
%           ~= inv(e3)
%
%       K_mgc = the action of multigrid method on
%             the Laplacian+convection
%           ~= inv(e7)

% insert the parameter of the problem where
%   Re            = the Reynolds number
%   np            = number of nodes on the x-axis
%   max_oseen_iter = maximum oseen iterations
```

mg_alg.m

```
function [u_lmax, k, flag] = mg_alg(f_lmax, lmax, matrices, u0_lmax, parents, ...
                                    isdir, nu1, nu2, mu, maxit, tol, smooth, omega)
% Multigrid algorithm.
% Can be used as standalone solver or as preconditioner for the
% Matlab pcg routine (see the pcg help for details).
%
% INPUT:   f_lmax    right hand side on the finest level
%          lmax      maximum level
%          matrices  cell array of matrices, matrices{l+1} contains
%                    the system matrix on level l, l = 0,...,lmax
%          u0_lmax   initial guess
%          parents   cell array containing the father-son relations
%          isdir     dirichlet node flags
%          nu1       number of pre-smoothing steps
```

```
%         nu2       number of post-smoothing steps
%         mu        number of recursive multigrid step calls,
%                   1 for V-cycle, 2 for W-cycle
%         maxit     maximum number of iterations
%         tol       tolerance for stopping criterion
%         smooth    type of smoother,
%                   'Jacobi' for damped Jacobi or 'SOR' for SOR
%         omega     relaxation parameter for the smoother
%
% OUTPUT:  u_lmax    solution on the finest level
%         k         number of performed iterations
%         flag      0 if the algorithm converged, 1 otherwise
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net


  flag = 0;
  for k = 1:maxit
    % perform a multigrid step on level lmax
    u_lmax = multigrid_step(f_lmax, lmax, matrices, u0_lmax, parents, ...
                                    isdir, nu1, nu2, mu, smooth, omega);
    % stopping criterion (relative residual):
    if (norm(matrices{lmax+1}*u_lmax - f_lmax)/norm(f_lmax) < tol)
      return;
    end
    u0_lmax = u_lmax;
  end
  flag = 1;
  return;

function u_l = multigrid_step(f_l, l, matrices, u0_l, parents, isdir, ...
                                      nu1, nu2, mu, smooth, omega)
  A_l = matrices{l+1}; % get system matrix of the current level
  if (strcmp(smooth, 'Jacobi'))
    u_l = jacobi(A_l, f_l, u0_l, nu1, omega); % smooth nu1-times
  else
    u_l = gs(A_l, f_l, u0_l, nu1, omega); % smooth nu1-times
  end
  d_l = f_l - A_l*u_l; % calculate defect
  d_lm1 = restrict(d_l, parents{l+1}, isdir); % restrict defect
  if l == 1     % solve direct on the coarsest level:
    w_lm1 = matrices{l}\d_lm1;
  else          % perform mu multigrid steps
    u0_lm1 = zeros(length(d_lm1), 1);
    for k = 1:mu
      w_lm1 = multigrid_step(d_lm1, l-1, matrices, u0_lm1, parents, ...
                                    isdir, nu1, nu2, mu, smooth, omega);
      u0_lm1 = w_lm1;
    end
  end
  w_l = interpolate(w_lm1, parents{l+1}); % interpolate correction
  u_l = u_l + w_l; % add correction
  if (strcmp(smooth, 'Jacobi'))
    u_l = jacobi(A_l, f_l, u_l, nu2, omega); % smooth nu2-times
  else
    u_l = gs(A_l, f_l, u_l, nu2, omega); % smooth nu2-times
  end
```

mg_B.m

```
function [z2] = mg_B(u);
global Hmax
global ps
global es
global ts
global Bs
global Ms

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

matrices = cell(steps+1,1);
parents = cell(steps+1,1);

p=ps{1};e=es{1};t=ts{1};
B = Bs{1}; M = Ms{1};
[nelem,nb,nn,m,n] = get_parameters(p,e,t);
M22 = speye(nb);
M11=B(1:m,1:m);
M21=sparse(nb,m);
matrices{1}=[M11,M21';M21,M22];

for lmax = 1:steps

  isdir =zeros(1,n); isdir(m+1:m+nb)=1;

  parents{lmax+1} = get_edges(p, t);  % get the father-son information

  p=ps{lmax+1};e=es{lmax+1};t=ts{lmax+1};

  B = Bs{lmax+1};  M = Ms{lmax+1};
  [nelem,nb,nn,m,n] = get_parameters(p,e,t);
  M22 = speye(nb);
  M11=B(1:m,1:m);
  M21=sparse(nb,m);
  matrices{lmax+1}=[M11,M21';M21,M22];

  f_lmax = [u;sparse(nb,1)];  % fix u here
  %f_lmax = sparse(n,1)-1;
  %size(u)
  %size(f_lmax)
  %[u] = restrict_v(f_lmax,lmax+2);
  %u = restrict(f_lmax, parents, isdir)


  u0_lmax = sparse(n,1);

  [u_lmax, iter, flag] = mg_alg(f_lmax, lmax, matrices, u0_lmax, parents, isdir, ...
                        nu1, nu2, mu, maxit, tol, smooth, omega);

  %fprintf('%d \t %d\t %d \n',lmax, dof, iter);
end
[Hmax , norm(f_lmax - matrices{lmax+1}*u_lmax)/m];

z2(1:m,1) = u_lmax(1:m,1);
```

mg_C.m

```
function [z2] = mg_C(u);
global Hmax
```

```
global ps
global es
global ts
global Cs

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

matrices = cell(steps+1,1);
parents = cell(steps+1,1);

p=ps{1};e=es{1};t=ts{1};
C = Cs{1};
[nelem,nb,nn,m,n] = get_parameters(p,e,t);
M22 = speye(nb);
M11=C(1:m,1:m);
M21=sparse(nb,m);
matrices{1}=[M11,M21';M21,M22];

for lmax = 1:steps

  isdir =zeros(1,n); isdir(m+1:m+nb)=1;

  parents{lmax+1} = get_edges(p, t);  % get the father-son information

  p=ps{lmax+1};e=es{lmax+1};t=ts{lmax+1};

  C = Cs{lmax+1};
  [nelem,nb,nn,m,n] = get_parameters(p,e,t);
  M22 = speye(nb);
  M11=C(1:m,1:m);
  M21=sparse(nb,m);
  matrices{lmax+1}=[M11,M21';M21,M22];

  f_lmax = [u;sparse(nb,1)];  % fix u here

  u0_lmax = sparse(n,1);

  [u_lmax, iter, flag] = mg_alg(f_lmax, lmax, matrices, u0_lmax, parents, isdir, ...
                        nu1, nu2, mu, maxit, tol, smooth, omega);

  %fprintf('%d \t %d\t %d \n',lmax, dof, iter);
end
[Hmax , norm(f_lmax - matrices{lmax+1}*u_lmax)/m];

z2(1:m,1) = u_lmax(1:m,1);
```

order_vec.m

```
function [order,rev] = order_vec(p,e,t);

po=p;eo=e;to=t;
[nelem,nb,nn,m,n] = get_parameters(p,e,t);

bo = union( e(1,:),e(2,:));
int = setdiff([1:n],bo);

order=[int,bo];
rev(order)=[1:n];
```

phi_info.m

```
function [phi,phix,phiy] = phi_info(nqpt,qx,qy);

phi1 = inline('1-x-y','x','y');
phi2 = inline('x','x','y');
phi3 = inline('y','x','y');

phi1x = inline('-1','x','y');
phi2x = inline('1','x','y');
phi3x = inline('0','x','y');

phi1y = inline('-1','x','y');
phi2y = inline('0','x','y');
phi3y = inline('1','x','y');

for i=1:nqpt
   x=qx(i);y=qy(i);
   phi(1,i)=phi1( x , y );
   phi(2,i)=phi2( x , y );
   phi(3,i)=phi3( x , y );

   phix(1,i)=phi1x( x , y );
   phix(2,i)=phi2x( x , y );
   phix(3,i)=phi3x( x , y );

   phiy(1,i)=phi1y( x , y );
   phiy(2,i)=phi2y( x , y );
   phiy(3,i)=phi3y( x , y );

end
```

ploteig.m

```
n = size(M,2);
m = size(B,1);
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);
S1 = inv(M) * B';
S  = C * S1;

MM = diag(diag(M));
z1=zeros(n,m);

P=[ MM, z1; z1', S];
v=eig(full(P*a_mat)); w=zeros(length(v));
plot(v,w,'ok')
```

precond1.m

```
function [z]=precond1(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);
```

```
z(1:n,1) = inv(M) * x(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;


% -------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
S1 = inv(M) * B';
S  = C * S1;

z2 = inv(S) * x2;

z(n+1:n+m,1) = z2(:,1);
```

precond2.m

```
function [z]=precond2(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);
z(1:n,1) = inv(M) * x(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;


% -------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
%w = inv(C1) * x(n+1:n+m,1);
[w] = mg_C(x2);
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
u = M1 * w ;

%---- step 3 ----------   B1^{T} * z2 = u
%z2 = inv( B1' ) * u ;
[z2] = mg_B(u);
```

```
z(n+1:n+m,1) = z2(:,1);
```

precond3.m

```
function [z]=precond3(x,nn)
global a_mat;global M;global B;global C;
global p;global e;global t

%  P = [ diag(M) ,  0  ; 0 , S_MG  ]
n = size(M,2);
m = size(B,1);

% ----- solve for z1 ---------
mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

%----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

%--- solve for z2 -------  C1 * w = x2 ; u = M1 * w ; B1^{T} * z2 = u
x2=x(n+1:n+m,1);
M1 =M(1:m,1:m);
[w] = mg_C(x2);
u = M1 * w ;
[z(n+1:n+m,1)] = mg_B(u);
```

precond4.m

```
function [z]=precond4(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);
z(1:n,1) = inv(M) * x(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;


% --------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
w = inv(C1) * x(n+1:n+m,1);
%[w] = mg_C(x2);
%--- end: step 1 -------
```

```
%---- step 2 --------   u = M1 * w
u = M1 * w ;

%---- step 3 ----------   B1^{T} * z2 = u
z2 = inv( B1' ) * u ;
%[z2] = mg_B(u);


z(n+1:n+m,1) = z2(:,1);
```

precond5.m

```
function [z]=precond5(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);

mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% ---------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%-----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
w = inv(C1) * x(n+1:n+m,1);
%[w] = mg_C(x2);
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
u = M1 * w ;

%---- step 3 ----------   B1^{T} * z2 = u
z2 = inv( B1' ) * u ;
%[z2] = mg_B(u);


z(n+1:n+m,1) = z2(:,1);
```

precond6.m

```
function [z]=precond6(x,nn)
global a_mat
global M
global B
```

```
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);

x1=x(1:n,1);
G = cholinc(-M,'0');
y1 = inv(G) * (-x1);
z1 = inv(G') * y1;
z(1:n,1) = z1;

%Aii(1:n,1) = diag(M);
%z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% -------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
%w = inv(C1) * x(n+1:n+m,1);
[w] = mg_C(x2);
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
u = M1 * w ;

%---- step 3 ----------   B1^{T} * z2 = u
%z2 = inv( B1' ) * u ;
[z2] = mg_B(u);


z(n+1:n+m,1) = z2(:,1);
```

precond7.m

```
function [z]=precond7(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);

x1=x(1:n,1);
```

```
G = cholinc(-M,'0');
y1 = inv(G) * (-x1);
z1 = inv(G') * y1;
z(1:n,1) = z1;

%Aii(1:n,1) = diag(M);
%z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% ---------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
w = inv(C1) * x(n+1:n+m,1);
%[w] = mg_C(x2);
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
u = M1 * w ;

%---- step 3 ----------   B1^{T} * z2 = u
z2 = inv( B1' ) * u ;
%[z2] = mg_B(u);


z(n+1:n+m,1) = z2(:,1);
```

precond8.m

```
function [z]=precond8(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t

z=x;
z(1,1) = z(1,1) + 1e-60;
```

precond9.m

```
function [z]=precond9(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t

n = size(M,2);
m = size(B,1);
```

```
mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;


% -------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
S1 = inv(M) * B';
S  = C * S1;

z2 = inv(S) * x2;

z(n+1:n+m,1) = z2(:,1);
```

precond10.m

```
function [z]=precond10(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);

mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% -------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
wa = inv(C1) * x(n+1:n+m,1);
[w] = mg_C(x2);
norm(wa-w)/norm(wa)
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
m1ii(1:m,1) = diag(M1);
```

```
u = m1ii .* w ;

%---- step 3 ----------   B1^{T} * z2 = u
z2a = inv( B1' ) * u ;
[z2] = mg_B(u);
norm(z2a-z2)/norm(z2a)


z(n+1:n+m,1) = z2(:,1);
```

precond11.m

```
function [z]=precond11(x,nn)
global a_mat;global M;global B;global C;
global p;global e;global t;global rjacobi

%  P = [ M_Jk ,  0  ; 0 , S_MG  ]

n = size(M,2);
m = size(B,1);

% ------ solve for z1 --------
[z(1:n,1)] = jacobiM(M,x(1:n,1),rjacobi);
z(1,1) = z(1,1) + 1e-60;

% ---------------------- multi-grid step start ------------
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ---------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

%--- solve for z2 -------  C1 * w = x2; u = M1 * w; B1^{T} * z2 = u
x2=x(n+1:n+m,1);
[w] = mg_C(x2);
u = M1 * w ;
[z(n+1:n+m,1)] = mg_B(u);
```

precond12.m

```
function [z]=precond12(x,nn)

%  P = [ diag(M) ,  B^T  ; 0 , S_MG  ]

global a_mat
global M
global B
global C
global p
global e
global t
global rjacobi


n = size(M,2);
m = size(B,1);

% ---------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);
```

```
%----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- solve for z2  -------  C1 * w = x2 ;  u = M1 * w ; B1^{T} * z2 = u
[w] = mg_C(x2);
u = M1 * w ;
[z2] = mg_B(u);
z(n+1:n+m,1) = z2(:,1);


%---- solve for z1 ----------   M * z1 = x1 - B^T * z2
x1=x(1:n,1);
b1 = x1 - B(1:m,1:n)'*z2;


mii(1:n,1) = diag(M);
z(1:n,1) = b1(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;
```

precond13.m

```
function [z]=precond13(x,nn)

%  P = [ M_Jk ,  B^T  ; 0 , S_MG  ]

global a_mat
global M
global B
global C
global p
global e
global t
global rjacobi


n = size(M,2);
m = size(B,1);

% ---------------------- multi-grid step start ------------
C1 =C(1:m,1:m);
B1 =B(1:m,1:m);
M1 =M(1:m,1:m);

%----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- solve for z2  -------  C1 * w = x2 ;  u = M1 * w ; B1^{T} * z2 = u
[w] = mg_C(x2);
u = M1 * w ;
[z2] = mg_B(u);
z(n+1:n+m,1) = z2(:,1);


%---- solve for z1 ----------   M * z1 = x1 - B^T * z2
x1=x(1:n,1);
b1 = x1 - B(1:m,1:n)'*z2;
```

```
[z(1:n,1)] = jacobiM(M,b1,rjacobi);
z(1,1) = z(1,1) + 1e-60;
```

precond14.m

```
function [z]=precond13(x,nn)
global a_mat
global M
global B
global C
global p
global e
global t


n = size(M,2);
m = size(B,1);
nb=n-m;

mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% ---------------------- multi-grid step start ------------
C1 =C(1:m,1:m); C2 =C(1:m,m+1:m+nb);
B1 =B(1:m,1:m); B2 =B(1:m,m+1:m+nb);
M1 =M(1:m,1:m); M2 =M(m+1:m+nb,m+1:m+nb);

%-----------------    start: Multigrid parameters ----------------------------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
[w] = mg_C(x2);
norm(wa-w)/norm(wa)
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
m1ii(1:m,1) = diag(M1);
u = m1ii .* w ;

%---- step 3 ----------   B1^{T} * z2 = u
[z2] = mg_B(u);
norm(z2a-z2)/norm(z2a)

KK = inv( C2 * inv(M2) * B2' );
r = KK * x2;

z(n+1:n+m,1) = z2(:,1) + r ;
```

precond122.m

```
function [z]=precond11(x,nn)
global a_mat
global M
global B
global C
global p
global e
```

```
global t


n = size(M,2);
m = size(B,1);
nb=n-m;

mii(1:n,1) = diag(M);
z(1:n,1) = x(1:n,1) ./ mii(1:n,1) ;
z(1,1) = z(1,1) + 1e-60;

% ---------- multi-grid step start ----------
C1 =C(1:m,1:m); C2 =C(1:m,m+1:m+nb);
B1 =B(1:m,1:m); B2 =B(1:m,m+1:m+nb);
M1 =M(1:m,1:m); M2 =M(m+1:m+nb,m+1:m+nb);

%----------    start: Multigrid parameters ----------
[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();
b = 'squareb1';

x2=x(n+1:n+m,1);
%--- step 1 -------  C1 * w = x2
wa = inv(C1) * x(n+1:n+m,1);
[w] = mg_C(x2);
norm(wa-w)/norm(wa)
%--- end: step 1 -------

%---- step 2 --------   u = M1 * w
m1ii(1:m,1) = diag(M1);
u = m1ii .* w ;

%---- step 3 ----------   B1^{T} * z2 = u
z2a = inv( B1' ) * u ;
[z2] = mg_B(u);
norm(z2a-z2)/norm(z2a)

KK = inv( C2 * inv(M2) * B2' );
r = KK * x2;

z(n+1:n+m,1) = z2(:,1) + r ;
```

relabel.m

```
function [p,e,t,po,eo,to] = relabel(p,e,t);

po=p;eo=e;to=t;
[nelem,nb,nn,m,n] = get_parameters(p,e,t);

bo = union( e(1,:),e(2,:));
int = setdiff([1:n],bo);

order=[int,bo];
v(order)=[1:n];

t(1:3,1:nelem) = v(t(1:3,1:nelem));
e(1:2,:) = v(e(1:2,:));
p(1:2,:) = p(1:2,order);
p = (p+1)/2;
```

relabel_vector.m

```
function [order,rev] = relabel_vector(pm,em,tm);

[nelem,nb,nn,m,n] = get_parameters(pm,em,tm);

bo = union( em(1,:),em(2,:));
int = setdiff([1:n],bo);

order=[int,bo];
rev(order)=[1:n];
```

relabel_vectors.m

```
function [order_s,rev_s] = relabel_vectors(pms,ems,tms);

[steps,smooth,omega,nu1,nu2,mu,tol,maxit,exact,prec,geom,geomb] = get_MG_info();

order_s = cell(steps+1,1);
rev_s = cell(steps+1,1);

for i=1:steps+1
   [order_s{i},rev_s{i}] = relabel_vector(pms{i},ems{i},tms{i});
end
```

reset2.m

```
gray=imread('gray.jpg');

axes(handles.axes6);
image(gray);
axis off;

axes(handles.axes7);
image(gray);
axis off;

axes(handles.axes8);
image(gray);
axis off;

axes(handles.axes9);
image(gray);
axis off;

set(handles.exact_sol, 'String', ' ');
set(handles.BP_sol, 'String', ' ');
set(handles.error, 'String', ' ');
set(handles.er2, 'String', ' ');
set(handles.msg1, 'String', ' ');
set(handles.cpu_time, 'String', ' ');

set(handles.gmres_iter, 'String', ' ');
set(handles.nelem, 'String', ' ');
set(handles.dof, 'String', ' ');
```

restrict.m

```
function d_lm1 = restrict(d_l, parents, isdir)
% Restriction from fine to coarse.
% Uses the algorithm from Jung/Langer: Methode der finiten Elemente
```

```
% fuer Ingenieure, Teubner, 2001, p. 279.
%
% INPUT:   d_l      vector on fine grid
%          parents  father-son relation
%
% OUTPUT:  d_lm1    vector on coarse grid
%
% VERSION 1.0
% DATE 25.3.2004
% EMAIL bernd@flemisch.net

  n_l = length(d_l);
  n_lm1 = n_l - size(parents, 2);
  d_lm1 = d_l(1:n_lm1);
  for i = n_lm1+1:n_l
    idx1 = parents(1, i-n_lm1);
    idx2 = parents(2, i-n_lm1);
    d_lm1(idx1) = d_lm1(idx1) + 0.5*d_l(i);
    d_lm1(idx2) = d_lm1(idx2) + 0.5*d_l(i);
  end
  [i, j] = find(isdir(1:n_lm1) == 1);
  d_lm1(j) = 0;
```

restrict_v.m

```
function [uc] = restrict_v(u,fine_level);
global order_s
global rev_s
global ps
global es
global ts

j=fine_level;
order = order_s{j};
rev = rev_s{j-1};
[nelemc,nbc,nnc,mc,nc] = get_parameters(ps{j-1},es{j-1},ts{j-1});

um = u(rev,1);
umc = um(1:nc,1);
uc = umc(rev,1);
```

solve_direct.m

```
function [sol_direct]=solve_direct(a_mat,rhs);

ddd=inv(a_mat);
sol_direct = ddd*rhs;
```