# SCILAB FINITE ELEMENT SOLVER FOR STATIONARY AND INCOMPRESSIBLE NAVIER-STOKES EQUATIONS

***Author:*** *Massimiliano Margonari*

***Keywords.*** Scilab; Open source software; Navier-Stokes equations

***Abstract:*** In this paper we show how Scilab can be used to solve Navier-stokes equations, for the incompressible and stationary planar flow. Three examples have been presented and some comparisons with reference solutions are provided.

***Contacts*** m.margonari@openeering.com

## 1. Introduction

Scilab is an open source software package for scientific and numerical computing developed and freely distributed by the Scilab Consortium (see [1]).
Scilab offers a high level programming language which allows the user to quickly implement his/her own applications in a smart way, without strong programming skills. Many toolboxes, developed by the users all over the world and made available through the internet, represent a real opportunity to create complex, efficient and multiplatform applications.
Scilab is regarded almost as a clone of the well-known MATLAB®. The two technologies have many points in common: the programming languages are very similar, they both use a compiled version of numerical libraries to make basic computations more efficient, they offer nice graphical tools. In brief, they both adopt the same philosophy, but Scilab has the advantage of being completely free.
Unfortunately, Scilab is not yet widely used in industrial area where, on the contrary, MATLAB® and MATLAB SIMULINK® are the most known and frequently used. This is probably due to the historical advantage that MATLAB® has over all the competitors. Launched to the markets in the late 70's, it was the first software of its kind. However, we have to recall that MATLAB® has many built-in functions that Scilab does not yet provide. In some cases, this could be determinant. While the number of Scilab users, their experiences and investments have grown steadily, the author of this article thinks that the need to satisfy a larger and more diverse market, also led to faster software developments in recent years. As in many other cases, also the marketing played a fundamental role in the diffusion of the product.
Scilab is mainly used for teaching purposes and, probably for this reason, it is often considered inadequate for the solution of real engineering problems. This is absolutely false, and in this article, we will demonstrate that it is possible to develop efficient and reliable solvers using Scilab also for non trivial problems.
To this aim, we choose the Navier-Stokes equations to model a planar stationary and incompressible fluid motion. The numerical solution of such equations is actually considered a difficult and challenging task, as it can be seen reading [3] and [4] just to provide two references. If the user has a strong background in fluid dynamics he/she can obviously implement more complex models than the one proposed in this document using the same Scilab platform.
Anyway, there are some industrial problems that can be adequately modeled through these equations: heat exchangers, boilers and more just to name a few possible applications.

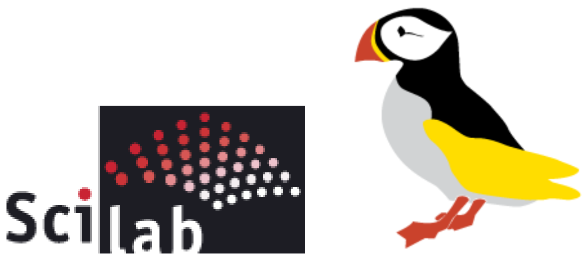

**Figure 1: The Scilab logo (on the left) and the puffin logo (on the right). Dr. Hu Baogang, Contributor Member of Scilab Scientific Board, chose a puffin because "The image of puffin reflects a spirit of freedom with proud, as carried in the endeavor of developing open-source software [...]".**

## 2. The Navier-Stokes equations for the incompressible fluid

Navier-Stokes equations can be derived applying the basic laws of mechanics, such as the conservation and the continuity principles, to a reference volume of fluid (see [2] for more details). After some mathematical manipulation, the user usually reaches the following system of equations:

$$\begin{cases} \nabla \cdot \boldsymbol{U} = 0 \\ -\nabla \cdot (\mu \nabla \boldsymbol{U}) + \rho \boldsymbol{U} \nabla \boldsymbol{U} + \nabla P = 0 \\ -\nabla \cdot (k \nabla T) + \rho c \boldsymbol{U} \nabla T = 0 \end{cases} \quad in\ \Omega \tag{1}$$

which are known as the continuity, the momentum and the energy equation respectively. They have to be solved in the domain $\Omega$, taking into account appropriate boundary conditions. The symbols "$\nabla \cdot$" and "$\nabla$" are used to indicate the divergence and the gradient operator respectively, while $\boldsymbol{U}$, $P$ and $T$ are the unknown velocity vector, the pressure and the temperature fields. The fluid properties are the density $\rho$, the viscosity $\mu$, the thermal conductivity $k$ and the specific heat $c$ which could depend on temperature.
We have to remember that in the most general equations (1), other terms we have neglected in this case (e.g. heat sources, body forces) could be involved. For sake of simplicity we imagine all the fluid properties as constant and we will consider, as mentioned above, only two dimensional domains. The former hypothesis represents a very important simplification because the energy equations completely decouple and therefore, it can be solved separately once the velocity field has been computed using the first two equations. The latter one can be easily removed, with some additional effort in programming.
It is fundamental to note that the momentum equation is non-linear, because of the presence of the advection term $\rho \boldsymbol{U} \nabla \boldsymbol{U}$. Moreover, the correct treatment of this term requires a special attention, as it will be briefly discussed in the following paragraphs, especially when its contribution becomes predominant with respect to the diffusive term.
Another source of difficulty is given by the first equation, which represents the incompressibility condition: in the followings we will discuss also this aspect, even if the interested reader is addressed to the literature (see [2], [3]) for a more detailed discussion on these topics.
For the solution of the equations reported in (1) we decide to use a traditional Galerkin weighted residual approach. As explained above, the energy equation can be solved separately and, for this reason, only the first two equations will be considered in the following. The same procedure identically applies also for the third one.
It is necessary to introduce two virtual fields, $\tilde{P}$ and $\tilde{\boldsymbol{U}}$, which multiply the first and the second equation respectively and integrate them in the domain:

$$\begin{gathered} \int_\Omega \tilde{P} [\nabla \cdot \boldsymbol{U}] \mathrm{d}\Omega = 0 \\ \int_\Omega \tilde{\boldsymbol{U}} [-\nabla \cdot (\mu \nabla \boldsymbol{U}) + \rho \boldsymbol{U} \nabla \boldsymbol{U} + \nabla P] \, d\Omega = 0 \end{gathered} \tag{2}$$

the divergence theorem can be invoked to rewrite the second equation in a more treatable way. It is possible to write:

$$\int_\Omega \tilde{\boldsymbol{U}} \nabla \cdot \mu \nabla \boldsymbol{U} \, d\Omega = \int_\Gamma \tilde{\boldsymbol{U}} \mu \nabla \boldsymbol{U} \cdot \hat{\boldsymbol{n}} d\Gamma - \int_\Omega \nabla \tilde{\boldsymbol{U}} \mu \nabla \boldsymbol{U} \, d\Omega \tag{3}$$

The integral over the boundary vanishes, in view of the incompressibility constraint, and therefore the system (2) can be rewritten as:

$$\begin{aligned} &\int_\Omega \left[\nabla\tilde{\boldsymbol{U}}\mu\nabla\boldsymbol{U} + \rho\tilde{\boldsymbol{U}}\boldsymbol{U}\nabla\boldsymbol{U} + \tilde{\boldsymbol{U}}\nabla P\right] d\Omega = 0 \\ &\qquad\int_\Omega \tilde{P}[\nabla\cdot\boldsymbol{U}]\mathrm{d}\Omega = 0 \end{aligned} \tag{4}$$

where the order of the equation has been changed. To solve numerically the above system it is necessary to introduce a discretization of the domain $\Omega$ and choose some appropriate test functions. It is well known (see [2]) that, in this case, the test functions used for virtual velocity and pressure fields have to be chosen in order to satisfy the *inf-sup* condition (also known as Babuska-Brezzi condition). For this reason we decide to use the six-noded triangular elements depicted in Figure 2; the velocity field is modeled using quadratic shape functions and two unknowns at each node are considered, while the pressure is modeled using linear shape functions and only three unknowns are used at the corner nodes. In this way we have three unknowns in the corner nodes and only two in the element midside nodes.
Equations (4) can be rewritten, once the discretization has been introduced, as the appropriate sum over the elements of certain contributions that can be computed numerically by means of a standard Gauss integration. The single element contribution can be seen, in matrix form, as:

$$\begin{bmatrix} D+C_x & 0 & P_x \\ 0 & D+C_y & P_y \\ I_x & I_y & 0 \end{bmatrix} \cdot \begin{bmatrix} U_x \\ U_y \\ P \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5}$$

where:

- $D = \mu\int_{\Omega_e} Q^t Q d\Omega_e$ is the diffusivity contribution, a (6 x 6) symmetric matrix The matrix $Q$ (2 x 6) collects the first derivatives in the two directions of the quadratic shape functions.
- $C_{(\cdot)} = \rho c\int_{\Omega_e} N U_{(\cdot)} N^t Q d\Omega_e$ is the convective non-linear contribution, a (6 x 6) unsymmetric matrix. The vector $N$ collects the quadratic shape functions.
- $P_{(\cdot)} = \int_{\Omega_e} N^t L_{(\cdot)} d\Omega_e$ is the pressure contribution, a (6 x 3) matrix. The vector $L_{(\cdot)}$ collects the first derivatives of the linear shape functions in the specified direction.
- $I_{(\cdot)} = \int_{\Omega_e} M^t Q_{(\cdot)} d\Omega_e$ is the term coming from the incompressibility condition, a (3 x 6) matrix. The vector $M$ collects the linear shape functions while $Q_{(\cdot)}$ collects the first derivatives of the quadratic shape functions in the specified direction.

All the integrals have to be evaluated on the element $\Omega_e$; these contributions have been always computed using a "master element" (it is common practice in a finite element approach) and a Gauss technique with seven points.
The element matrix is clearly unsymmetric and it contains also the nonlinear terms due to the convection. The peculiar structure of matrix (5), with some zero diagonal terms, suggests interpreting the pressure unknowns as a sort of Lagrangian multipliers which introduce a linear constraint in the model. This constraint is actually given by the incompressibility condition as imposed by the continuity equation.
The element matrix and the known vector, which in our case is always zero, have to be assembled into a global matrix and vector taking into account the applied boundary conditions.

The solution strategy adopted to deal with the nonlinear nature of the equations system is probably the simplest one and it is usually known as the *recursive approach* (or *Picard approach*). An initial guess for the velocity field has to be provided and a first system of linear equations can be assembled and solved. In the element matrix $C_{(\cdot)}$ reported above the term $U_{(\cdot)}$ collects the guess velocity field.
Once the linear system has been solved, the new computed velocity field can be compared with the guessed field. If no significant differences are found, the solution process can be stopped otherwise a new iteration has to be performed using the new computed velocity field.
This process usually leads to the solution in a reasonable amount of iterations and it has the advantage to be very easy to implement. There are more effective techniques, such as the Newton-Raphson scheme, but they usually require to compute the Jacobian of the system and they are harder to implement.
We decide to use the following criterion to stop the iteration process:

$$\frac{\|x_i - x_{i-1}\|}{\|x_i\|} \leq tol = 10^{-8} \qquad (6)$$

where the index *i* represents the iteration step and $x$ is the solution vector, collecting both the velocity and pressure unknowns.
The approach used in this document, that is a standard Galerkin weighted residuals, is not ideally suited for convection dominated problems: it is actually known that when the so-called Peclet number, which expresses the ratio between convective and diffusion contributions, grows the computed solution suffers from a non-physical oscillatory behavior (see [2] for details). The same problem appears also when dealing with the energy equation (the third one in (2)), when the convective contribution is sufficiently high.
This phenomenon can uniquely be ascribed to some deficiency of the numerical technique; for this reason many workarounds have been proposed to correctly deal with convection dominated problems. The most known are surely the streamline upwinding schemes, the Petrov-Galerkin, least square Galerkin approaches and other stabilization techniques.
In this work we do not adopt any of these techniques, knowing that the computed solution with a pure Galerkin approach will be reliable only in the case of diffusion dominated problems. As already mentioned above, it could be in principle possible to implement whatever technique to improve the code and to make the solution process less sensitive to the flow nature, but this is not the objective of the work.

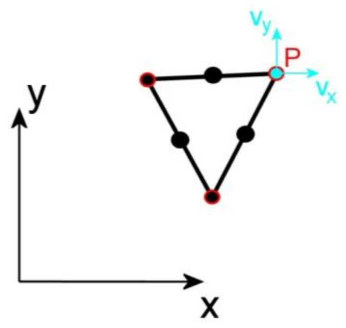

**Figure 2: The six-noded finite element used to discretize the fluid domain. The velocity field is modeled using quadratic shape functions and two unknowns at each black node are considered (Vx and Vy), while the pressure (P) is modeled using linear shape functions and only three unknowns are used at the corner nodes (the red ones).**

## 3. Implementation details

The first step to deal with is surely to define the domain and its discretization. The best would be to have a parametric definition of the geometry in order to allow an easy, may be automatic, modification of the domain. To this aim we decided to use another open source software, *Gmsh* (see [2]), which has been chosen among the many others available because it is really easy to use, powerful, and it can be used also as a postprocessor and a graphical tool to visualize results.

Then a text input file where the user provides all the information needed to define the model has to be organized. We decided to define some *sections* inside which the user specifies the fluid properties and the boundary conditions. In Figure 3 the input file for the channel problem solved in the following is shown. The section *$Fluid*, *$Velocity* and *$Pressure* can be easily recognized. In the first one a list of fluid properties for each fluid domain in the model is provided, while in the last two sections the velocity field in the two directions and the pressure on the boundaries are given.

The Scilab solver has been organized in six files containing functions grouped according to their role. In this way it is extremely easy to add and remove components to the solver leading to an easier software development.

The *main.sce* function is in charge of managing the entire process calling the proper functions when needed. First, it is necessary to read the text file (*.msh) written by *Gmsh* containing the mesh and the input file given by the user. Then, all the data structures have to be organized and the matrices and vectors required for the subsequent numerical solution have to be allocated.

The iterative process described above can now start: the global system of equations is computed and then solved: the best strategy to adopt in this case for the matrix storage is surely the sparse scheme, in order to reduce as much as possible the memory waste during the solution. The *taucs* library can be invoked to solve the linear system by means to an LU decomposition and get a result very fast and easy. It is worth mentioning that the solution of the linear system is invoked in Scilab with just three command lines: with the first one the LU decomposition is computed, with the second command the backward process is perform and finally the memory is cleaned up.

This friendly and easy way of managing the solution of a linear system allows the user to access a very efficient library without spending too much time in developing dedicated code.

At the end of each step the convergence has to be checked and eventually the process has to be iterated. Once the final solution has been found a result file is written and it can be read by *Gmsh*.

```
$Fluid
9 1 1.e-3   !Physical index, density, viscosity
$End

$Velocity
1 1 1 0 0    !Physical index, code x, code y (1=assigned
vel. in dir, 0=unknown vel. in dir), ux, uy
3 1 1 0 0
5 1 1 0 0
6 1 1 0 0
7 1 1 0 0
8 1 1 0 0
4 1 1 0.3 0
$End

$Pressure
2 0       !Physical index, pressure value
$End
```

**Figure 3: The input file used to set up the channel problem used as a benchmark problem. It has a very simple structure: there are some sections where the user can define the fluid properties and the boundary conditions directly on the physical (geometrical) entities defined in the model, and not on the nodes. This obviously simplifies a lot the set up phase, allowing a very compact, clear and easy to change way to specify complex conditions also on different meshes of the same model.**

## 4. Benchmark computations of a laminar flow around a cylinder

In order to test the solver written with Scilab we decided to solve a simple problem which has been used by different authors (see [3], [6]) as a benchmark problem. This benchmark can test different numerical approaches for the solution of the incompressible, steady and unsteady, Navier-Stokes equations. In Figure 4 the problem is drawn, where the geometry and the boundary conditions can be found. The fluid density is set to 1 and the viscosity to $10^{-3}$. A parabolic (Poiseulle) velocity field in x direction is imposed on the inlet, as shown in equation (7),

$$\begin{cases} U_x(y) = \frac{4U_m y(h-y)}{h^2} \\ U_y(y) = 0 \end{cases} \tag{7}$$

with $U_m = 0.3$, a zero pressure condition is imposed on the outlet. The velocity in both directions is imposed to be zero on the other boundaries. The Reynolds number is computed as $Re = \overline{U}D/\nu$, where the mean velocity at the inlet ($\overline{U} = 2U_m/3$), the circle diameter $D$ and the kinematic viscosity $\nu = \mu/\rho$ have been used.
In Figure 5 the adopted meshes have been drawn. The first has 809 elements, 1729 nodes, totally 3486 unknowns while the second has 2609 elements, 5409 nodes, totally 11478 unknowns.
The computations can be performed on a common laptop. In our case, the user has to wait around 43 [sec] to solve the first mesh, while the total solution time is around 310 [sec] for the second model; in both cases 17 iterations are necessary to reach the convergence.

The longest part of the solution time is spent to compute the element contributions and fill the matrix: this is mainly due to the fact that the system solution invokes the *taucs*, which is a compiled library, while the matrix fill-in is done directly in Scilab which is interpreted, and not compiled, leading to a less performing run time.
The whole solution time is however always acceptable even for the finest mesh.
The same problem has been solved also with ANSYS-Flotran (2375 elements, 2523 nodes) and results can be compared with the ones provided by our solver. The comparison is encouraging because the global behavior is well captured also with the coarser mesh and the numerical differences registered between the maximum and minimum values are always acceptable, considering that different grids are used by the solvers.
Other two quantities have been computed and compared with the analogous quantities proposed in [6]. The first one is the recirculation length that is the region behind the circle where the velocity along x is not positive, whose expected value is between 0.0842 and 0.0852; the coarser mesh provides a value of 0.0836 and the finer one a value of 0.0846.
The second quantity which can be compared is the pressure drop across the circle, computed as the difference between the pressures in (0.15; 0.20) and (0.25; 0.20); the expected value should fall between 0.1172 and 0.1176. In our case the coarser mesh gives 0.1191 while the finer gives 0.1177.

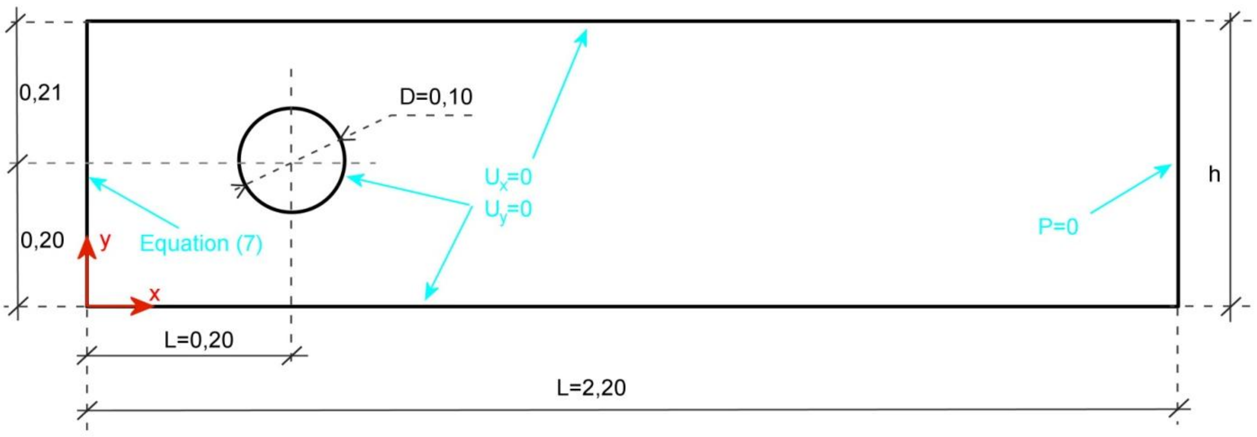

**Figure 4: The benchmark problem of a laminar flow around a cylinder used to test our solver; the boundary conditions are drawn in blue. The same problem has been solved using different computational strategies in [6]; the interested reader is addressed to this reference for more details.**

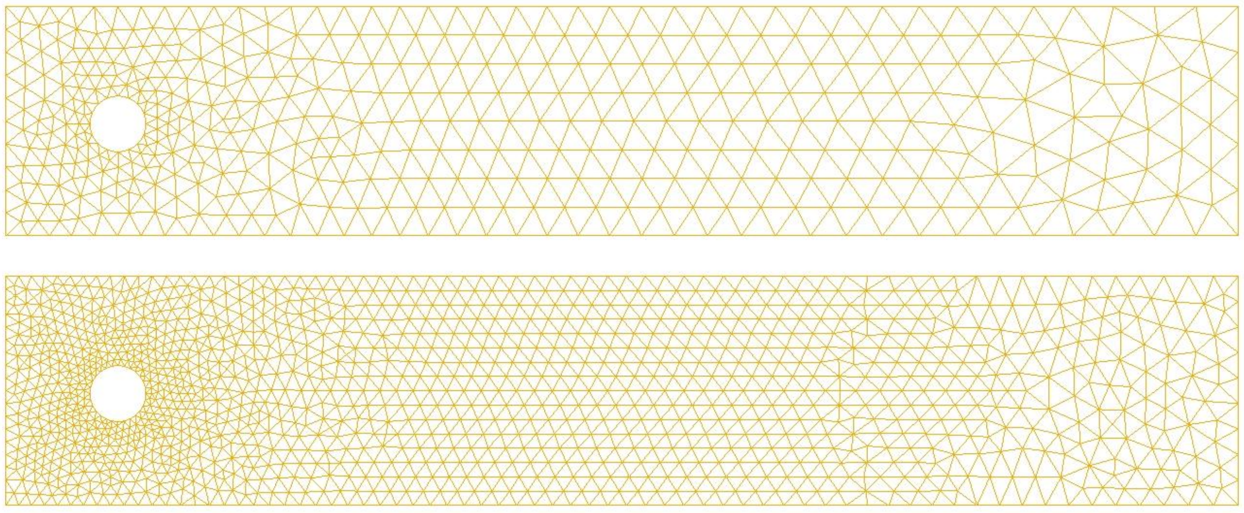

**Figure 5: The two meshes used for the benchmark. On the top the coarse one (3486 unknowns) and on the bottom the finer one (11478 unknowns).**

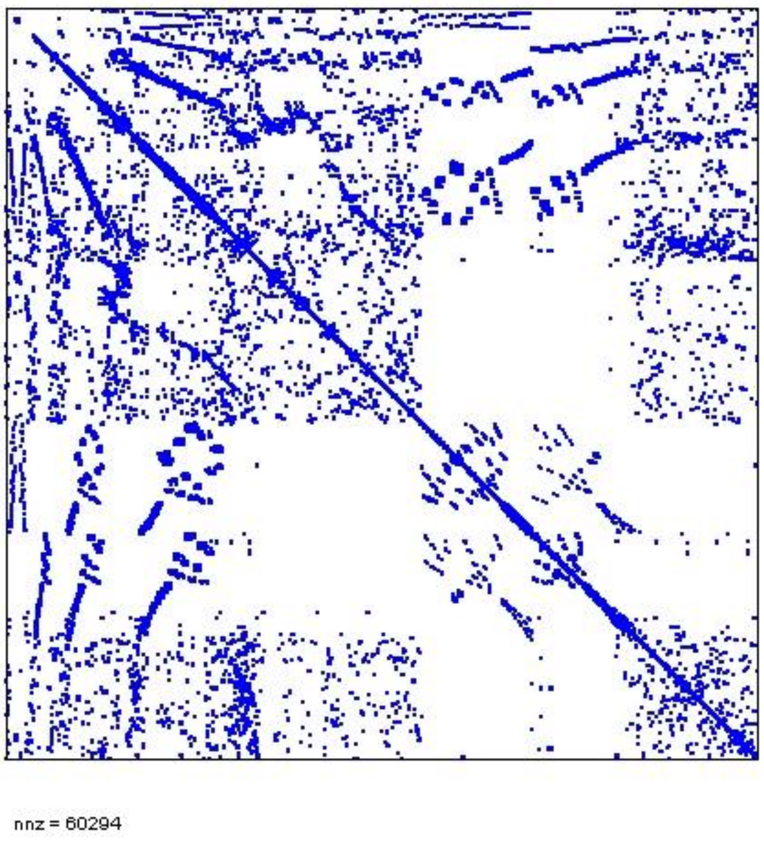

**Figure 6: The sparsity pattern of the system of linear equations that have to be solved. It has to be noted that the pattern is symmetric with respect to the diagonal, but unfortunately the matrix is not. The non-zero terms amount to 60294, leading to a storage requirement of 60294x(8+2*4) = 965 Kbytes when double precision is used. If a full square matrix were used, 11478*11478*8 = 1053956 Kbytes would have be necessary!**

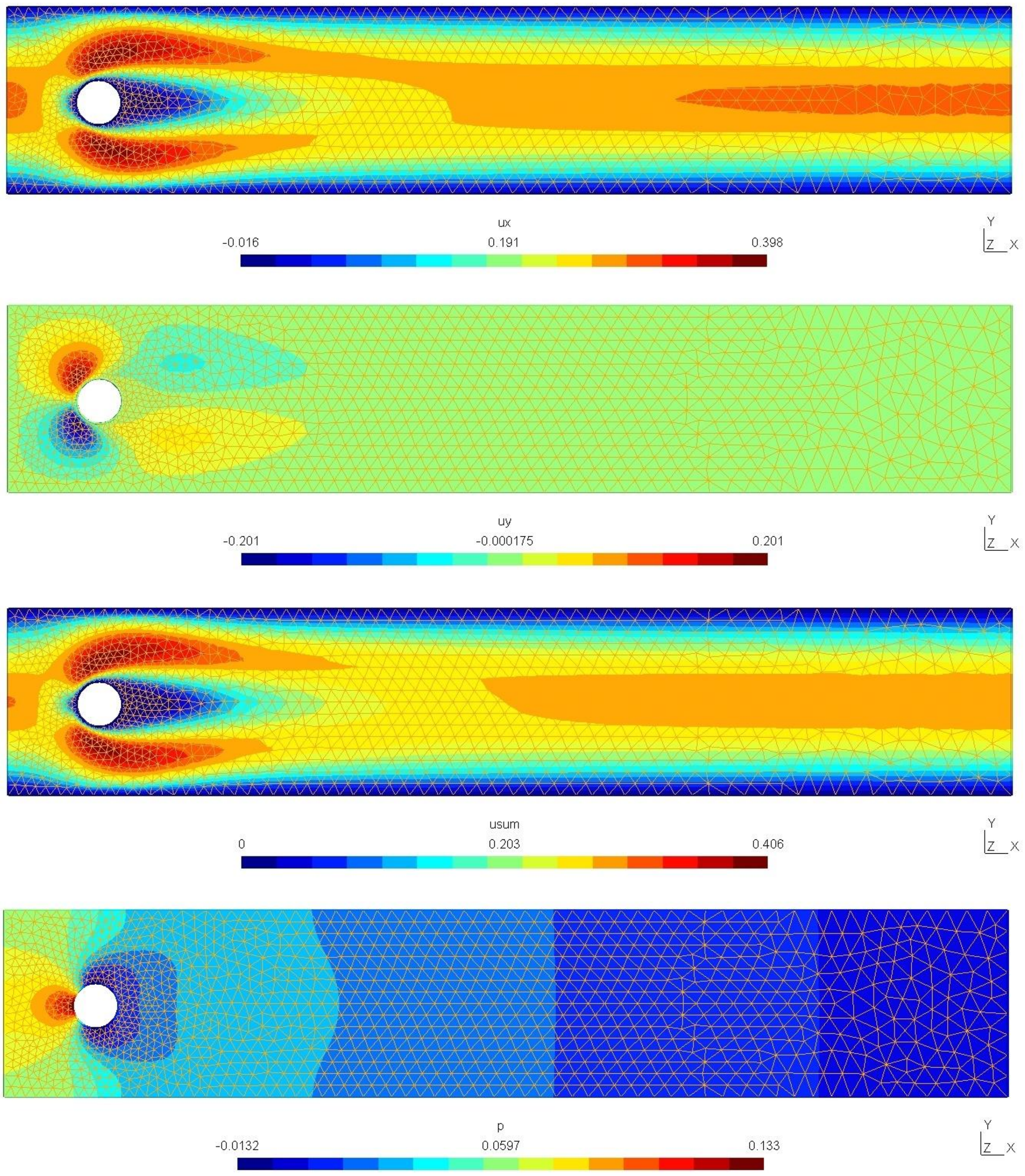

**Figure 7: Starting from top, the x and y components of velocity, the velocity magnitude and the pressure for Reynolds number equal to 20, computed with the finer mesh.**

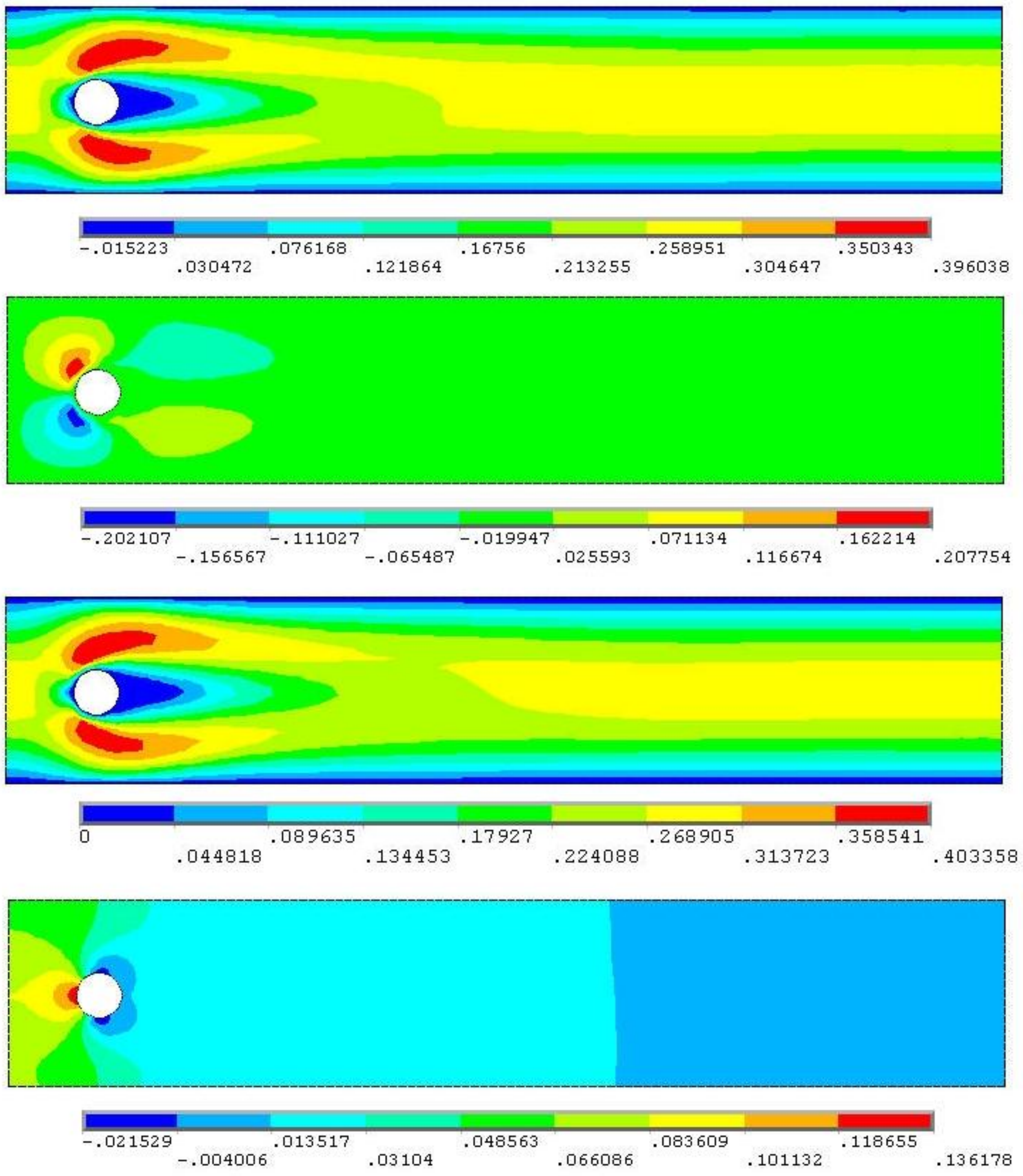

**Figure 8: Starting from top, the x and y components of velocity, the velocity magnitude and the pressure for Reynolds number equal to 20, computed with the ANSYS-Flotran solver (2375 elements, 2523 nodes).**

## 5. Cavity flow problem

A second standard benchmark for incompressible flow is considered in this section. It is the flow of an isothermal fluid in a square cavity with unit sides, as schematically represented in Figure 9; the velocity field has been set to zero along all the boundaries, except than the upper one, where a uniform unitary horizontal velocity has been imposed. In order to make the problem solvable a zero pressure has been imposed to the lower left corner of the cavity.
We would like to guide the interested reader to [3], where the same benchmark problem has been solved. Some comparisons between the position of the main vortex obtained with our solver and the analogous quantity computed by different authors and collected in [3] have been done and summarized in Table 1. In Figure 10 the velocity vector (top) and magnitude (bottom) are plotted for three different cases; the Reynolds number is computed as the inverse of the kinematic viscosity, being the reference length, the fluid density and the velocity all set to one. As the Reynolds number grows the center of the main vortex tends to mode trough the center of the cavity.

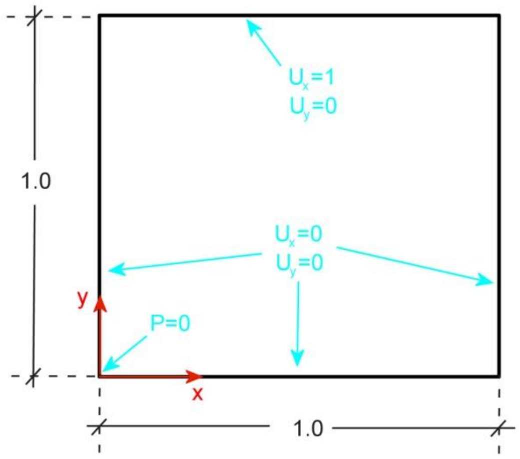

**Figure 9: The geometry and the boundary conditions of the second benchmark used to test the solver.**

| Reynolds Number | Author | x | y |
|---|---|---|---|
| 100 | Solver proposed in [3] | 0.62 | 0.74 |
| | Burggraf (1996) | 0.62 | 0.74 |
| | Tuann and Olson (1978) | 0.61 | 0.722 |
| | **Scilab solver** | **0.617** | **0.736** |
| 400 | Solver proposed in [3] | 0.568 | 0.606 |
| | Burggraf (1996) | 0.560 | 0.620 |
| | Tuann and Olson (1978) | 0.506 | 0.583 |
| | Ozawa (1975) | 0.559 | 0.614 |
| | **Scilab solver** | **0.558** | **0.606** |
| 1000 | Solver proposed in [3] | 0.540 | 0.573 |
| | Ozawa (1975) | 0.533 | 0.569 |
| | Goda (1979) | 0.538 | 0.575 |
| | **Scilab solver** | **0.534** | **0.569** |

**Table 1: The results collected in [3] have been reported here and compared with the analogous quantities computed with our solver (Scilab solver). A satisfactory agreement is observed.**

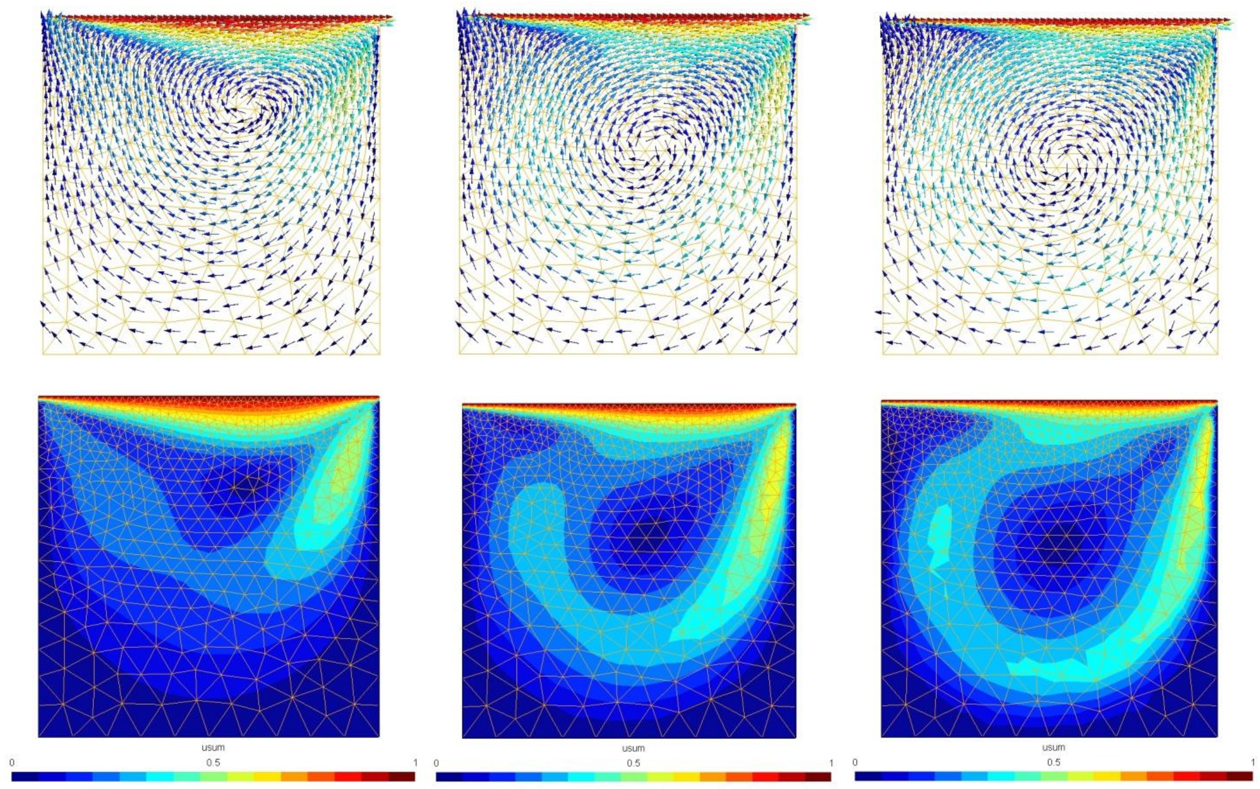

**Figure 10: The velocity vector (top) and the velocity magnitude (bottom) plotted superimposed to the mesh for $\mathrm{Re} = 100$ (left), for $\mathrm{Re} = 400$ (center) and for $\mathrm{Re} = 1000$ (right). The main vortex tends to the center of the cavity as the Reynolds numbers grows and secondary vortexes appear.**

## 6. Thermo-fluid simulation of an heat exchanger

The solver has been tested and it has been verified that it provides accurate results for low Reynolds numbers. A new problem, may be more interesting from an engineering point of view, has been considered: let us imagine that a warm water flow (density of 1000 [Kg/m3], viscosity of 5·10-4 [Pa s], thermal conductivity 0.6 [W/m°C] and specific heat 4186 [J/Kg°C]) with a given velocity enters into a sort of heat exchanger where some hot circles are present. We would like to compute the outlet fluid temperature imaging that the flow is sufficiently low to allow a pure Galerkin approach.
In Figure 11 the mesh for this model is drawn, together with some dimensioning: we decided to consider only the upper part of this heat exchanger in view of the symmetry with respect to the x-axis. The mesh contains 10673 nodes, leading to 22587 velocities and pressures nodal unknowns and 10302 nodal temperatures unknowns.
The symmetry conditions are simply given by imposing homogeneous vertical velocity and thermal flux on the boundaries lying on the symmetry axis. The horizontal inlet velocity follows a parabolic law which goes to zero on the boundary and assume a maximum value of 1·10-3 [m/s] on the symmetry axis. The inlet temperature is 20 [°C] and the temperature of the circle surfaces has been set to 50 [°C]. The outlet pressure has been set to zero in order to get a unique solution. As explained above, the velocity and pressure fields can be computed first and then the energy equation can be tackled in a second phase to compute the temperature in each point.
In Figure 12 the fluid velocity magnitude and in Figure 13 the temperature field are drawn.

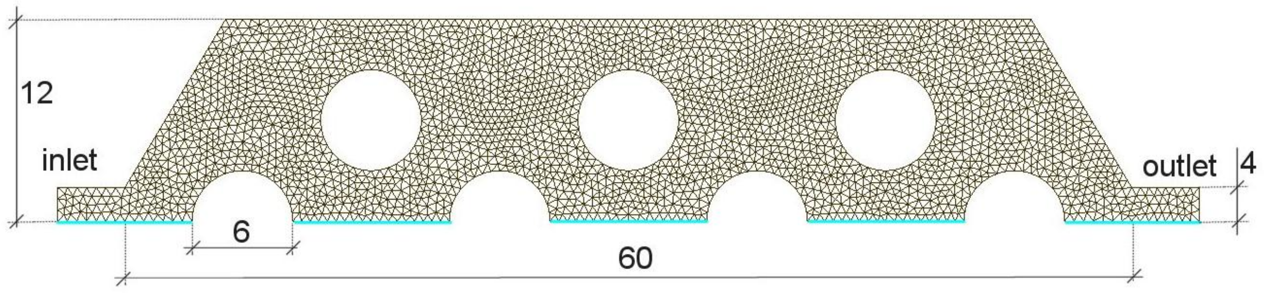

**Figure 11: The heat exchanger considered in this work. The symmetry axis is highlighted in blue and some dimensioning (in [cm]) is reported.**

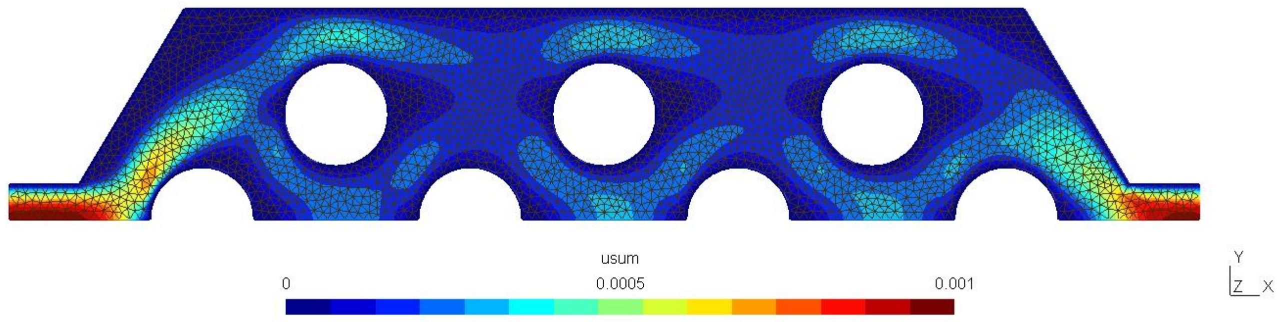

**Figure 12: The velocity magnitude plotted superimposed to the mesh.**

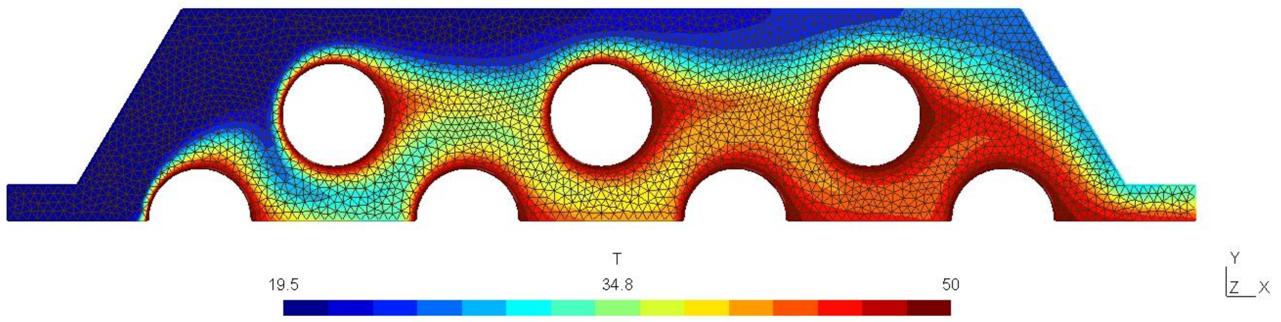

**Figure 13: The temperature field. It can be seen that the inlet temperature is 20 [°C], the circles temperature is 50 [°C], while the outlet temperature vary from a minimum of 32.60 [°C] up to a maximum of 44.58 [°C].**

## 7. Conclusions

In this work we have shown how to use Scilab to solve complex problems in an efficient manner. In order to convince the reader that this is feasible, a solver for the Navier-Stokes equations for the incompressible and stationary flow has been implemented using the standard tools provided with the Scilab distribution. Two examples have been proposed and some comparisons with results provided by commercial software and available in the literature have been done in order to test the solver.
It is worth mentioning that a certain background in finite element analysis is obviously mandatory, but no advanced programming skills are necessary to implement the solver.

## 8. References

[1] *http://www.scilab.org/* to have more information on Scilab
[2] The Gmsh can be freely downloaded from: *http://www.geuz.org/gmsh/*
[3] J. Donea, A. Huerta, *Finite Element Methods for Flow Problems*, (2003) Wiley
[4] J. H. Ferziger, M. Peric, *Computational Methods for Fluid Dynamics*, (2002) Springer, third edition
[5] R. Rannacher, *Finite Element Methods for the Incompressible Navier-Stokes Equations*, (1999) downloaded from *http://ganymed.iwr.uni-heidelberg.de/Oberwolfach-Seminar*
[6] M. Schafer, S. Turek, *Benchmark Computations of laminar Flow Around a Cylinder,* downloaded from *http://www.mathematik.uni-dortmund.de/de/personen/person/Stefan+Turek.html*