Project SG2212/SG3114

Development of a Navier–Stokes solver as a demonstration of concepts

due 23/03/2016

1 Background

The velocity and pressure fields for an incompressible flow can be obtained by solving the Navier–Stokes equations numerically. Under the assumption of constant density (incompressible), the non-dimensional mass and momentum conservation equations are reduced to

$$\frac{\partial u_i}{\partial x_i} = 0 \tag{1}$$

$$\frac{\partial u_i}{\partial t} + \frac{\partial (u_i u_j)}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{1}{Re}\frac{\partial^2 u_i}{\partial x_j \partial x_j} + f_i \ , \tag{2}$$

with u_i denoting the velocity in the direction x_i, P the pressure, and the Reynolds number Re. f_i is a volume force. Note that the summation convention has been used. The corresponding transport equation for a scalar θ such as the temperature or pollutant concentration is

$$\frac{\partial \theta}{\partial t} + \frac{\partial (u_j \theta)}{\partial x_j} = \frac{1}{Pe}\frac{\partial^2 \theta}{\partial x_j \partial x_j} \ , \tag{3}$$

with the Péclet number $Pe = Pr \cdot Re$ and the Prandtl number Pr. In the case of a so-called *active scalar*, the Boussinesq approximation can be used to model the influence of variable density onto the momentum equations by setting $f_2 = Ri\theta$. Here, we assume that the gravity acts in the $y = x_2$ direction, hence the force is applied onto the second component of f_i. Finally, Ri denotes the Richardson number.

As can be seen in the above equations, the continuity equation is not an evolution equation anymore as in the case of compressible fluids. This means that mass conservation becomes like an additional constraint for the momentum conservation equations. This requires extra attention when the equations are solved numerically. For time-dependent flows a common way to solve the discrete system is the so-called projection or pressure correction method. Sometimes this method and its generalisations are also called splitting methods. A short description of this method is given next.

Let us discretise the above equations using an explicit scheme for time derivatives (explicit Euler)

$$\frac{U^{n+1} - U^n}{\Delta t} + N(U^n)U^n + GP^{n+1} = f(t^n) \ , \tag{4}$$

$$DU^{n+1} = 0 \ , \tag{5}$$

where $N(U)U$ denotes the advection and viscous terms, GP the pressure gradient, Δt the time step and $f(t)$ the forcing terms including the boundary conditions. Further, D stands for divergence operator. Note that the pressure term is discretised using the values at the time step $n+1$. The reason for this will become clear later on. Applying the projection method to the set

of equations above, one can rewrite them as follows. First, in the prediction step an intermediate flow field U^* is computed which does not necessarily satisfy the continuity equation,

$$\frac{U^* - U^n}{\Delta t} + N(U^n)U^n = f(t^n) \ . \tag{6}$$

A correction step is then introduced in order to correct for the continuity equation,

$$\frac{U^{n+1} - U^*}{\Delta t} + GP^{n+1} = 0 \ . \tag{7}$$

Applying the divergence operator D to equation (7) and using the divergence-free condition (5), one can derive the following Poisson equation for the pressure,

$$DGP^{n+1} = \Delta P^{n+1} = \frac{1}{\Delta t}(DU^*) \ , \tag{8}$$

using the symbol Δ for the Laplace operator. The computed pressure from this step can be inserted in equation (7) to obtain U^{n+1},

$$U^{n+1} = U^* - \Delta t GP^{n+1} \ . \tag{9}$$

As will be discussed later, in equation (8) homogeneous Neumann boundary conditions are employed. Note that the pressure is only determined up to an additive constant (only pressure derivatives enter the evolution equations). This means that the pressure must be set explicitly in one point of the domain to avoid singular expressions.

The solution of the scalar transport equation (3) is simpler than the momentum equations, as the scalar θ does not need to satisfy the divergence-free condition. Therefore, the integration can be done in a straight-forward way using explicit time integration (velocity field from the previous time step).

2 Simulation code

Your task for this project is to develop a simulation programme that can solve the Navier–Stokes equations in a rectangular domain. This code should be written in MATLAB, based on the templates given on the course homepage and in Appendix B (`SG2212_template.m`, `DD_template.m`, `avg_template.m`). To summarise, the features of the code are:

- Incompressible Navier–Stokes plus scalar equations (Boussinesq approximation; only for PhD students reading course SG3114),
- Second-order finite difference on a staggered grid,
- Explicit Euler time integration for both advection and viscous terms,
- Sparse matrices for the Poisson equation for the pressure correction,
- Dirichlet and Neumann boundary conditions on a rectangular domain in two dimensions (2D).

As this code is mainly used for educational purposes demonstrating the various concepts used in CFD, the complexity of the numerical method is intentionally kept on a low level. Therefore, a number of features common in CFD software are not treated in the code:

- 3D formulation and implicit time stepping for viscous terms,
- CFL condition for time stepping,
- Higher order time integration methods,
- Higher order spatial discretisation,
- more efficient solution strategies, *e.g.* using a Cholesky decomposition,
- More complex geometries, multiblock distributions, mappings *etc.*

In order to demonstrate the capabilities of your code we will consider two physical flow cases which are introduced in the following Sections 3 and 4. A detailed description of the numerical method and its implementation is given in the Appendix.

3 Task 1: Lid-driven cavity (all course participants)

Consider the incompressible flow in a two-dimensional rectangular domain with side length l_x and l_y. The side and lower walls are all solid and still, requiring no-slip boundary conditions. The top wall is moved with a constant speed $u = 1$ in the positive x-direction. This flow case is a common test problem for Navier–Stokes solvers, and usually called the *lid-driven cavity*. A sketch is given in Figure 1.

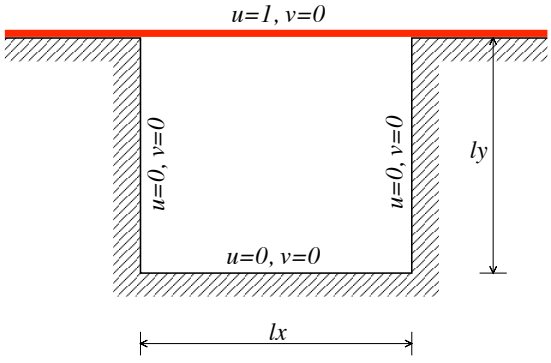

Figure 1: Sketch of the lid-driven cavity. The moving top wall is indicated in red.

The mass and momentum conservation equations in non-dimensional form can be written as

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \,, \tag{10}$$

$$\frac{\partial u}{\partial t} + \frac{\partial P}{\partial x} = -\frac{\partial (u^2)}{\partial x} - \frac{\partial (uv)}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \,, \tag{11}$$

$$\frac{\partial v}{\partial t} + \frac{\partial P}{\partial y} = -\frac{\partial (uv)}{\partial x} - \frac{\partial (v^2)}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \,, \tag{12}$$

with associated boundary conditions

$$x = 0: \quad u = v = 0 \tag{13}$$
$$x = l_x: \quad u = v = 0 \tag{14}$$
$$y = 0: \quad u = v = 0 \tag{15}$$
$$y = l_y: \quad u = 1, v = 0. \tag{16}$$

You can optionally include also the advection of a passive scalar, governed by

$$\frac{\partial \theta}{\partial t} = -\frac{\partial (u\theta)}{\partial x} - \frac{\partial (v\theta)}{\partial y} + \frac{1}{Pe}\left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2}\right) , \tag{17}$$

with $Pe = Re \cdot Pr$. Choose $Pr = 0.71$ which is a good approximation for air. The boundary conditions for the temperature θ are

$$x = 0: \quad \partial\theta/\partial x = 0 \quad \text{(adiabatic)} \tag{18}$$
$$x = l_x: \quad \partial\theta/\partial x = 0 \quad \text{(adiabatic)} \tag{19}$$
$$y = 0: \quad \theta = 0 \tag{20}$$
$$y = l_y: \quad \theta = 1 \ . \tag{21}$$

Initial conditions at $t = 0$ are non-moving fluid, *i.e.* $u(t = 0) = v(t = 0) = 0$. For the scalar, assume a linear profile in the vertical direction, *i.e.* $\theta(t = 0) = y/l_y$.

Your task is to write a MATLAB code which solves the Navier–Stokes equations for the flow case described above using the projection method on a staggered grid. The implementation of the scalar equation is only compulsive for PhD students (course SG3114). To check your implementation, answer Q1–Q5 and run your code for cases A–C given below.

Questions:

Q1) Discuss why the `kron` operator (Kronecker tensor product) as introduced in Section A.4 indeed returns the matrix for the second derivative in the two-dimensional case.

Q2) For case A (spatial discretisation as stated below) determine "experimentally" the maximum time step Δt that is stable for the velocity field. Can you relate that maximum time step to the grid resolution and Reynolds number by considering stability conditions for both convective and diffusive terms (consider a 2D problem!). Which of the two is more strict for cases A–C?
For PhD students: Choosing $Pr = 0.71$, is the scalar equation less stable than the momentum equations? Why?

Q3) The boundary conditions for U on the top and bottom boundaries as well as for V on the left and right boundaries should be imposed by choosing the correct values for the dummy cells (cells outside the domain). Write down the expressions for $U_{i+\frac{1}{2},0}$ and $V_{0,j+\frac{1}{2}}$ at the dummy cells.

Also the expressions for the pressure at the dummy cells $P_{0,j}$ and $P_{i,0}$ should be chosen such that the Neumann boundary condition at the boundaries is satisfied. Give the expression for the pressure in the dummy cells and write down the discretised $\nabla^2 P_{1,j}$ for $j = 2, \cdots, N_y - 1$ (Laplace operator including boundary conditions).

Q4) Place a probe at the domain centre to compare the time history of the horizontal velocity(u) for the flow cases A–C given below. Try to estimate the time it takes for each run to establish a stationary solution. How does this time correlate with the Reynolds number?

Q5) Implement also a moving lower wall using appropriate boundary conditions. Try (at least) three cases ($Re = 100$) with the lower wall moving with the same velocity and same direction as the upper one, with the opposite velocity, and one with a lower velocity at the lower wall. Report plots of your results, and in particular discuss the symmetry properties of the resulting flow patterns.

Flow cases:

A) Run the code for $Re = 10$, $l_x = l_y = 1$. Use a total of $N_x = N_y = 30$ grid cells and a time step $\Delta t = 0.001$.

For low values of Re, one obtains a flow field which is (nearly) symmetric in the x-direction, see Figure 2, with very little transient behaviour.

B) Run the code for $Re = 100$, $l_x = l_y = 1$, $N_x = N_y = 50$, $\Delta t = 0.001$.

For these moderate values of Re (≈ 100) the flow becomes asymmetric due to action of inertia, but it is still steady, see Figures 3. This can be clearly seen by means of a time series recorded in the middle of the domain, see Q4.

C) (optional) Run the code for $Re = 8000$ or even higher Re, $l_x = l_y = 1$, $N_x = N_y = 100$, $\Delta t = 0.001$.

Note also that increasing the Reynolds number will give rise to thinner shear layers, which require more grid points to properly resolve them. You can experiment with the combinations Re, N_x, N_y and Δt, and observe how the results (and stability) are affected (see Q2).

You should hand in the MATLAB code and a report with answers to questions Q1–Q5 together with a discussion and plots illustrating the flow development for the three cases A–C above. In order to give you an idea of how of the flow behaviour, consider Fig. 2 for case A ($Re = 10$), Fig. 3 for case B ($Re = 100$) and Fig. 4 for case C ($Re = 8000$).

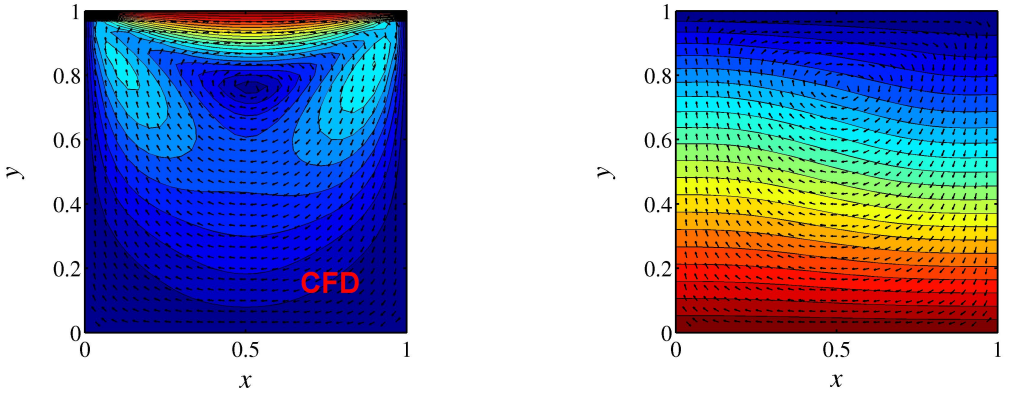

Figure 2: Simulation results for the lid-driven cavity, case A ($Re = 10$). Velocity field *(left)* and temperature field *(right)* at $t = 20$. The colour scale ranges from 0 (blue) to 1 (red).

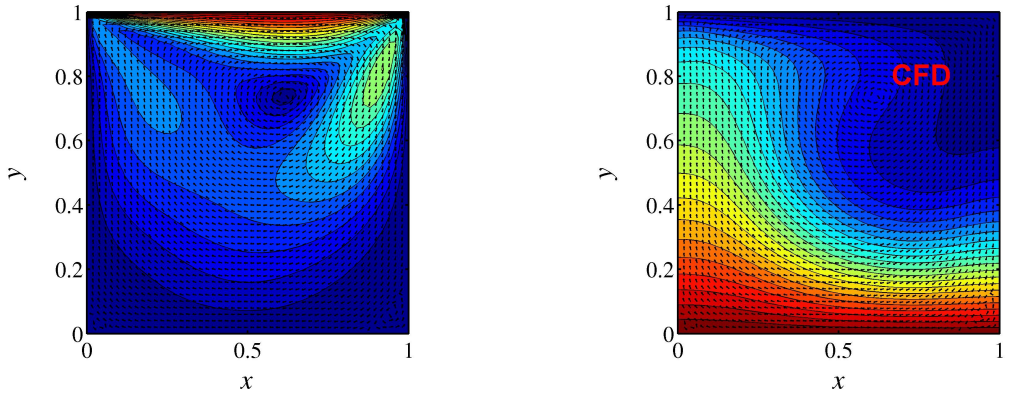

Figure 3: Simulation results for the lid-driven cavity, case B ($Re = 100$). Velocity field *(left)* and temperature field *(right)* at $t = 20$. The colour scale ranges from 0 (blue) to 1 (red).

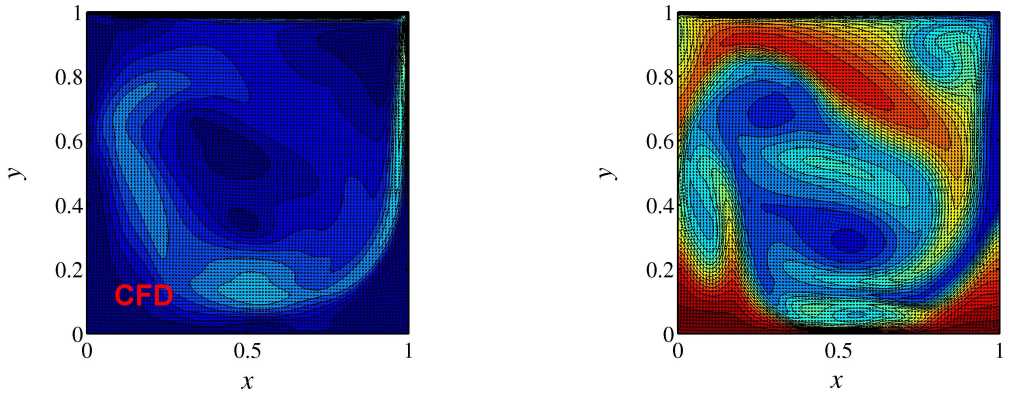

Figure 4: Simulation results for the lid-driven cavity, case C ($Re = 8000$). Velocity field *(left)* and temperature field *(right)* at $t = 20$. The colour scale ranges from 0 (blue) to 1 (red).

A detailed description of the different steps of the algorithm and boundary conditions is given in the Appendix A. A template for the code is available in Appendix B and the course homepage.

4 Task 2: Rayleigh-Bénard problem (only for PhD students SG3114)

The flow induced by small density variations due to temperature differences between two walls is called Rayleigh-Bénard convection. The control parameter is the so-called Rayleigh number Ra which describes the ratio of buoyancy to viscous forces acting on the flow.

Depending on Ra, various flow regimes can be studied, ranging from stable flow configuration for low Ra, two-dimensional convection rolls (*i.e.* large counter-rotating vortices, see below), hexagonal convection cells, and finally for high Ra fully turbulent flow. The major relevance for us to study this flow is that using comparably simple tools from linear stability analysis the changeover from stable to unstable (*i.e.* amplifying) flow can be exactly determined. To numerically check this so-called critical Rayleigh number Ra_c provides a very useful validation of the numerical code.

If the flow field is subjected to a temperature gradient between the two walls, density variation in the flow may be generated. These density variations, due to buoyancy forces, induce motion in the fluid. The corresponding force in the momentum equations can be modelled by means of the Boussinesq approximation. In order to tackle this flow problem, the governing equations (1)-(2) are thus coupled to the temperature equation (3) using the forcing term f_2. Then, a slightly different non-dimensionalisation has to be employed, essentially based on a convective velocity scale. The only non-dimensional numbers in the equations are the Rayleigh number Ra and the Prandtl number Pr. For a two-dimensional flow the final non-dimensional equations can be written as

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \ , \tag{22}$$

$$\frac{\partial u}{\partial t} + \frac{\partial P}{\partial x} = -\frac{\partial (u^2)}{\partial x} - \frac{\partial (uv)}{\partial y} + Pr \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) \ , \tag{23}$$

$$\frac{\partial v}{\partial t} + \frac{\partial P}{\partial y} = -\frac{\partial (uv)}{\partial x} - \frac{\partial (v^2)}{\partial y} + Pr \left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} \right) + f_2 \ , \tag{24}$$

$$\frac{\partial \theta}{\partial t} = -\frac{\partial (u\theta)}{\partial x} - \frac{\partial (v\theta)}{\partial y} + \left(\frac{\partial^2 \theta}{\partial x^2} + \frac{\partial^2 \theta}{\partial y^2} \right) \ . \tag{25}$$

The forcing in the normal direction, caused by buoyancy forces, is

$$f_2 = RaPr\theta \ . \tag{26}$$

The original formulation of the Rayleigh-Bénard convection problem is considered as the flow between two infinite parallel walls, see the sketch in Fig. 6. However, for the present project, we consider a two-dimensional cavity where the top and bottom walls are kept at different temperatures and the side-walls are assumed to be adiabatic ($\frac{\partial \theta}{\partial n} = 0$). A sketch of the geometry is given in Figure 5. Here, the boundary conditions are

$$x = 0: \quad u = v = 0, \quad \frac{\partial \theta}{\partial x} = 0 \ , \tag{27}$$

$$x = l_x: \quad u = v = 0, \quad \frac{\partial \theta}{\partial x} = 0 \ , \tag{28}$$

$$y = 0: \quad u = v = 0, \quad \theta = \theta_B \ , \tag{29}$$

$$y = l_y: \quad u = v = 0, \quad \theta = \theta_T \ . \tag{30}$$

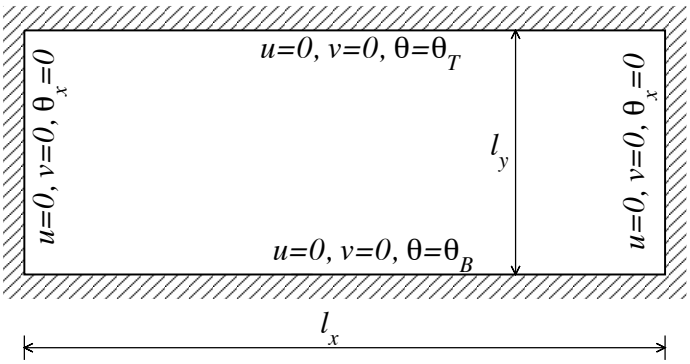

Figure 5: Close cavity with non-isothermal walls.

Initial conditions are at $t = 0$ non-moving fluid, *i.e.* $u(t = 0) = v(t = 0) = 0$. For the scalar, assume a linear profile in the vertical direction,

$$\theta(x, y, t = 0) = \theta_B + \frac{y}{l_y}(\theta_T - \theta_B) \ . \tag{31}$$

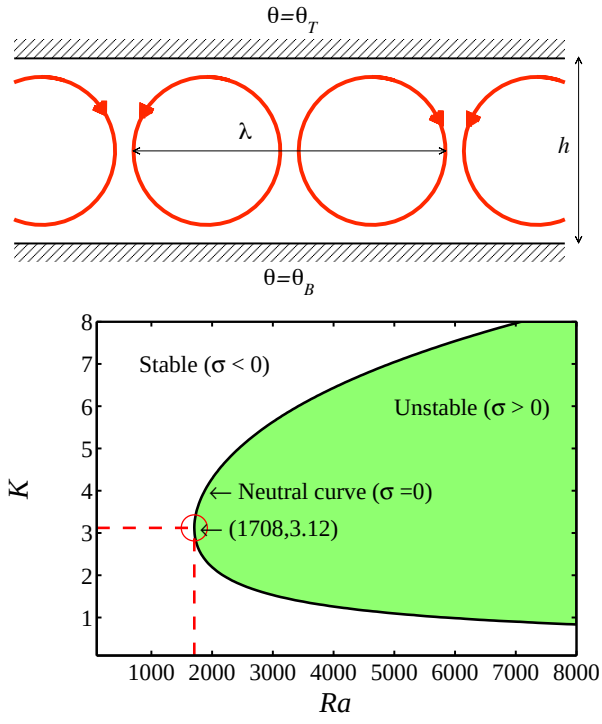

Figure 6: *Top:* Sketch of the flow structure (convection cells). *Bottom:* Stable and unstable (shaded) regions for Rayleigh-Bénard problem. Here, $\lambda = 2\pi/K$ is the wavelength of the unstable perturbation. The critical Rayleigh number is $Ra_c = 1708$ with associated spatial wavenumber $K = 2\pi/\lambda_c = 3.12$ ($\lambda_c = 2.01$).

For cases where $\theta_T > \theta_B$, the density is decreasing with increasing height. This corresponds to a state with is called *stable stratification*, which means that the flow is stable and all disturbances

will decay eventually. However, if $\theta_T < \theta_B$, corresponding to *unstable stratification*, for large enough values of Ra the flow becomes unstable. A stability diagram for this case is shown in Fig. 6. It can be seen that the first instability appears for $Ra_c = 1708$ with a wavenumber $K_c = 3.12$, which corresponds to a wavelength of $\lambda_c = 2.01$.

Your task is to include the forcing f_2 and the energy equation into the code developed in Task 1 and run it for cases A and B given below. Your report should discuss the results for these cases together with answers to questions Q1–Q3.

For all runs, choose the Prandtl number $Pr = 0.71$, the box size $l_x = 10$, $l_y = 1$, and the resolution $N_x = 200$, $N_y = 20$. The domain thus becomes elongated in the x-direction, imitating an infinite domain. As initial conditions random small-scale noise of moderate amplitude ($\pm 0.1(\theta_B - \theta_T)$) should be used in order to trigger the instability. The time step Δt should be chosen such that the time integration remains stable.

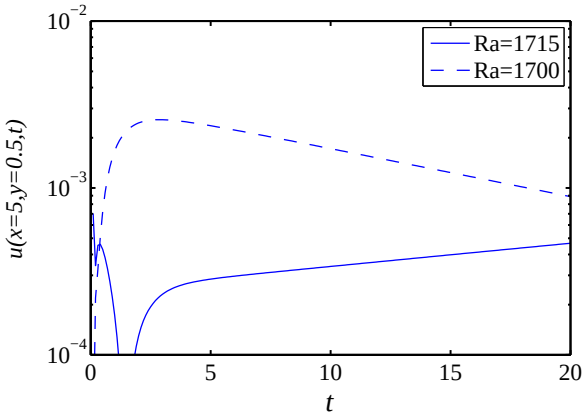

Figure 7: Temporal evolution of the probe signals for two Rayleigh numbers close to the neutrally stable state ($Ra_c = 1708$). It can clearly be seen that the supercritical trajectory leads to exponential growth, whereas the subcritical case eventually will exponentially decay.

Questions:

Q1) Give the definitions of the non-dimensional parameters Pr and Ra both in terms of mathematical symbols and words (physical meaning). Why do the equations not contain any Reynolds number?

Q2) Estimate the wavelength of the instability at $Ra = 1715$. What boundary conditions should be employed (both for velocity and temperature) to exactly reproduce the critical Rayleigh number?

Q3) Based on time signals such as the example given in Figure 7, try to estimate the critical Rayleigh number for your specific discretisation (see description for run B).

Flow cases:

A) Run the code for $Ra = 100, 1000, 50000$, $\theta_T = 1$ and $\theta_B = 0$.

These parameters correspond to stable flow. You should observe that initial disturbances will decay due to viscosity. For the case of $Ra = 50000$ the appropriate time step must be selected.

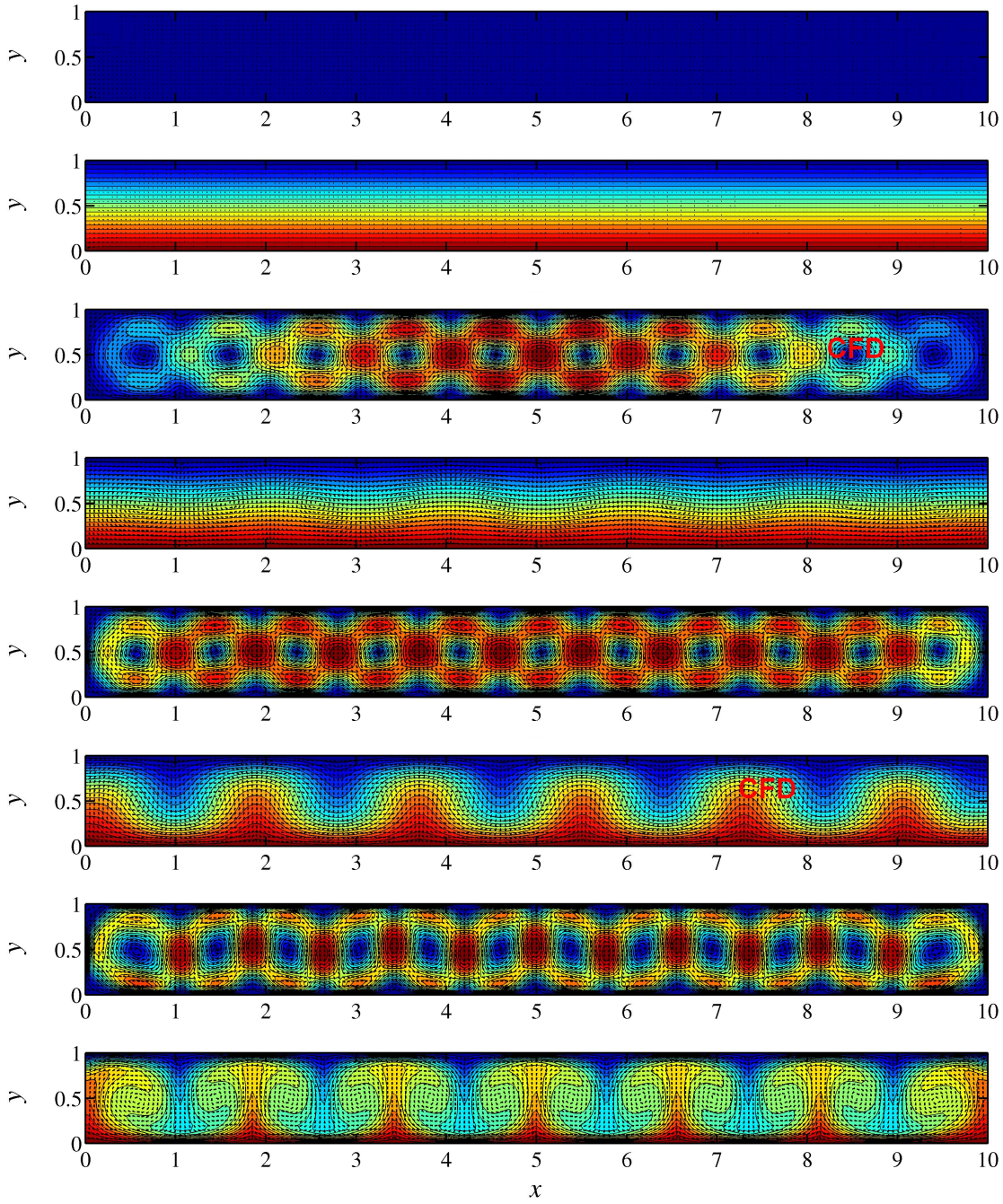

Figure 8: Stationary flow patterns for large times corresponding to $Ra = 1700$, $Ra = 1715$, $Ra = 2700$ and $Ra = 50000$ (from top to bottom). Both the velocity distribution and the temperature field are shown for each Rayleigh number.

B) Run the code for $\theta_T = 0$ and $\theta_B = 1$.

Here, if the value of Rayleigh number is large enough ($Ra > 1708$) flow instability will be observed. Run the code for different values of Ra and try to find "experimentally" the critical value Ra_c. To determine the stability of a specific run, consider the time evolution of a probe in the flow, see Fig. 7. If you observe exponential growth (solid line in Fig. 7), you are in the unstable regime, otherwise the flow is stable.

A Solution procedure and implementation

A.1 Grid

In order to avoid spurious checkerboard solution the equations are discretized on a staggered grid. Figure 9 shows the numerical domain where the pressure p is defined in cell centres and the velocities u and v are defined in the centre of the vertical and horizontal cell faces, respectively. Table 1 gives the number of unknowns for each of the flow variables to solve for. Here, N_x and

Table 1: Resolution for the various solution variables.

Variable	Interiour resolution	Boundary conditions included
P	$N_x \times N_y$	$(N_x + 2) \times (N_y + 2)$
U	$(N_x - 1) \times N_y$	$(N_x + 1) \times (N_y + 2)$
V	$N_x \times (N_y - 1)$	$(N_x + 2) \times (N_y + 1)$

N_y are the number of cells in the x and y directions, respectively. The coordinates of the grid points (cell corners) are given as

$$X_i = i\frac{l_x}{N_x} \ , \quad i = 0, \cdots, N_x \ , \tag{32}$$

$$Y_j = j\frac{l_y}{N_y} \ , \quad j = 0, \cdots, N_y \ , \tag{33}$$

where l_x and l_y are the lengths in x and y directions. Note that number of grid points (including the boundary points) is $N_x + 1$ and $N_y + 1$, respectively.

A.2 Boundary conditions

A.2.1 Velocity

Boundary conditions for both velocity components are given all around the rectangular domain. In the following, these vectors are denoted U_S for U at the lower boundary (south), V_W for V on the left boundary (west), *etc.* Thus, on each edge a total of $(N_x + 1)$ or $(N_y + 1)$ boundary values are specified on the cell corners. The Dirichlet boundary condition for U on the left and the right boundaries of computational domain can be directly applied as the velocity nodes lie on those boundaries, see Fig. 9. The same is valid for V on the top and the bottom boundaries. However, the boundary conditions for U on the top and bottom boundaries should be imposed by choosing the correct values for the dummy cells (cells outside the domain) in such a way that linear interpolation over the boundary will give the correct value at the boundary. The formulation of the boundary conditions is asked in Q3.

A.2.2 Pressure

As it was shown during the lecture, we have the choice of specifying homogeneous Neumann boundary conditions for the pressure. In other words the normal derivative at the boundaries

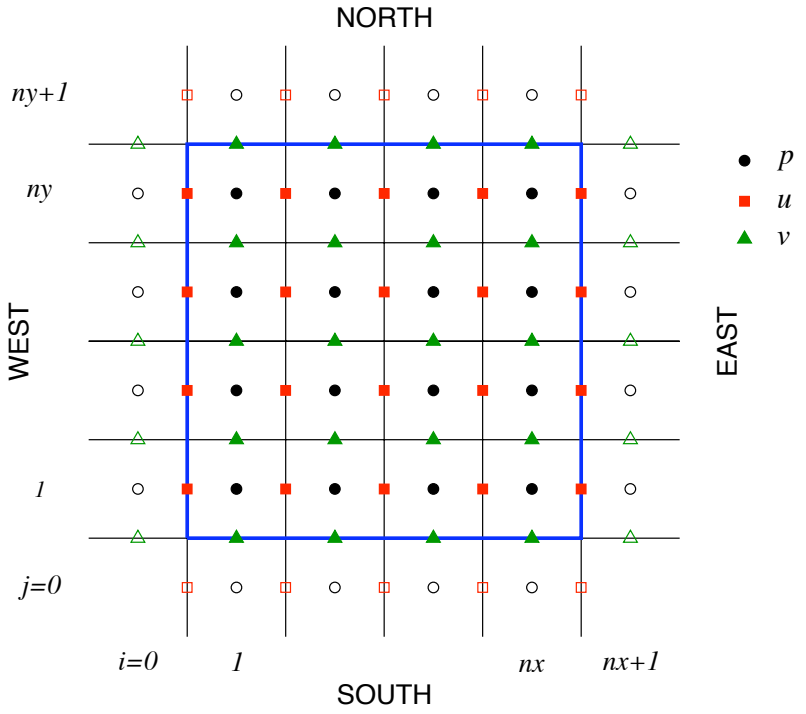

Figure 9: Sketch of the staggered grid. Circles are pressure nodes, squares U-velocity, and triangles V-velocity. Filled symbols correspond to interiour points, whereas open symbols represent the dummy values. The domain boundary is indicated by the thick blue line, on which also the boundary conditions are given.

should vanish for the pressure, *i.e.*,

$$\frac{\partial P}{\partial n} = 0 \ . \tag{34}$$

Again, the value for the pressure dummy cells has to be chosen appropriately to fulfill this condition. The discrete formulation of this boundary condition should also be given as part of Q3.

A.3 Data structure

All variables are stored as matrices in MATLAB. Note that the first index in MATLAB denotes the row and the second index the column, which means that the direction of increasing x is from top to bottom in the matrices. Similarly, increasing the y direction is from left to right in the matrix. Therefore, the storage of variables in the matrices corresponds to the *transpose* of the "physical" domain.

We introduce matrices `U` and `V` which only contain the unknown velocity values at the inner cells, and not the (known) boundary values. Thus, the size for the matrix `U` containing the u velocity is $(N_x - 1) \times N_y$. Similarly, `V` containing the v velocity has $N_x \times (N_y - 1)$ elements.

At time $t = 0$ the velocities can be initiated with zeros (*i.e.* fluid at rest) in MATLAB as

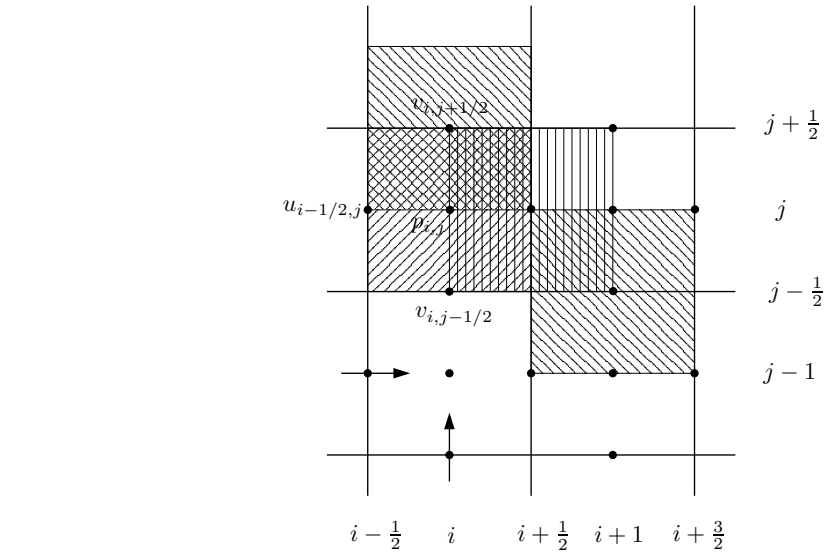

Figure 10: Sketch of the staggered grid. Open symbols indicate dummy cells situated outside the computational domain.

```
U=zeros(nx-1,ny);
V=zeros(nx,ny-1);
```

To perform central difference over the inner points, we need to include the boundary points in `U` and `V`. These *extended* matrices are called `Ue` and `Ve`. They can be created by adding the appropriate rows and columns to the `U` and `V` matrices. For example for the u velocity this is done as follows:

1) Add vectors `uE` and `uW` containing the boundary values at the right and left boundaries of the physical domain, respectively (transposed in MATLAB). In MATLAB code:

```
Ue=[uW ; U ; uE];
```

2) Introduce dummy cells with values which give the correct boundary conditions on the top and bottom boundaries of the physical domain. The boundary values are stored in vectors `uN` and `uS` as described above.

   In MATLAB:

```
Ue=[2uS'-Ue(:,1) Ue 2uN'-Ue(:,end)];
```

The matrix `Ve` is built in a similar way. Then, the size of the new matrices will be

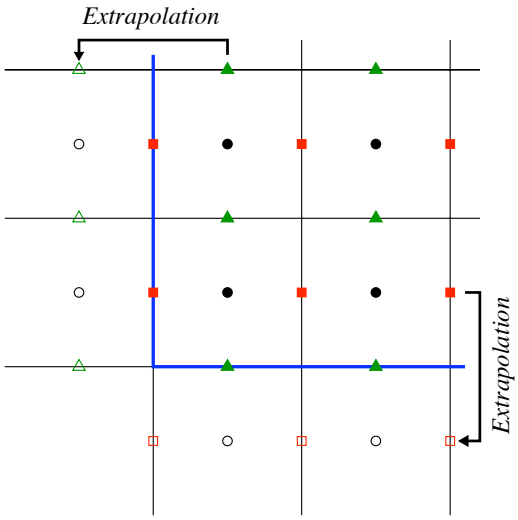

Figure 11: Boundary condition treatment of velocity

`Ue`: $(N_x + 1) \times (N_y + 2)$
`Ve`: $(N_x + 2) \times (N_y + 1)$

The advection (non-linear) terms are calculated using the values of the velocity field on the boundaries of the control volumes which are different from the locations the discrete velocity field is defined. Therefore, we introduce matrices which contain an averaged value of u and v on the boundaries of the control volumes. To generate these data `Ue` and `Ve` should be averaged in the x or y direction depending on the term to be evaluated. Here, we define the *averaging* function `avg( )` such that

$$[\texttt{avg}(f)]_i = \frac{1}{2}\left(f_{i-1} + f_i\right) \ . \tag{35}$$

Note that the length of the resulting vector $\texttt{avg}(f)$ is one element less than the length of f. Furthermore, $\texttt{avg}(f)$ can be applied in both the x- and y-directions. In the example below, the function makes the averaging in either x or y direction.

In MATLAB:

```
function B=avg(A,idim)
if(idim==1)
  B=1/2 [ A(2:end,:) + A(1:end-1,:) ];
elseif(idim==2)
  B=1/2 [ A(:,2:end) + A(:,1:end-1) ];
end;
```

Applying this function to average the u velocity in the y direction can be achieved as follows in MATLAB:

```
Ua=avg(Ue,2);
```

After averaging the size of the matrix `Ua` will be $(N_x + 1) \times (N_y + 1)$.

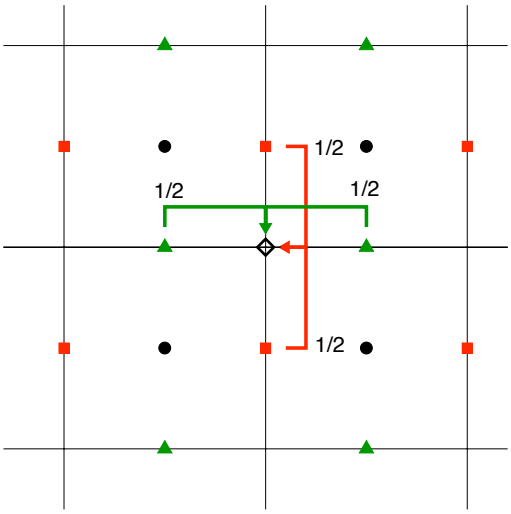

Figure 12: Averaging of velocities.

A.4 Construction of Laplace (L_p) operator

One of the most crucial steps of the code is the implementation of the Laplace operator. We need to construct that in a efficient way. To save memory and gain speed when the equations are solved we make use of *sparse* matrix storage in MATLAB. This can be done in a smart and concise but slightly more complicated way in MATLAB. Let us first define the Laplace operator for a one-dimensional case. Using central finite differences second order, with Neumann boundary condition at the two boundaries we can define a derivative matrix with the following structure,

$$\underline{\underline{D}}_2 = \frac{1}{h^2}\begin{bmatrix} -1 & 1 & & \\ 1 & -2 & 1 & \\ & \ddots & \ddots & \ddots \\ & & 1 & -1 \end{bmatrix} . \tag{36}$$

This is implemented in the function `DD(n,h)`, available as template.

To extended this operator for a two-dimensional case, one can use the `kron` operator in MATLAB to perform a Kronecker tensor product,

$$\texttt{kron}(A,B) = \begin{bmatrix} A(1,1)\cdot B & A(1,2)\cdot B & \cdots \\ A(2,1)\cdot B & A(2,2)\cdot B & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix} . \tag{37}$$

Then one obtains the derivative matrix to compute $U_{xx} = \underline{\underline{L_P}} \cdot U$ by choosing

$$\underline{\underline{A}} = \underline{\underline{I}}_{N_y \times N_y}, \quad \underline{\underline{B}} = \underline{\underline{D}}_2 ,$$

and U_{yy} by choosing

$$\underline{\underline{A}} = \underline{\underline{D}}_2, \quad \underline{\underline{B}} = \underline{\underline{I}}_{N_x \times N_x} .$$

In MATLAB:

```
Lp=kron(speye(ny),DD(nx,dx))+...
```

The actual level of the pressure is not determined since only the pressure gradient enters the Navier–Stokes equations, *i.e.* both $P + const.$ and P satisfy the Poisson equation. Therefore, one needs to fix the value of the pressure at one node, *e.g.* $P_{1,1} = 0$. This is achieved by the following modification of the equation system

$$L_p(1,:) = 0, \quad L_p(1,1) = 1, \quad R(1) = 0 \ .$$

Ignoring this gives singular matrices. The above observation is of course a general feature: For any incompressible flow, the pressure is only determined up to a constant.

A.5 Outline of required steps

Assume that we consider an iteration that starts at $t = t^n$ with U^n, V^n and we will march towards $t = t^{n+1} = t^n + \Delta t$.

1. Compute the non-linear advection terms, namely ${NL_x}^n, {NL_x}^n$:

$$ {NL_x}^n = -((U^n)^2)_x - (U^n V^n)_y \tag{38}$$
$$ {NL_y}^n = -(U^n V^n)_x - ((V^n)^2)_y. \tag{39}$$

   The term ${NL_x}^n$ is evaluated on the U velocity grid, similarly ${NL_y}^n$ on the V grid (see Fig. 10). The derivatives should be calculated based on the difference between the velocity values at neighbouring velocity nodes. Therefore, the velocities U^n and V^n need to be interpolated to the these nodes such that derivatives can be taken. For example for $\left(U^n V^n\right)_x$ we proceed as follows:

$$ ((U^n V^n)_x)_{i,j+\frac{1}{2}} = \frac{1}{h_x}\left[U_{i+\frac{1}{2},j+\frac{1}{2}} V_{i+\frac{1}{2},j+\frac{1}{2}} - U_{i-\frac{1}{2},j+\frac{1}{2}} V_{i-\frac{1}{2},j+\frac{1}{2}}\right] \tag{40}$$
$$ \text{where} \qquad U_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left[U_{i+\frac{1}{2},j+1} + U_{i+\frac{1}{2},j}\right] \tag{41}$$
$$ \text{and} \qquad V_{i+\frac{1}{2},j+\frac{1}{2}} = \frac{1}{2}\left[V_{i+1,j+\frac{1}{2}} + V_{i,j+\frac{1}{2}}\right]. \tag{42}$$

   In MATLAB:

```
Ua=avg(Ue,2);
Va=avg(Ve,1);
UVx = 1/dx * diff( Ua.*Va, 1 , 1);
```

   For the terms $((U^n)^2)_x$ and $((V^n)^2)_y$ the difference is taken between the pressure nodes onto which the velocity has to be interpolated first.

2. Compute the viscous diffusion terms, namely $DIFF_x^n, DIFF_y^n$

$$ DIFF_x^n = \frac{1}{Re}[U_{xx}^n + U_{yy}^n] \tag{43}$$
$$ DIFF_y^n = \frac{1}{Re}[V_{xx}^n + V_{yy}^n] \ . \tag{44}$$

For example

$$[DIFF_x^n]_{i+\frac{1}{2},j} = \frac{1}{Re}\left(\frac{U_{i-\frac{1}{2},j} - 2U_{i+\frac{1}{2},j} + U_{i+\frac{3}{2},j}}{h_x^2} + \frac{U_{i+\frac{1}{2},j-1} - 2U_{i+\frac{1}{2},j} + U_{i+\frac{1}{2},j+1}}{h_y^2}\right) . \tag{45}$$

Use the extended grid and store only interior points.

In MATLAB:

```
viscu = diff( Ue(:,2:end-1),2,1 )/dx^2 + ...
        diff( Ue(2:end-1,:),2,2 )/dy^2;
```

The command `diff( )` is a built-in MATLAB operator which takes the difference between adjacent components. The first argument is the matrix to operate on, the second is the order of the difference (here 2 for the second derivative) and the third is specifying along which matrix dimension the derivative is taken (1 for x-direction and 2 for y-direction).

3. Compute forcing terms if required, F_x^n, F_y^n.

4. Advance the solution to an intermediate time level (U^* and V^*) using explicit Euler

$$\frac{U^* - U^n}{\Delta t} = NL_x^n + DIFF_x^n + F_x^n \ , \tag{46}$$

$$\frac{V^* - V^n}{\Delta t} = NL_y^n + DIFF_y^n + F_y^n \ . \tag{47}$$

5. Solve the Poisson equation for the pressure

$$\Delta P^{n+1} = \frac{1}{\Delta t}(D_x U^* + D_y V^*) \ . \tag{48}$$

Now we need to solve the Poisson equation with homogeneous Neumann conditions for P^{n+1}. Construct the discretised Laplace operator L_p (for details on the structure of the operator L_p see Section A.4 above) and evaluate $D_x U^* + D_y V^*$ with central differences second order.

$$L_p P^{n+1} = R \quad \text{with} \quad R = \frac{1}{\Delta t}(D_x U^* + D_y V^*). \tag{49}$$

$$\Rightarrow \quad P^{n+1} = {L_p}^{-1} R \ . \tag{50}$$

Here, P^{n+1} and R are of size $N_x \times N_y$, containing the unknown pressure and the right-hand side (r.h.s.) at the cell centres, respectively.

In MATLAB:

```
rhs = (diff([uW ; U ; uE])/dx + ...)/dt;
rhs = reshape(rhs,nx*ny,1);
P = Lp\rhs;
P = reshape(P,nx,ny);
```

6. Compute the velocity field at time step $n+1$ according to

$$U^{n+1} = U^* - \Delta t G P^{n+1} \ . \tag{51}$$

B Code

B.1 SG2212_template.m

```
% Navier-Stokes solver,
% adapted for course SG2212
% KTH Mechanics
%
% Depends on avg.m and DD.m
%
% Code version:
% 20150224

clear all

%----------------------------------------

lid_driven_cavity=1;

if (lid_driven_cavity==1)
  % Parameters for test case I: Lid-driven cavity
  % The Richardson number is zero, i.e. passive scalar.

  Pr = 0.71;     % Prandtl number
  Re = ...;     % Reynolds number
  Ri = 0.;       % Richardson number

  dt = ...;    % time step
  Tf = 20;       % final time
  Lx = 1;        % width of box
  Ly = 1;        % height of box
  Nx = ...;       % number of cells in x
  Ny = ...;       % number of cells in y
  ig = 200;      % number of iterations between output

  % Boundary and initial conditions:
  Utop = 1.;
  Ubottom = 0.;
  % IF TEMPERATURE: Tbottom = 1.; Ttop = 0.;
  % IF TEMPERATURE: namp = 0.;
else
  % Parameters for test case II: Rayleigh-Bnard convection
  % The DNS will be stable for Ra=1705, and unstable for Ra=1715
  % (Theoretical limit for pure sinusoidal waves
  % with L=2.01h: Ra=1708)
  % Note the alternative scaling for convection problems.

  Pr = 0.71;     % Prandtl number
  Ra = ...;     % Rayleigh number
```

```
46
47    Re = 1./Pr;     % Reynolds number
48    Ri = Ra*Pr;     % Richardson number
49
50    dt = ...;   % time step
51    Tf = 20;        % final time
52    Lx = 10.;       % width of box
53    Ly = 1;         % height of box
54    Nx = ...;       % number of cells in x
55    Ny = ...;        % number of cells in y
56    ig = 200;       % number of iterations between output
57
58    % Boundary and initial conditions:
59    Utop = 0.;
60    Ubottom = 0.;
61    % IF TEMPERATURE: Tbottom = 1.; Ttop = 0.;
62    namp = 0.1;
63  end
64
65
66  %----------------------------------------
67
68  % Number of iterations
69  Nit = ...
70  % Spatial grid: Location of corners
71  x = linspace( ... );
72  y = linspace( ... );
73  % Grid spacing
74
75  dx = ...
76  dy = ...
77  % Boundary conditions:
78  uN = x*0+Utop;    vN = avg(x,2)*0;
79  uS = ...          vS = ...
80  uW = ...          vW = ...
81  uE = ...          vE = ...
82  tN = ...          tS = ...
83  % Initial conditions
84  U = ...; V = ...;
85  % linear profile for T with random noise
86  % IF TEMPERATURE: T = ... + namp*rand(Nx,Ny)
87  % Time series
88  tser = [];
89  Tser = [];
90
91  %----------------------------------------
92
93  % Compute system matrices for pressure
94  % First set homogeneous Neumann condition all around
```

```
95  % Laplace operator on cell centres: Fxx + Fyy
96  Lp = kron(speye(Ny), ... ) + kron( ... ,speye(Nx));
97  % Set one Dirichlet value to fix pressure in that point
98  Lp(1,:) = ... ; Lp(1,1) = ... ;
99  % For more speed, you could pre-compute the LU decomposition
100 % [LLp,ULp] = lu(Lp);
101 %----------------------------------------
102
103 % Progress bar (do not replace the ... )
104 fprintf(...
105     '[         |         |         |         |         ]\n')
106
107 %----------------------------------------
108
109 % Main loop over iterations
110
111 for k = 1:Nit
112
113    % include all boundary points for u and v (linear extrapolation
114    % for ghost cells) into extended array (Ue,Ve)
115    Ue = ...
116    Ve = ...
117
118    % averaged (Ua,Va) of u and v on corners
119    Ua = ...
120    Va = ...
121
122    % construct individual parts of nonlinear terms
123    dUVdx = ...
124    dUVdy = ...
125    dU2dx = ...
126    dV2dy = ...
127
128    % treat viscosity explicitly
129    viscu = ...
130    viscv = ...
131
132    % buoyancy term
133    % IF TEMPERATURE: fy = ...
134
135    % compose final nonlinear term + explicit viscous terms
136    U = U + dt/Re*... - dt*(...);
137    V = V + dt/Re*... - dt*(...) + % IF TEMPERATURE: dt*fy;
138
139    % pressure correction, Dirichlet P=0 at (1,1)
140    rhs = (diff( ... )/dx + diff( ... )/dy)/dt;
141    rhs = reshape(rhs,Nx*Ny,1);
142    rhs(1) = ...
143    P = Lp\rhs;
```

```
144     % alternatively, you can use the pre-computed LU decompositon
145     % P = ULp\(LLp\rhs);
146     % or as another alternative, reformulate HW6 as function
147     % and call it here:
148     % [PP,msoriter] = GS_SOR_solver(PP,rhs,Lp,Nx,Ny,dx,dy,1.95,1e-4);
149     P = reshape(P,Nx,Ny);
150
151     % apply pressure correction
152     U = U - ...
153     V = V - ...
154
155     % Temperature equation
156     % IF TEMPERATURE: Te = ...
157     % IF TEMPERATURE: Tu = ...
158     % IF TEMPERATURE: Tv = ...
159     % IF TEMPERATURE: H = ...
160     % IF TEMPERATURE: T = T + dt*H;
161
162     %----------------------------------------
163
164     % progress bar
165     if floor(51*k/Nit)>floor(51*(k-1)/Nit), fprintf('.'), end
166
167     % plot solution if needed
168     if k==1|floor(k/ig)==k/ig
169
170       % compute divergence on cell centres
171       if (1==1)
172         div = diff([uW;U;uE])/dx + diff([vS' V vN'],1,2)/dy;
173
174         figure(1);clf; hold on;
175         contourf(avg(x,2),avg(y,2),div');colorbar
176         axis equal; axis([0 Lx 0 Ly]);
177         title(sprintf('divergence at t=%g',k*dt))
178         drawnow
179       end
180
181       % compute velocity on cell corners
182       Ua = ...
183       Va = ...
184       Len = sqrt(Ua.^2+Va.^2+eps);
185
186       figure(2);clf;hold on;
187       %contourf(avg(x,2),avg(y,2),P');colorbar
188       contourf(x,y,sqrt(Ua.^2+Va.^2)',20,'k-');colorbar
189       quiver(x,y,(Ua./Len)',(Va./Len)',.4,'k-')
190       axis equal; axis([0 Lx 0 Ly]);
191       title(sprintf('u at t=%g',k*dt))
192       drawnow
```

```
193
194       % IF TEMPERATURE: % compute temperature on cell corners
195       % IF TEMPERATURE: Ta = ...
196
197       % IF TEMPERATURE: figure(3); clf; hold on;
198       % IF TEMPERATURE: contourf(x,y,Ta',20,'k-');colorbar
199       % IF TEMPERATURE: quiver(x,y,(Ua./Len)',(Va./Len)',.4,'k-')
200       % IF TEMPERATURE: axis equal; axis([0 Lx 0 Ly]);
201       % IF TEMPERATURE: title(sprintf('T at t=%g',k*dt))
202       % IF TEMPERATURE: drawnow
203
204       % Time history
205       if (1==1)
206         figure(4); hold on;
207         tser = [tser k*dt];
208         Tser = [Tser Ue(ceil((Nx+1)/2),ceil((Ny+1)/2))];
209         plot(tser,abs(Tser))
210         title(sprintf('Probe signal at x=%g, y=%g',...
211               x(ceil((Nx+1)/2)),y(ceil((Ny+1)/2))))
212         set(gca,'yscale','log')
213         xlabel('time t');ylabel('u(t)')
214       end
215     end
216 end
217 fprintf('\n')
```

B.2 DD_template.m

```
function A = DD(n,dx)
% DD(n,dx)
%
% One-dimensional finite-difference derivative matrix
% of size n times n for second derivative:
% dx^2 * f''(x_j) = -f(x_j-1) + 2*f(x_j) - f(x_j+1)
%
% Homogeneous Neumann boundary conditions on the boundaries
% are imposed, i.e.
% f(x_0) = f(x_1)
% if the wall lies between x_0 and x_1. This gives then
% dx^2 * f''(x_j) = + f(x_0) - 2*f(x_1) + f(x_2)
%                 = + f(x_1) - 2*f(x_1) + f(x_2)
%                 =              f(x_1) + f(x_2)
%
% For n=5 and h=1 the following result is obtained:
%
% A =
%
%    -1     1     0     0     0
%     1    -2     1     0     0
%     0     1    -2     1     0
%     0     0     1    -2     1
%     0     0     0     1    -1
%
% This function belongs to SG2212.m

A = spdiags( ... )/...;

% "spdiags" generalises the function "diags" such that multiple
% vectors can be provided which are then put on the
% respective diagonals. In addition, sparse storage is used.
% See "help spdiags" for more information.
```

B.3 avg_template.m

```
function B = avg(A,idim)
% AVG(A,idim)
%
% Averaging function to go from cell centres (pressure nodes)
% to cell corners (velocity nodes) and vice versa.
% avg acts on index idim; default is idim=1.
%
% This function belongs to SG2212.m

if nargin<2, idim = 1; end

if (idim==1)
  B = (A( ... , ... )+A( ... , ... ))/2;
elseif (idim==2)
  B = (A( ... , ... )+A( ... , ... ))/2;
else
  error('avg(A,idim): idim must be 1 or 2')
end
```