Technische Universität München
Institut für Informatik
Benjamin Rüth
Benjamin Uekermann
Andreas Reiser
Stefan Gavranovic

Summer Term 2018
Worksheet 2

Praktikum Wissenschaftliches Rechnen Computational Fluid Dynamics

Worksheet 2

Arbitrary Geometries and Energy Transport for the Navier-Stokes Equations

Deadline: May 15, 12:00

In the second part of the lab course, we first extend our Navier-Stakes solver, such that it can work on arbitrary domain geometries and apply various boundary conditions. Afterwards, we will introduce an additional temperature equation to model energy transport. The extended program should read all physical and numerical parameters, together with the boundary conditions, from a single input file `problem.dat`. The program should then be started by the command '`./sim problem`'. Changes of parameters or boundary conditions should no longer require modifications in the source code, nor recompilation of the program.
Additionally to your code and the case files, please submit *screenshots* of all your results. Furthermore, describe your findings and the code usage in a brief `README` text file.

1 Arbitrary Geometries for the Navier-Stokes Equations

1.1 Other Boundary Conditions

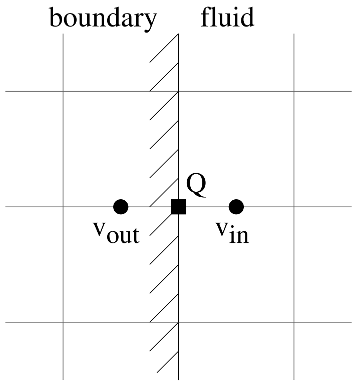
In this section, we consider the boundary conditions once again which have already been discussed in Worksheet 1. We deal with the question how reasonable boundary conditions for the velocity can be implemented. In particular, we introduce three new types of boundary conditions: *free-slip*, *inflow*, and *outflow* conditions.

Free-slip Conditions:

Free-slip conditions model the case when the fluid can flow freely parallel to the domain boundary, but cannot cross the boundary. As a consequence, the velocity component normal to the wall should be zero, as well as the normal derivative of the velocity component parallel to the wall. In our rectangular domain that has been discretized with the help of a staggered grid, the discrete velocity components normal to the wall lie directly at the boundary. Thus, we may set (just as for the no-slip condition):

$$\begin{aligned} u_{0,j} = 0, \quad u_{imax,j} = 0, \qquad j = 1, \ldots, jmax; \\ v_{i,0} = 0, \quad v_{i,jmax} = 0, \qquad i = 1, \ldots, imax \end{aligned} \tag{1.1}$$

(assuming free-slip conditions at all four boundaries).



The normal derivative $\partial v / \partial n$ of the tangential velocity at a boundary point Q may be discretized by the expression $(v_i - v_a)/\delta x$, so that the requirement $\partial v/\partial n = 0$ leads to the condition

$$v_a = v_i.$$

We thus obtain the further boundary conditions

$$\begin{array}{lll} v_{0,j} = v_{1,j}, & v_{imax+1,j} = v_{imax,j}, & j = 1,\dots,jmax; \\ u_{i,0} = u_{i,1}, & u_{i,jmax+1} = u_{i,jmax}, & i = 1,\dots,imax \end{array} \tag{1.2}$$

(again, for all four boundaries).

Outflow Conditions:

For outflow boundary conditions, the normal derivatives of both velocity components are set to zero at the boundary, which means that the total velocity does not change in the direction normal to the boundary. This can be realized in the discrete grid by setting the velocity values at the boundary equal to their neighboring velocities inside the region, i.e.,

$$\begin{array}{lll} u_{0,j} = u_{1,j}, & u_{imax,j} = u_{imax-1,j}, & \multirow{2}{*}{$j = 1,\dots,jmax;$} \\ v_{0,j} = v_{1,j}, & v_{imax+1,j} = v_{imax,j}, & \\ u_{i,0} = u_{i,1}, & u_{i,jmax+1} = u_{i,jmax}, & \multirow{2}{*}{$i = 1,\dots,imax.$} \\ v_{i,0} = v_{i,1}, & v_{i,jmax} = v_{i,jmax-1}, & \end{array} \tag{1.3}$$

Inflow Conditions:

For inflow boundary conditions, the velocities on an inflow boundary are explicitly given. However, since these velocities take diverse values in different examples and are not always constant at the whole boundary, we cannot read these values from the input file in a simple way. Instead, we set these boundary conditions, according to the considered problem, in a function `spec_boundary_val` that is called each time directly after the function `boundary_val`.

Remark:

The boundary conditions for p, F and G have the same form as for no-slip conditions in Worksheet 1 (formulas (16) and (17)), regardless of whether we adopt free-slip, outflow, or inflow conditions.

1.2 Treatment Of General Geometries

Up to now, we have described the simulation of flows in rectangular domains. A simple extension enables us to approximately treat flows in arbitrary two-dimensional geometries. For this, we embed the flow region Ω in a rectangle $\mathcal{R}$ of the smallest possible size, which is then covered by a grid, as described in the previous worksheets. The cells of $\mathcal{R}$ are classified into *fluid cells* (that lie completely or mostly in Ω) and *obstacle cells* (which lie completely or mostly in the obstacle region, $\mathcal{H} := \mathcal{R} \setminus \Omega$). The fluid problem is then to be solved only in the fluid cells. Obstacle cells which share an edge (*boundary edge*) with at least one fluid cell are denoted as *boundary cells.* They play a similar role as

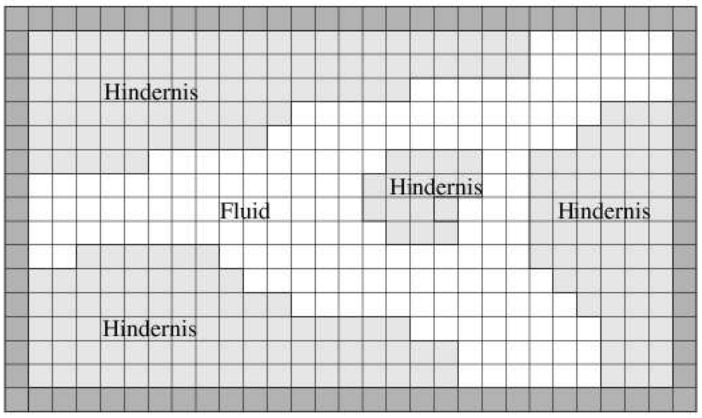


Fig. 1: Embedding of an arbitrary domain into a rectangular domain.

the cells of the artificial cell layers at the outer boundaries. By the way, the cells of the outer boundary layers are also denoted as obstacle cells. A domain Ω with an arbitrary curved boundary is thus approximated by a domain $\tilde{\Omega}$ whose boundary is specified by a set of cell edges (see Figure 1).

Obstacle Boundary Conditions:

In fluid cells adjacent to obstacle cells, we need the following boundary values in order to compute F and G according to (9) and (10) from Worksheet 1:

- values of the normal velocity components at the boundary edges and
- values of the tangential velocity components on edges between two boundary cells.

We only implement no-slip conditions at the internal boundary edges. In the example shown in Figure 2, the boundary edges are marked with a square, whereas the edges between two boundary cells are marked with a circle. Furthermore, to compute the pressure at the centers of these fluid cells, the F and G values at the boundary edges are needed, as well as the pressure values at the centers of the adjacent boundary cells (see the cross in Figure 2).
To classify the grid cells, we will use an integer array `int **Flag` where each entry is a bitfield with the following layout:

0	1	2	3	4	5	6	7	8
`FLUID`	`NO-SLIP`	`FREE-SLIP`	`OUTFLOW`	`INFLOW`	`B_N`	`B_S`	`B_W`	`B_O`

To implement the initialization of boundary values, the set of boundary cells must be further classified: in the cell flag array we store which of the four neighboring cells are fluid cells. Thus, the macro value `B_O` designates e.g. a boundary cell whose right (eastern, “Ost”) neighbor belongs to $\tilde{\Omega}$. Similar, `B_SW` may denote a boundary cell

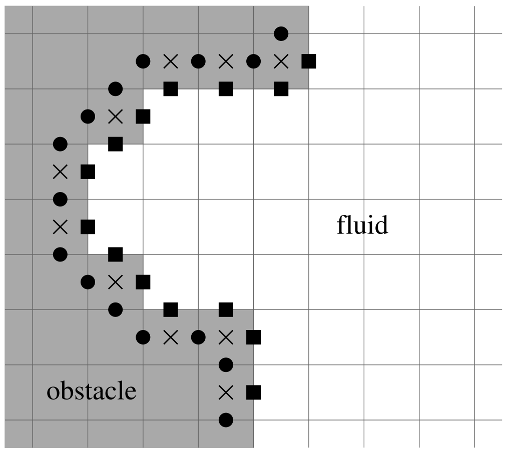


Fig.2: Required boundary values

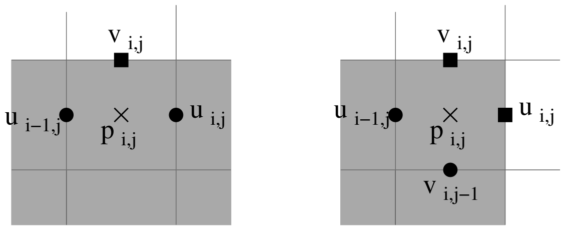


Fig. 3: Setting boundary values at obstacle cells (left: edge cell `B_N`, right: corner cell `B_NO`

whose lower (southern) and left (western) neighbors belong to $\tilde{\Omega}$. We assume that only boundary cells with one neighboring fluid cell [1] (*edge cells*) or two neighboring fluid cells sharing a corner (*corner cells*) may occur. Boundary cells with two opposite or even three or four neighboring fluid cells are excluded (forbidden boundary cells), because these no longer allow uniquely defined boundary values. The accuracy of representing the geometry is thus limited by the double cell length.
In accordance to the formulas (14), (15), (16) and (17) of Worksheet 1, we set for a northern edge cell (i,j) having the flag `B_N`

$$\begin{array}{lll} v_{i,j} = 0, & u_{i-1,j} = -u_{i-1,j+1}, & u_{i,j} = -u_{i,j+1}, \\ G_{i,j} = v_{i,j}, & p_{i,j} = p_{i,j+1} & \end{array} \tag{1.4}$$

or, for a western edge cell (i,j) with flag `B_W`

$$\begin{array}{lll} u_{i-1,j} = 0, & v_{i,j-1} = -v_{i-1,j-1}, & v_{i,j} = -v_{i-1,j}, \\ F_{i-1,j} = u_{i-1,j} & p_{i,j} = p_{i-1,j}. & \end{array} \tag{1.5}$$

The boundary values for `B_S` and `B_O` cells are set analogously.

[1]We consider cells to be *neighboring* if they share a common edge, not only a corner.

For corner cells, we apply the condition to the normal velocity component at the two edges, and the condition for the tangential velocity component at the remaining two edges. Thus, e.g. for a cell (i, j) with flag `B_NO`

$$\begin{aligned} u_{i,j} &= 0, & u_{i-1,j} &= -u_{i-1,j+1}, & F_{i,j} &= u_{i,j}, \\ v_{i,j} &= 0, & v_{i,j-1} &= -v_{i+1,j-1}, & G_{i,j} &= v_{i,j}, \\ & & p_{i,j} &= (p_{i,j+1} + p_{i+1,j})/2. & & \end{aligned} \tag{1.6}$$

The solution of the pressure equation (Worksheet 1, formula (11)) is constrained to the fluid cells, whereas the computation of the F and G values according to (9) and (10), Worksheet 1, as well as the correction to the velocity value according to (7), (8), Worksheet 1, takes place only on edges between two fluid cells.

As a summary, we shall list once more all cell and edge definitions:

• fluid cell:	cell lying completely or mostly in Ω
• obstacle cell:	cell lying completely or mostly in $\mathcal{R} \setminus \Omega$
• boundary cell:	obstacle cell bordering on one or more fluid cells
• edge cell:	boundary cell bordering on exactly one fluid cell
• corner cell:	boundary cell bordering on two fluid cells sharing a corner
• forbidden boundary cell:	boundary cell that is neither corner, nor edge cell
• boundary edge:	edge between a fluid cell and a boundary cell

1.3 Implementation

1. Extend the function `read_parameters` such that it also reads the needed `char*` variables `problem` and `geometry`.
   - `problem` is a short string which indicates the scenario to be calculated, i. e. driven-cavity, . . .
   - `geometry` is the file name/path to the geometry.pgm file.
2. Use a new function `void init_flag(problem, geometry, imax, jmax, Flag)` to set up the integer array `int **Flag` with the right dimensions and values. Use the function `read_pgm(filename)` in `helper.c` to read in a geometry.pgm file. It's your choice how to use and do this exactly. Think about different options:
   - Use the geometry.pgm file for all scenarios with a 1:1 equivalence (each entry in the file corresponds to exactly one cell).
   - Only use it as template and scale it to the appropriate size.
   - Don't use it for scenarios without obstacles (driven cavity, . . .).

   Use one loop to first translate the values of your geometry file to fluid/obstacle flags. Following this, do another pass to set the `B_X` flags. Using bitwise operations, this can be done quite elegantly. The flags `B_X` are set to 1 when the corresponding neighbouring cell is a fluid cell. Thus the flag 010001000 belongs to an edge cell with a no-slip boundary condition and one fluid neighbour above. Of course only one of the first five fields can be set at a given time. You may use `assert()` to rule out any forbidden configurations early. When setting the initial values for the velocities and pressure (and later temperature), make sure that you only set them for fluid cells.

FLUID	NO-SLIP	FREE-SLIP	OUTFLOW	INFLOW	B_N	B_S	B_W	B_O
0	1	0	0	0	1	0	0	0

3. In `boundary_values`, `spec_boundary_val` and `calculate_fg`, boundary values must be assigned to the boundary cells defined by the flag array, as described above.

4. In `SOR`, apart from the boundary values for the pressure at the boundary stripe, also the boundary values in the interior boundary cells should be set before each iteration step using Neumann conditions.

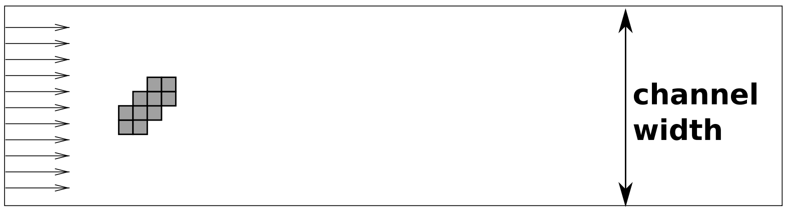
5. In `SOR`, the iteration and the residual computation needs to be limited to the fluid cells, whereas F and G values in `calculate_fg` and velocities in `calculate_uv` are only calculated on edges separating two fluid cells. Besides, the normalization of the residual in `SOR` should be produced by dividing by the current number of fluid cells, and not by $imax \cdot jmax$.

6. Extend the output to also include the geometry. For simplification, you may leave out the domain boundaries.

1.4 Example Problems

With the help of these modifications of the flow program, we are now able to compute a whole bunch of new examples. We only need to set up a new input file for each example. Besides, the problem-specific obstacle cells should be implemented in the function `init_flag` as well as the inflow conditions and further special boundary conditions in the function `spec_boundary_val`.

a) **The Karman Vortex Street**:

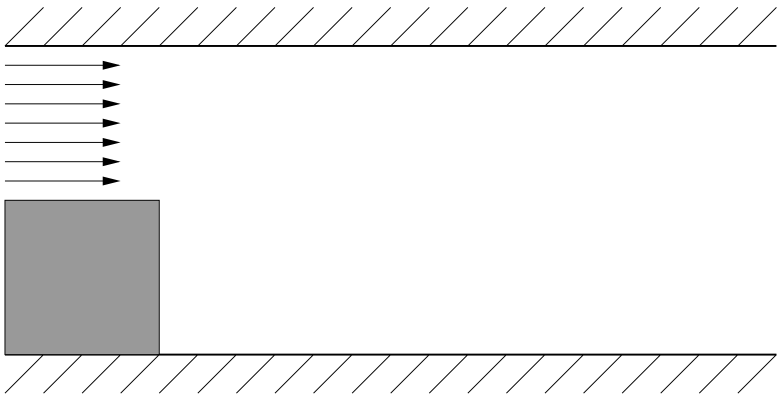


The flow in a channel hits a tilted plate. At the left boundary, the fluid inflow has a constant velocity profile ($u = 1.0$, $v = 0.0$), while at the upper and lower boundaries no-slip conditions are imposed. The plate occupies one fifth of the channel width and is three cells thick. The distance from the left boundary is assumed to be the same as the distance to the lower and upper boundaries. Make sure that there are no forbidden obstacle cells. Use the following values for the parameters:

imax = 100	jmax = 20	xlength = 10	ylength = 2
dt = 0.05	t_end = 20	tau = 0.5	dt_value = 2.0
eps = 0.001	omg = 1.7	alpha = 0.9	itermax = 500
GX = 0.0	GY = 0.0	Re = 10000	
UI = 1.0	VI = 0.0	PI = 0.0	

Use the tools provided by ParaView to characterize the flow around the obstacle.

b) **Flow over a Step**:



The fluid flows through a channel widening on one side. No-slip conditions are imposed at the upper and lower walls. The obstacle domain is represented by a square filling up half of the channel height. Use the following values for the problem parameters:

imax = 100	jmax = 20	xlength = 10	ylength = 2
dt = 0.05	t_end = 500	tau = 0.5	dt_value = 10.0
eps = 0.001	omg = 1.7	alpha = 0.9	itermax = 500
GX = 0.0	GY = 0.0	Re = 100	
UI = 0.0	VI = 0.0	PI = 0.0	

2 Energy Transport

For various practical applications, determining the effects of temperature variations on the flow or the transfer of heat within the flow, is of high relevance. In this part of the CFD-Lab course, we will extend our existing incompressible Navier-Stokes solver by an energy transport equation following the Boussinesq approximation. In Worksheet 4, we will then couple this energy transport to an external structural mechanics solver.

2.1 Boussinesq Approximation

The principle effect of temperature on a fluid results from the fact that changes in temperature cause variations in the fluid's density: When heated, a fluid's volume increases, thus making it lighter and causing it to rise. This results in the occurrence of buoyancy forces which depend on the temperature (thermal buoyancy forces). Incorporating yet further effects are difficult to treat. Hence simplifications are necessary. The most common approach is to use the Boussinesq approximation:

- Density is constant except in the buoyancy terms (continuity equation retains its incompressibility),
- all other fluid properties are assumed constant (e.g. viscosity),
- viscous dissipation is negligibly small, and
- the relation between density ρ and temperature T is assumed to be linear.

To express the linear dependence of the density on the temperature, we define the coefficient of thermal expansion $\beta = -\rho^{-1}\frac{\partial \rho}{\partial T}$. These assumptions, of course, limit the applicability of the Boussinesq approximation. For example, it is only valid for small temperature variations. For more detailed explanations, please refer to *Griebel et al., Numerical Simulation in Fluid Dynamics: A Practical Introduction, Section 9.3.*
The dimensionless *energy equation* in 2D reads:

$$\frac{\partial T}{\partial t} + \frac{\partial (uT)}{\partial x} + \frac{\partial (vT)}{\partial y} = \frac{1}{Re \cdot Pr}\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) . \tag{2.1}$$

The Prandtl number Pr describes the ratio of diffusion in the momentum equations and the diffusion in the energy equation: $Pr := \frac{\nu}{\alpha}$ with kinematic viscosity ν and the thermal diffusivity α. The Prandtl number is entirely a property of the fluid and not of the flow as the Reynolds number. The energy equation is a classical convection-diffusion equation: Temperature is convected with the flow $\partial(uT)/\partial x$, $\partial(vT)/\partial y$ but also diffuses $\partial^2 T/\partial x^2 + \partial^2 T/\partial y^2$.

To account for the thermal buoyancy forces, the momentum equations have to be extended as follows:

$$\frac{\partial u}{\partial t} + \frac{\partial (u^2)}{\partial x} + \frac{\partial (uv)}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + (1-\beta\, T) g_x \,, \tag{2.2}$$

$$\frac{\partial v}{\partial t} + \frac{\partial (uv)}{\partial x} + \frac{\partial (v^2)}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + (1-\beta\, T) g_y \,. \tag{2.3}$$

2.2 Discretization of the Energy Equation

To discretize the new energy equation, we locate the temperature values T as well as the discrete stencils at the cell centers, similar to the pressure values. We use a pure forward Euler for the time integration and similar stencils as for the momentum equations. We get

$$\left[\frac{\partial T}{\partial t}\right]_{i,j}^{(n+1)} + \left[\frac{\partial (uT)}{\partial x}\right]_{i,j} + \left[\frac{\partial (vT)}{\partial y}\right]_{i,j} = \frac{1}{Re \cdot Pr}\left(\left[\frac{\partial^2 T}{\partial x^2}\right]_{i,j} + \left[\frac{\partial^2 T}{\partial y^2}\right]_{i,j}\right) \,, \tag{2.4}$$

with the stencils

$$\left[\frac{\partial T}{\partial t}\right]_{i,j}^{n+1} = \frac{1}{\delta t}\left(T_{i,j}^{(n+1)} - T_{i,j}^{(n)}\right) \,,$$

$$\begin{aligned}\left[\frac{\partial (uT)}{\partial x}\right]_{i,j} &= \frac{1}{\delta x}\left(u_{i,j}\frac{T_{i,j}+T_{i+1,j}}{2} - u_{i-1,j}\frac{T_{i-1,j}+T_{i,j}}{2}\right) \\ &+ \frac{\gamma}{\delta x}\left(|u_{i,j}|\frac{T_{i,j}-T_{i+1,j}}{2} - |u_{i-1,j}|\frac{T_{i-1,j}-T_{i,j}}{2}\right) \,,\end{aligned}$$

$$\begin{aligned}\left[\frac{\partial (vT)}{\partial y}\right]_{i,j} &= \frac{1}{\delta y}\left(v_{i,j}\frac{T_{i,j}+T_{i,j+1}}{2} - v_{i,j-1}\frac{T_{i,j-1}+T_{i,j}}{2}\right) \\ &+ \frac{\gamma}{\delta y}\left(|v_{i,j}|\frac{T_{i,j}-T_{i,j+1}}{2} - |v_{i,j-1}|\frac{T_{i,j-1}-T_{i,j}}{2}\right) \,,\end{aligned}$$

$$\left[\frac{\partial^2 T}{\partial x^2}\right]_{i,j} = \frac{T_{i+1,j} - 2\,T_{i,j} + T_{i-1,j}}{(\delta x)^2} \,,$$

$$\left[\frac{\partial^2 T}{\partial y^2}\right]_{i,j} = \frac{T_{i,j+1} - 2\,T_{i,j} + T_{i,j-1}}{(\delta y)^2} \,.$$

$\gamma \in [0,1]$ denotes the donor-cell coefficient. It can be set to the same value as for the momentum equations. We use γ here to avoid any confusion with the thermal diffusivity.

Due to the purely explicit time integration, we get a further stability constraint:

$$\delta t < \frac{Re \cdot Pr}{2} \left(\frac{1}{(\delta x)^2} + \frac{1}{(\delta y)^2} \right)^{-1} , \tag{2.5}$$

which has to be included in the adaptive timestep size computation.

2.3 Boundary and Initial Conditions

As usual, we distinguish between Dirichlet and Neumann boundary conditions, $\Gamma = \Gamma_D \cup \Gamma_N$.

Dirichlet Boundary Conditions

At a Dirichlet boundary condition, the temperature is prescribed

$$T\big|_{\Gamma_D} = T_D ,$$

modelling the heating or cooling at a wall or the fixed temperature at an inflow. To satisfy a discrete Dirichlet condition directly at the wall, the two neighboring cell centers are averaged. We get

$$\begin{aligned}
T_{0,j} &= 2T_D - T_{1,j} , && \text{for } j = 1, \ldots, j_{max} \text{ at the left boundary,} && (2.6)\\
T_{i_{max}+1,j} &= 2T_D - T_{i_{max},j} , && \text{for } j = 1, \ldots, j_{max} \text{ at the right boundary,} && (2.7)\\
T_{i,0} &= 2T_D - T_{i,1} , && \text{for } i = 1, \ldots, i_{max} \text{ at the bottom boundary,} && (2.8)\\
T_{i,j_{max}+1} &= 2T_D - T_{i,j_{max}} , && \text{for } i = 1, \ldots, i_{max} \text{ at the top boundary.} && (2.9)
\end{aligned}$$

Neumann Boundary Conditions

With a Neumann boundary condition, we describe a given heat flux across a boundary

$$-\frac{\partial T}{\partial n}\bigg|_{\Gamma_N} = \frac{q_N}{\kappa} ,$$

where n is the normal vector at the boundary pointing outward, κ is the fluid's thermal conductivity, and q_N is the given heat flux. The boundary condition describes how much heat is passed over the boundary, which depends on the material properties as well as the temperature difference. When q_N is zero, the boundary condition is called adiabatic: no heat exchange across the boundary, modeling an insulated wall or a standard outflow condition. To satisfy a discrete Neumann condition directly at the wall, we use

$$\begin{aligned}
T_{0,j} &= T_{1,j} + \delta x \cdot \kappa^{-1} q_N , && \text{for } j = 1, \ldots, j_{max} \text{ at the left boundary,} && (2.10)\\
T_{i_{max}+1,j} &= T_{i_{max},j} + \delta x \cdot \kappa^{-1} q_N , && \text{for } j = 1, \ldots, j_{max} \text{ at the right boundary,} && (2.11)\\
T_{i,0} &= T_{i,1} + \delta y \cdot \kappa^{-1} q_N , && \text{for } i = 1, \ldots, i_{max} \text{ at the bottom boundary,} && (2.12)\\
T_{i,j_{max}+1} &= T_{i,j_{max}} + \delta y \cdot \kappa^{-1} q_N , && \text{for } i = 1, \ldots, i_{max} \text{ at the top boundary.} && (2.13)
\end{aligned}$$

Initial Conditions

As initial condition, we simply give an explicit value of T to set everywhere.

2.4 Adaption of Momentum Equations and the Algorithm

Finally, we also have to adapt the discrete momentum equations. F and G have to be updated as follows:

$$\tilde{F}_{i,j}^{(n)} := F_{i,j}^{(n)} - \beta \frac{\delta t}{2} \left(T_{i,j}^{(n+1)} + T_{i+1,j}^{(n+1)} \right) g_x \ , \tag{2.14}$$

$$\tilde{G}_{i,j}^{(n)} := G_{i,j}^{(n)} - \beta \frac{\delta t}{2} \left(T_{i,j}^{(n+1)} + T_{i,j+1}^{(n+1)} \right) g_x \ . \tag{2.15}$$

We get the following updated algorithm:

```
Read the problem parameters
Set $t := 0$, $n := 0$
Assign initial values to $u, v, p, T$
While $t < t_{\text{end}}$
    Select $\delta t$ (further include new $T$ criterion)
    Set boundary values for $u$, $v$, and T (according to (2.6) to (2.13))
    Compute $T^{(n+1)}$ according to (2.4)
    Compute new $\tilde{F}^{(n)}$ and $\tilde{G}^{(n)}$ according to (2.14) and (2.15)
    Compute rhs of the pressure equation (with new $\tilde{F}$ and $\tilde{G}$)
    Set $it := 0$
    While $it < itmax$ and $res > \varepsilon$
        Perform a SOR iteration and retrieve the residual
        $it := it + 1$
    Compute $u^{(n+1)}$ and $v^{(n+1)}$ (with new $\tilde{F}$ and $\tilde{G}$)
    Output of $u, v, p, T$ values for visualization, if necessary
    $t := t + \delta t$
    $n := n + 1$
```

2.5 Implementation

1. As before, we need new parameters which need to be read from the configuration file. Extend the read functionality to also read in the initial Temperature `TI`, the Prandtl number `Pr` and the coefficient of thermal expansion β.
2. Adapt `calculate_dt` with the new maximum timestep condition.
3. Initialize the needed array `double** Temp` for the temperature values. Add and implement a new method `calculate_temp` in `uvp.c`. Keep in mind that you can't immediately update a value in place with a newly calculated one.

4. Make the necessary adjustment in the calculation of the F and G terms.

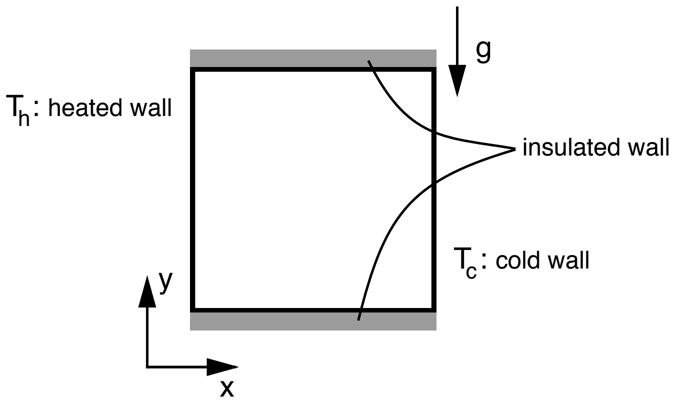
5. Set the right temperature boundary values. Obstacles are always insulated, i. e. just copy the temperature value.

6. Add the temperature to the output.

2.6 Example Problems

c) **Natural Convection**:

In the previous example, we always had forced convection, we prescribed a fixed velocity at some boundary. Now, with the added energy transport, we can also simulate flows solely driven by gravity. Use the setup as shown in the picture, all boundaries have no-slip conditions. Use both of the provided parameter values[1].



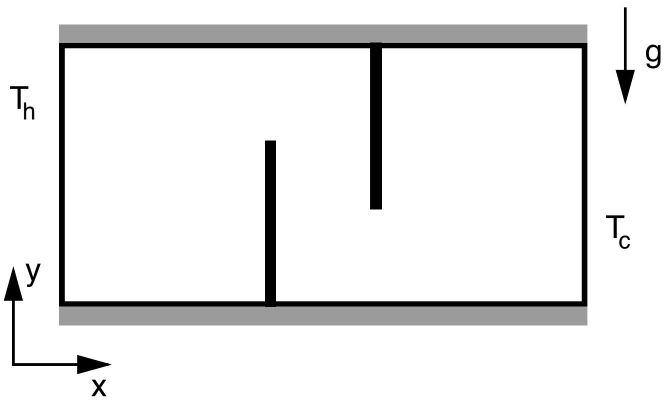
imax = 50	jmax = 50	xlength = 1	ylength = 1
dt = 0.05	t_end = 1000	tau = 0.5	dt_value = 10
eps = 0.00001	omg = 1.7	alpha = 0.5	itermax = 100
GX = 0.0	GY = -1.1	Re = 1000	PR = 7
UI = 0.0	VI = 0.0	PI = 0.0	
TI = 0.0	T_h = 1.0	T_c = 0.0	beta = 0.00021

imax = 50	jmax = 50	xlength = 1	ylength = 1
dt = 0.05	t_end = 10000	tau = 0.5	dt_value = 50
eps = 0.00001	omg = 1.7	alpha = 0.5	itermax = 100
GX = 0.0	GY = -1.1	Re = 20000	PR = 7
UI = 0.0	VI = 0.0	PI = 0.0	
TI = 0.0	T_h = 1.0	T_c = 0.0	beta = 0.00021

[1]For all example problems, alpha refers to the donor-cell coefficient and not to the thermal diffusivity.

d) **Fluid Trap**:

To approach more realistic configurations such as those found in lakes, in cooling systems of buildings, or in solar panels, we must also consider internal obstacle structures. Consider the domain shown in the picture. All boundaries have no-slip conditions. Use the given parameters for your simulation. What happens when you switch the sides of the heated and cold wall?



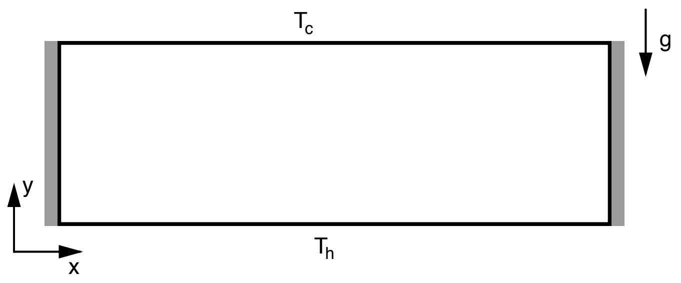
imax = 100	jmax = 50	xlength = 2	ylength = 1
dt = 0.05	t_end = 2000	tau = 0.5	dt_value = 10.0
eps = 0.00001	omg = 1.7	alpha = 0.5	itermax = 1000
GX = 0.0	GY = -9.81	Re = 10000	PR = 7
UI = 0.0	VI = 0.0	PI = 0.0	
TI = 0.0	T_h = 0.5	T_c = -0.5	beta = 0.00063

e) **Rayleigh-Bénard Convection**:

Excerpt from Wikipedia[2]:
Rayleigh-Bénard convection is a type of natural convection, occurring in a plane horizontal layer of fluid heated from below, in which the fluid develops a regular pattern of convection cells known as Bénard cells. Rayleigh-Bénard convection is one of the most commonly studied convection phenomena because of its analytical and experimental accessibility.

Please read through the complete article. Use the provided parameter values as a baseline, but experiment with different geometric settings and resolutions. All boundaries have - again - no-slip conditions.

[2] https://en.wikipedia.org/wiki/Rayleigh-Bénard_convection



imax = 85	jmax = 18	xlength = 8.5	ylength = 1
dt = 0.05	t_end = 45000	tau = 0.5	dt_value = 100.0
eps = 0.00001	omg = 1.7	alpha = 0.5	itermax = 100
GX = 0.0	GY = -0.3924	Re = 33.73	PR = 12500
UI = 0.0	VI = 0.0	PI = 0.0	
TI = 293	T_h = 294.78	T_c = 291.20	beta = 0.00179