

**Riccardo Zamolo, Enrico Nobile**  
*DIA - Dipartimento di Ingegneria e Architettura*  
*Università degli Studi di Trieste*

*Esercitazioni di Termofluidodinamica Computazionale*

**2D differentially heated cavity with  
ANSYS Fluent and MATLAB**



April 2023



# 1 Introduction

## 1.1 Problem definition

This tutorial document shows how to perform a CFD analysis of a 2D differentially heated cavity using ANSYS Workbench 2022 (hereafter “WB”) for the simulations and MATLAB for some postprocessing calculations.

The differentially heated cavity is a typical CFD benchmark problem which involves a buoyancy-driven flow due to density variations, which are in turn due to temperature variations, i.e., natural convection:

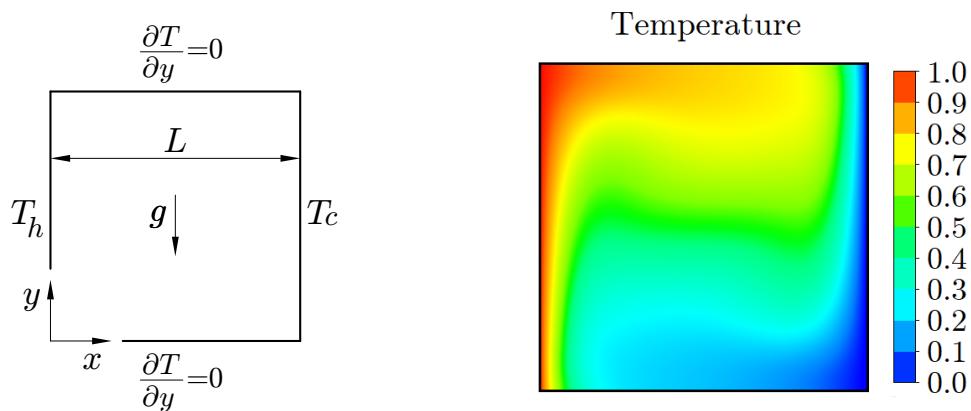
“Buoyancy-driven flow in a square cavity with vertical sides which are differentially heated is a suitable vehicle for testing and validating computer codes used for a wide variety of practical problems.” [1]

With reference to Figure 1, a fluid is enclosed in a 2D square cavity with side length  $L$  where the left and right walls are kept at fixed temperatures  $T_h$  and  $T_c$ , respectively, while the horizontal walls are adiabatic, i.e., no heat transfer. The fluid is subjected to the gravity  $g$  along the vertical direction  $y$ ; in other words the 2D cavity can be imagined in a vertical plane: this is necessary for the appearance of a buoyancy-driven flow due to density variations.

Intuitively, for a given fluid, the intensity of the buoyancy forces increases as both the temperature difference  $\Delta T = T_h - T_c$  and the dimension  $L$  increase. Therefore, weak convective motions are expected for small cavities with small  $\Delta T$ , while stronger convective flows are expected for big cavities with large  $\Delta T$ . Accordingly, it can also be expected that sufficiently strong buoyancy forces will lead to unsteady flows, while steady flows are expected for weak buoyancy forces. In this document we will focus on the latter case, i.e., a laminar, steady-state natural convection problem.

## 1.2 Governing equations

The aforementioned steady-state problem is described by the following coupled conservation equations of mass, momentum and energy with Boussinesq approximation, i.e., linear depen-



**Figure 1:** 2D differentially heated cavity: schematic representation of the problem (left) and example of temperature field for  $Ra = 10^5$  and  $Pr = 0.71$  (right).

dence of density on temperature in the buoyancy term alone:

$$\nabla \cdot \mathbf{u} = 0 \quad (1)$$

$$(\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho_0}\nabla p + \nu\nabla^2\mathbf{u} + g\beta(T - T_0)\mathbf{e}_y \quad (2)$$

$$\mathbf{u} \cdot \nabla T = \alpha\nabla^2 T \quad (3)$$

where  $\mathbf{e}_y$  is the unit vector along  $y$ ,  $\rho_0$  is the reference density, respectively,  $\nu$  is the kinematic viscosity,  $g$  is the gravitational acceleration,  $\beta$  is the thermal expansion coefficient and  $\alpha$  is the thermal diffusivity.  $T_0$  is the reference temperature, which can be assumed to be the mean temperature  $(T_h + T_c)/2$ . Other choices for the reference values of temperature and density in equation (2) are equivalent from a mathematical point of view since they only affect the pressure by a multiplicative constant  $\rho_0$  and by a static head  $g\beta T_0\rho_0$ .

The physical parameters of the problem are 6:  $L$  from the geometry,  $\Delta T$  from the boundary conditions,  $\rho_0$ ,  $\nu$ ,  $g\beta$  and  $\alpha$  from equations (1)-(3). The involved fundamental units are 4: length, mass, time and temperature. The Buckingham theorem (BT) states that the problem is then completely described by  $6 - 4 = 2$  adimensional groups (the dimensional matrix has rank 4). For example, the following groups can be obtained (they are not unique):

$$\pi_1 = \frac{\nu}{\alpha} =: \text{Pr} \quad (\text{Prandtl number}) \quad (4)$$

$$\pi_2 = \frac{g\beta\Delta TL^3}{\nu\alpha} =: \text{Ra} \quad (\text{Rayleigh number}) \quad (5)$$

Pr is the ratio between the momentum and thermal diffusion coefficients, i.e., kinematic viscosity  $\nu$  and thermal diffusivity  $\alpha$ , and therefore it is strictly related to the relative thickness of the momentum and thermal boundary layers. It depends only on the fluid:  $\text{Pr} \ll 1$  for liquid metals,  $\text{Pr} \approx 1$  for gases,  $\text{Pr} \gg 1$  for oils.

Ra is the ratio between the characteristic time scales of diffusive and convective thermal transport due to conduction and buoyancy, respectively. Therefore, the larger the Ra, the faster the convective dynamics compared to the conductive one. For a given Pr, the transition from steady to unsteady flows is controlled by Ra only. In this document the chosen fluid is air with  $\text{Pr} = 0.71$ , for which the onset of unsteadiness occurs for  $\text{Ra} \approx 1.82 \times 10^8$  [3].

Since the ratio between two adimensional groups is adimensional as well, another choice for the adimensionalization could be obtained by considering  $\text{Ra}/\text{Pr}$  instead of Ra:

$$\frac{\text{Ra}}{\text{Pr}} = \frac{g\beta\Delta TL^3}{\nu^2} =: \text{Gr} \quad (\text{Grashof number}) \quad (6)$$

which is the ratio between buoyancy and viscous forces: the larger the Gr, the stronger the convection due to buoyancy.

Without manipulating equations (1)-(3), the dimensional analysis tells also us how to scale the dependent variables of the problem, i.e., nondimensionalization: trivially, the position  $\mathbf{x}$  and the temperature difference  $T - T_0$  can be scaled by  $L$  and  $\Delta T$ , respectively. The dimensional analysis then states that the velocity  $\mathbf{u}$  and the pressure  $p$  can be scaled by  $u_0 = \alpha/L$  and  $\rho_0 u_0^2$ , respectively (again, these choices are not unique).

The same values can be obtained by a direct nondimensionalization of equations (1)-(3). By considering the following scalings:

$$\hat{\mathbf{x}} := \frac{\mathbf{x}}{L}, \quad \hat{\mathbf{u}} := \frac{\mathbf{u}}{u_0}, \quad \hat{T} := \frac{T - T_0}{\Delta T}, \quad (7)$$

equation (3) takes the following form:

$$\hat{\mathbf{u}} \cdot \hat{\nabla} \hat{T} = \frac{\alpha}{u_0 L} \hat{\nabla}^2 \hat{T} \quad (8)$$

where the coefficient  $\alpha/(u_0 L)$  must be nondimensional, since every other term is nondimensional as well. We can therefore choose  $\alpha/(u_0 L) = 1$  for the sake of convenience, i.e.,  $u_0 = \alpha/L$ , as previously obtained.

Equation (2), on the other hand, becomes:

$$(\hat{\mathbf{u}} \cdot \hat{\nabla}) \hat{\mathbf{u}} = -\frac{1}{\rho_0 u_0^2} \hat{\nabla} p + \frac{\nu}{\alpha} \hat{\nabla}^2 \hat{\mathbf{u}} + \frac{g\beta\Delta T L^3}{\alpha^2} \hat{T} \mathbf{e}_y \quad (9)$$

and again, since every term must be nondimensional, the following nondimensional terms arise, as previously obtained:

$$\frac{p}{\rho_0 u_0^2} =: \hat{p}, \quad \frac{\nu}{\alpha} = \text{Pr}, \quad \frac{g\beta\Delta T L^3}{\alpha^2} = \frac{g\beta\Delta T L^3 \nu}{\nu \alpha} \frac{\nu}{\alpha} = \text{Ra Pr} \quad (10)$$

In the end, the nondimensional form of equations (1)-(3) is therefore:

$$\hat{\nabla} \cdot \hat{\mathbf{u}} = 0 \quad (11)$$

$$(\hat{\mathbf{u}} \cdot \hat{\nabla}) \hat{\mathbf{u}} = -\hat{\nabla} \hat{p} + \text{Pr} \hat{\nabla}^2 \hat{\mathbf{u}} + \text{Ra Pr} \hat{T} \mathbf{e}_y \quad (12)$$

$$\hat{\mathbf{u}} \cdot \hat{\nabla} \hat{T} = \hat{\nabla}^2 \hat{T} \quad (13)$$

which depends only on the two adimensional numbers Ra and Pr as previously obtained from the BT, i.e.,  $\hat{\mathbf{u}}, \hat{p}, \hat{T} = f(\text{Ra}, \text{Pr})$ .

By considering the heat transfer coefficient  $h$  as another physical (dependent) variable defined by the wall heat flux  $q = h\Delta T = -k\partial T/\partial x$ , where  $k$  is the thermal conductivity of the fluid, we can also obtain the following adimensional group:

$$\pi_3 = \frac{hL}{k} =: \text{Nu}^{\text{BT}} = f(\text{Ra}, \text{Pr}) \quad (\text{Nusselt number}) \quad (14)$$

Nu is therefore the ratio between the heat flux at the boundary and the reference one due to  $\Delta T$ , i.e.,  $k\Delta T/L$ . Equation (14) can be interpreted both locally, i.e., at a given height  $y$  on the isothermal walls, and globally, i.e., by considering the integral mean over each isothermal wall:

$$\text{Nu}_y = \frac{-\frac{\partial T}{\partial x} \Big|_{x=0,L}}{\frac{\Delta T}{L}} = -\frac{\partial \hat{T}}{\partial \hat{x}} \Big|_{\hat{x}=0,1} \quad (15)$$

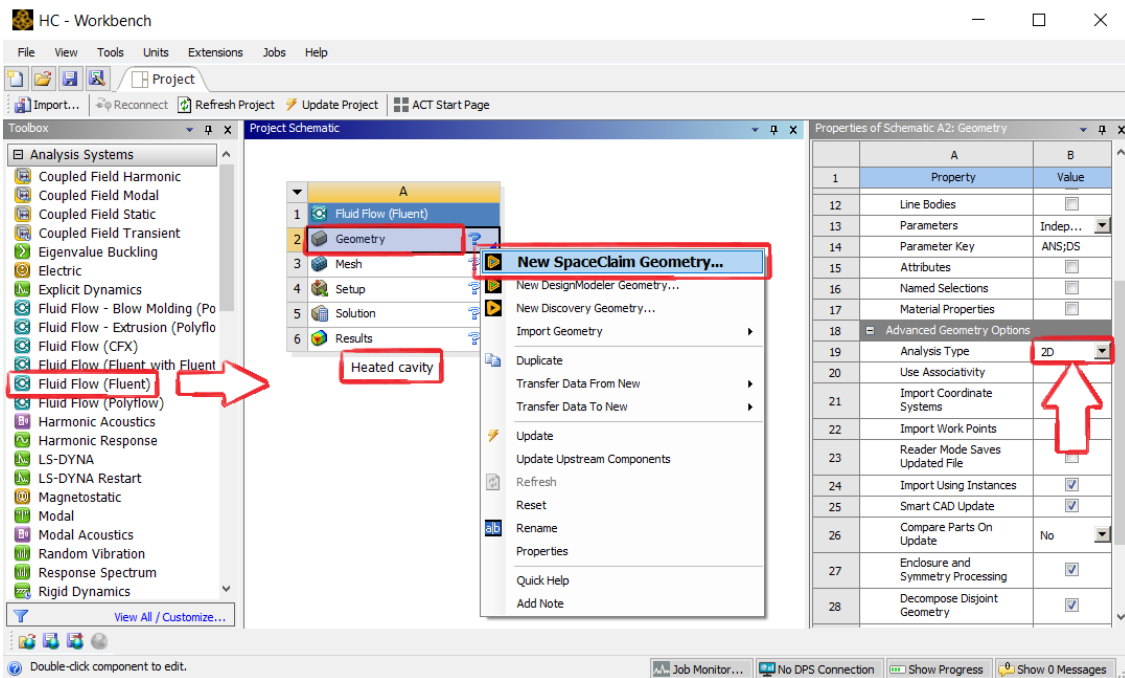
$$\overline{\text{Nu}} = \frac{1}{L} \int_0^L \text{Nu}_y dy = -\int_0^1 \frac{\partial \hat{T}}{\partial \hat{x}} \Big|_{\hat{x}=0,1} d\hat{y} \quad (16)$$

which are thus, by their own definition, functions of Ra and Pr only, as stated by the BT.

## 2 Workbench project

The geometry of the square cavity will be defined using SpaceClaim, the uniform cartesian mesh will be generated using ANSYS Meshing and the problem will be solved using ANSYS Fluent. Each of these components is directly available from WB as shown in Figure 2.

- Start WB, *Toolbox* tab on the left → drag&drop a *Fluid Flow (Fluent)* component to the main white window (*Project Schematic*), rename it as *Heated cavity*
- Right click on *Geometry* → *Properties* (menu on the right) → *Advanced Geometry Options* → *Analysis Type* → 2D
- *File* → *Save as...* → project name: HC
- Right click on *Geometry* → *New SpaceClaim Geometry...* to start SpaceClaim:





**Figure 2:** WB project (Fluid Flow with Fluent, 2D), starting SpaceClaim.

Regarding the physical parameters to consider in our numerical simulations, we have seen in Section 1.2 that any combination of  $L$ ,  $\Delta T$ ,  $\rho_0$ ,  $\nu$ ,  $g\beta$  and  $\alpha$  is equivalent as long as the Ra and Pr numbers are the same. Therefore, for the sake of simplicity, it is easier to refer to the nondimensional equations (11)-(13) which depends only on Ra and Pr, and which are based on a cavity with unit side length  $L$  and unit temperature difference  $\Delta T$  (see equations (7)). By comparing the coefficients of the nondimensional equations (11)-(13) with the ones of the dimensional equations (1)-(3) we can easily see that they match when, for example:

- $\rho_0 = 1 \text{ kg/m}^3$
- $\alpha = k/(\rho_0 \cdot c_p) = 1 \text{ m}^2/\text{s}$ , i.e.,  $k = 1 \text{ W}/(\text{m} \cdot \text{K})$  and  $c_p = 1 \text{ J}/(\text{kg} \cdot \text{K})$
- $\nu$  is numerically equal to Pr in  $\text{kg}/(\text{m} \cdot \text{s})$  units
- $g$  is numerically equal to Ra in  $\text{m}/\text{s}^2$  units
- $\beta$  is numerically equal to Pr in  $\text{K}^{-1}$  units

which are the values employed in the following (with  $L = 1 \text{ m}$  and  $\Delta T = T_h - T_c = 1 \text{ K}$ ).

## 2.1 Geometry definition with SpaceClaim

- Click on *Sketch mode* icon  at the bottom of the main graphical window and move the mouse position the sketch on the  $x - y$  plane (or click on the  $z$  axis)
- Click on *Plan view* icon  to have an orthogonal plan view:

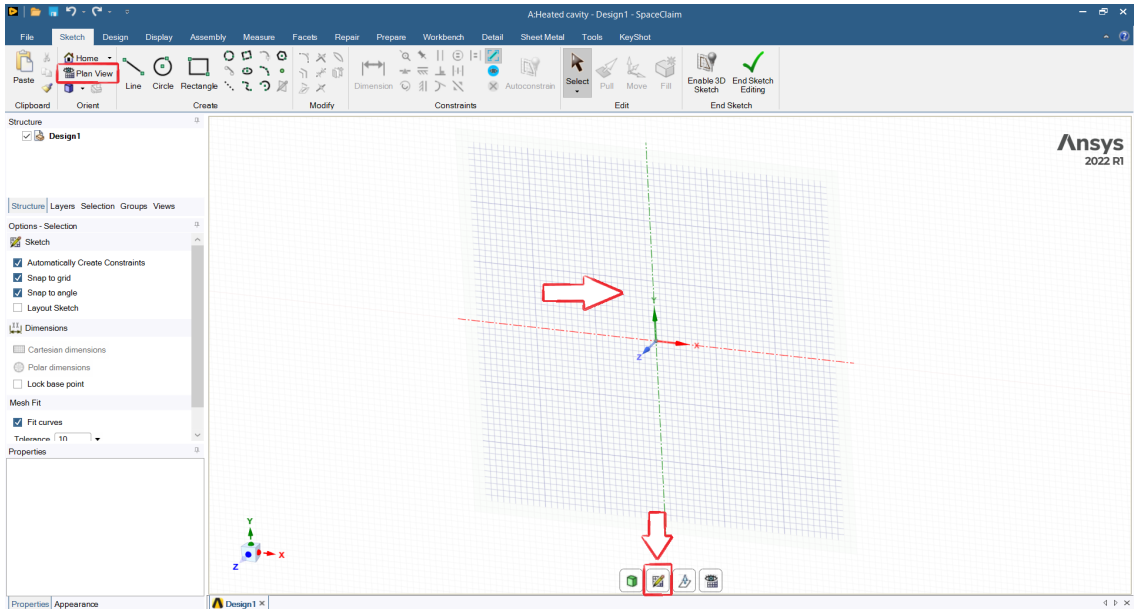


Figure 3: Sketch on the  $x - y$  plane.

- Using *Rectangle* (or *Line*), define a square with side length  $L = 1 \text{ m} = 1000 \text{ mm}$  with the bottom left corner at the origin of the  $x - y$  axes:

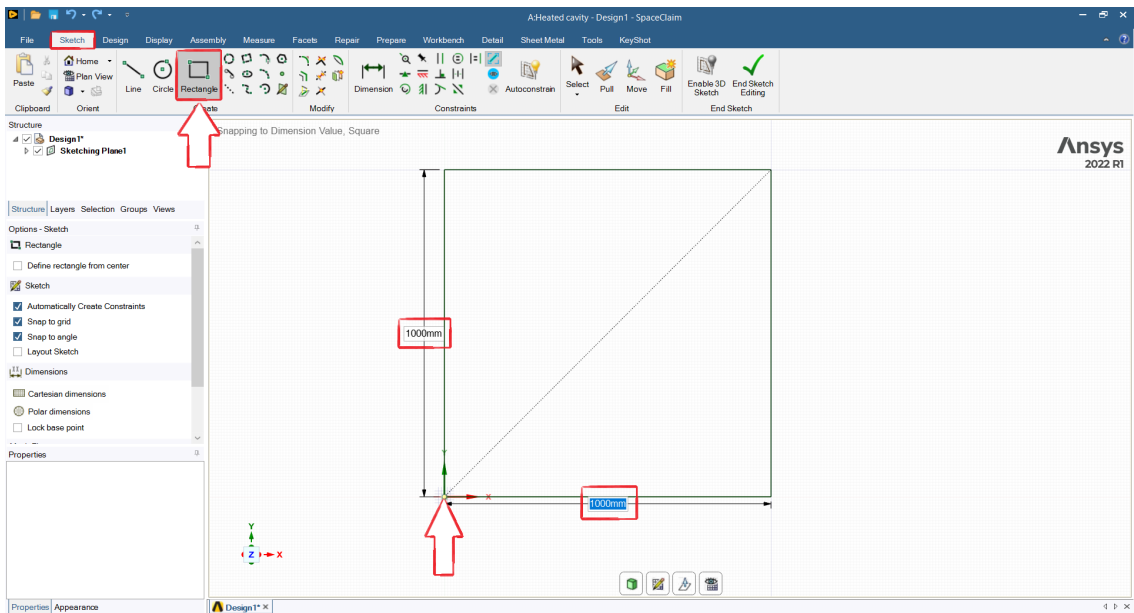


Figure 4: Sketch of the 2D square cavity.

– Click on *End Sketch Editing*:

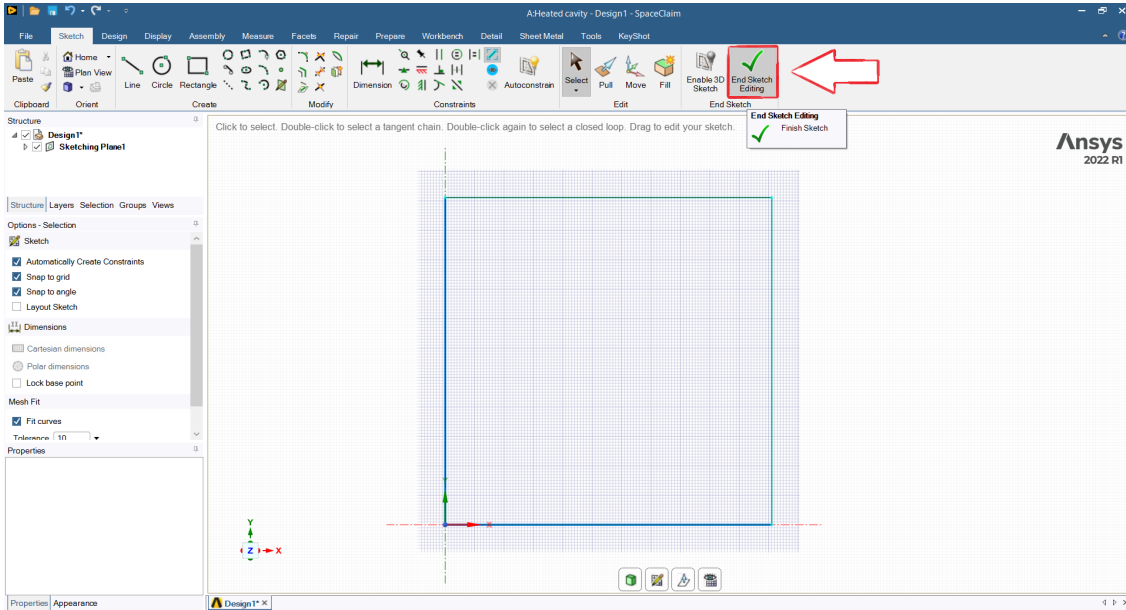


Figure 5: 2D square cavity.

– *Structure* tab on the left → right click on *Surface* → rename it to *Cavity*:

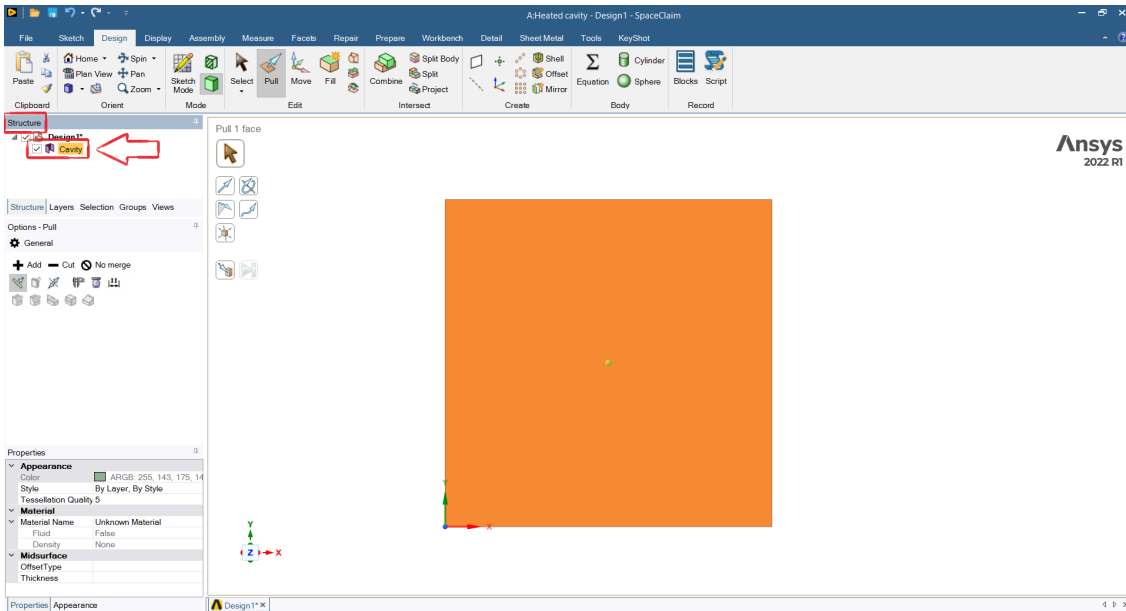


Figure 6: 2D square cavity.

– *File* → *Save Project* and close SpaceClaim.



## 2.2 Meshing the cavity with ANSYS Meshing

Now that the cavity has been defined, we can start meshing it using ANSYS Meshing. Since the geometry is a square, we will create a simple structured cartesian mesh.

– In WB, right click on *Mesh* → *Edit* to start ANSYS Meshing:

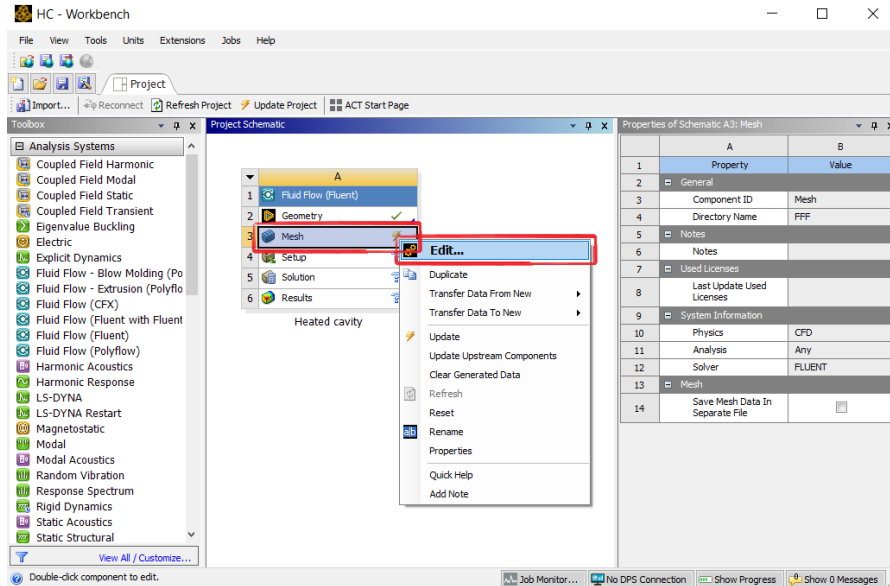



Figure 7: Starting ANSYS Meshing from WB.

– Click on the *Edge* icon  → click on the left wall → right click on it → *Create Named Selection* and specify the name *Hot*:

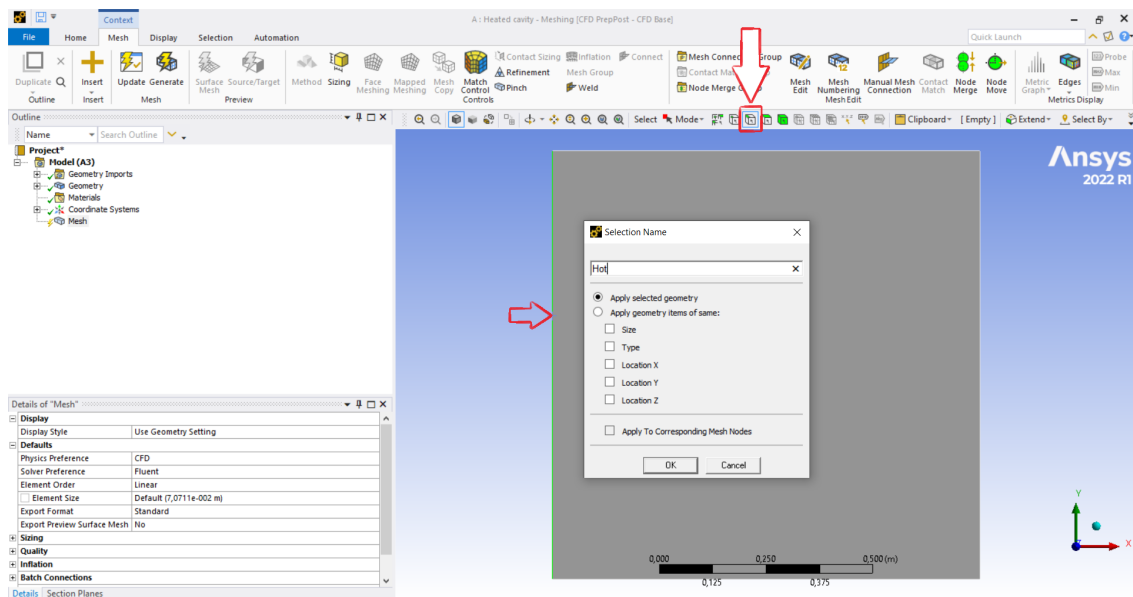


Figure 8: Named selection.

– Repeat this operation for each wall of the cavity (*Cold* for the right wall, *Top*, *Bottom*).

– Right click on *Mesh* → *Insert* → *Face Meshing*:

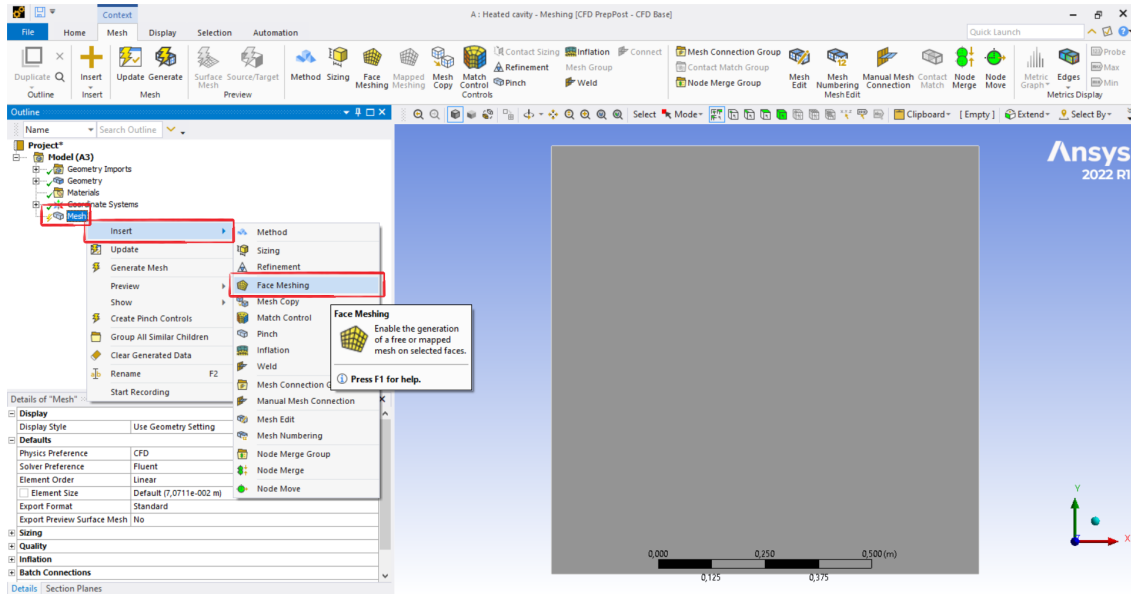


Figure 9: Face meshing.

– *Geometry* → click on *No Selection* and select the cavity → click on *Apply*:

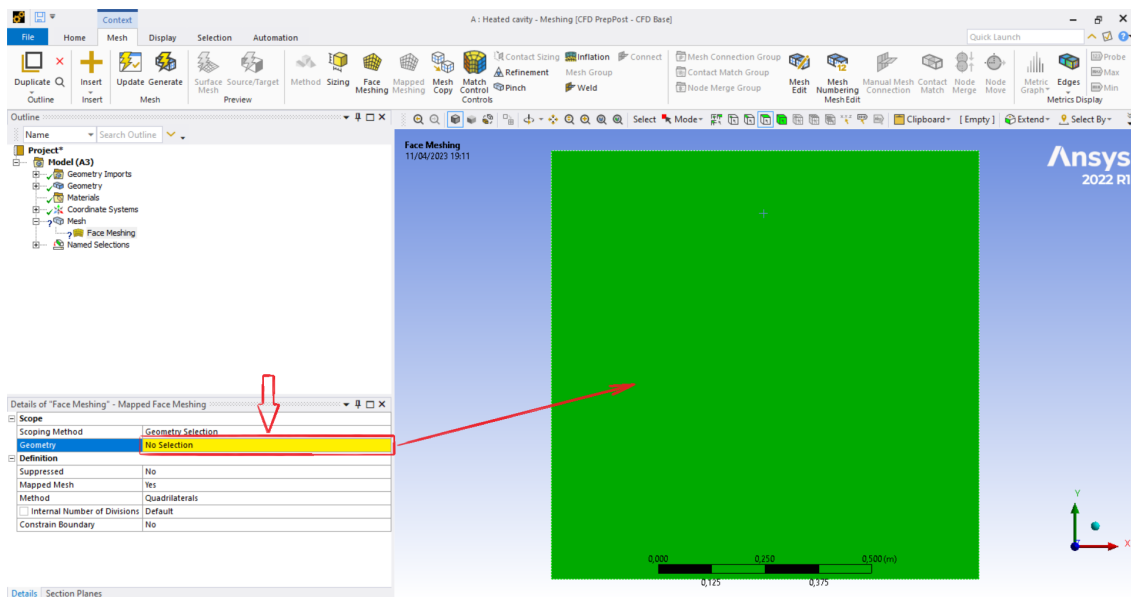
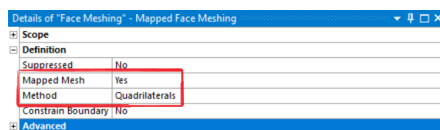


Figure 10: Face meshing.

– Check the following settings (*Mapped Mesh*: Yes, *Method*: Quadrilaterals):



– Right click on *Mesh* → *Insert* → *Sizing*:

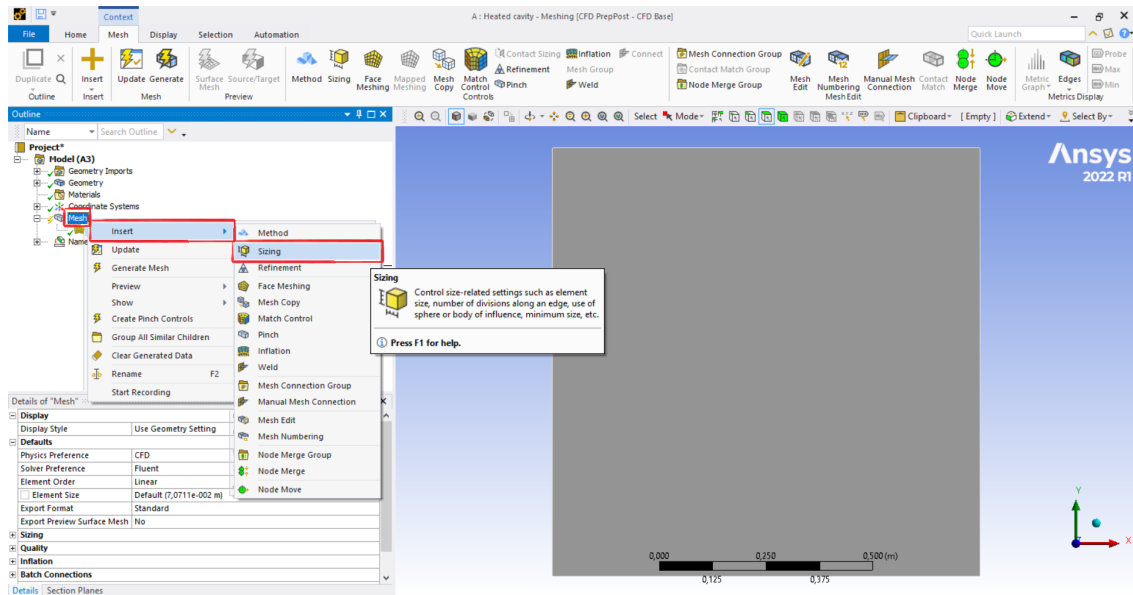



Figure 11: Sizing at the walls.

– Click on the *Edge* icon  → select all walls (hold *CTRL* to select multiple edges) → click on *Apply*:

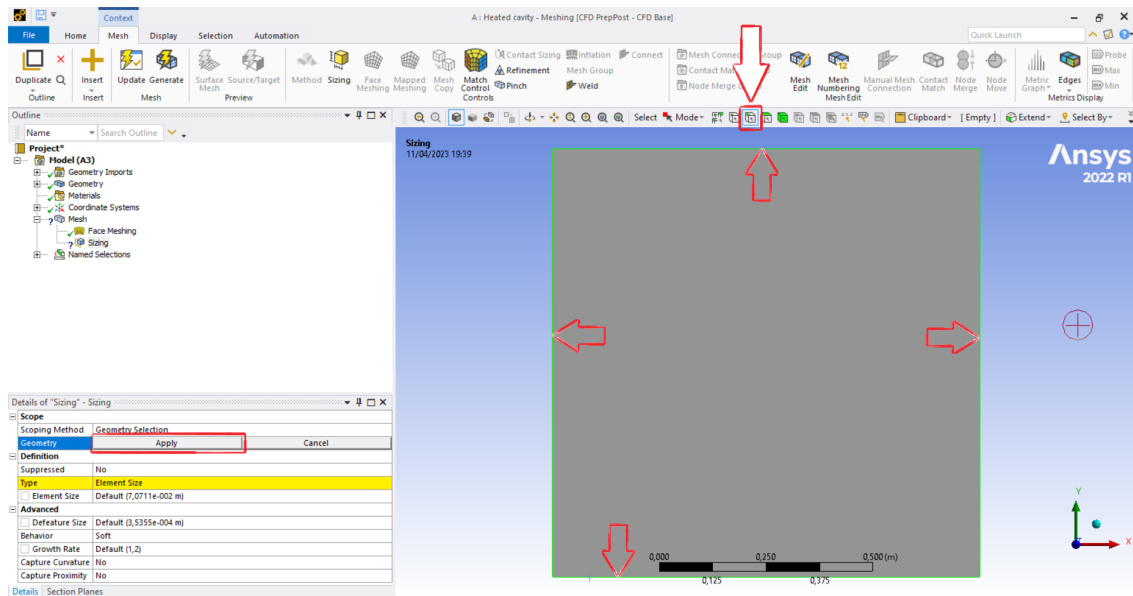


Figure 12: Sizing at the walls.

- Set the following settings:
  - *Type*: Number of Divisions
  - *Number of Divisions*: 10
  - *Behavior*: Hard
- Click on the white box next to *Number of Divisions* to set it as a new input parameter **P**:

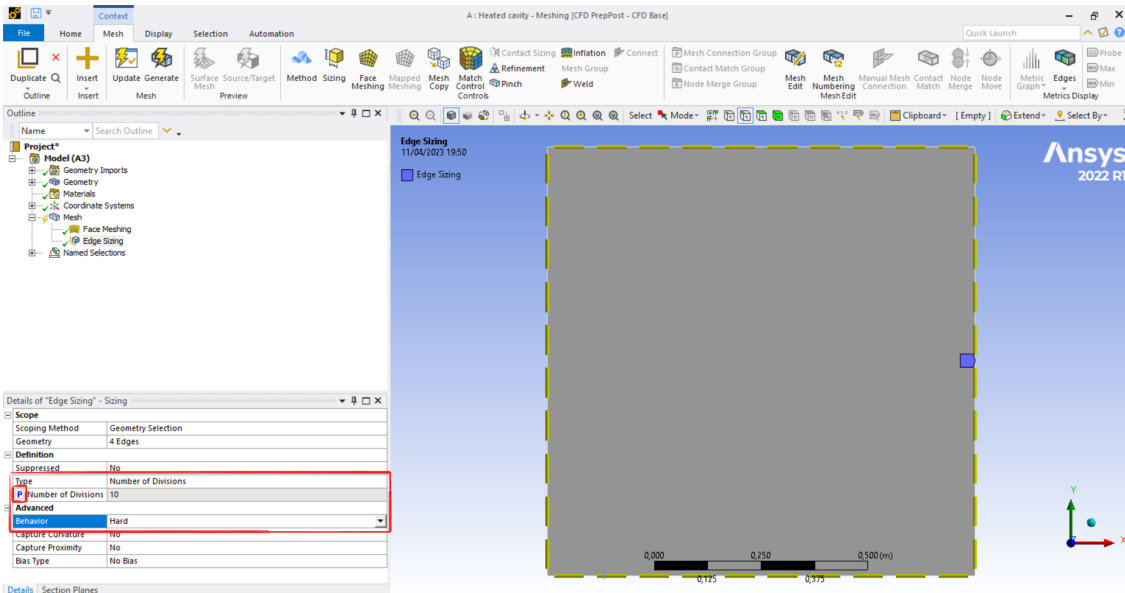


Figure 13: Sizing at the walls.

- Click on *Generate* to generate the mesh:

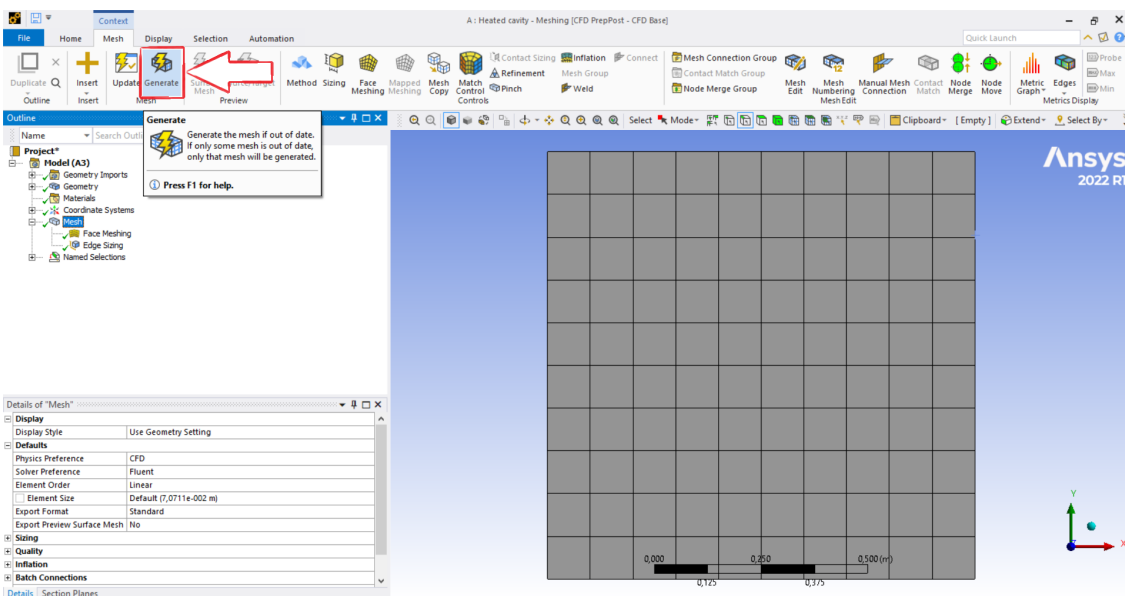


Figure 14: Cartesian mesh for the cavity.

- *File* → *Save Project* and close Meshing.

## 2.3 Setup the problem in Fluent

- In WB, right click on *Mesh* → *Update*
- Right click on *Setup* → *Edit* to run Fluent
- In the *Fluent Launcher* window, tick *Double Precision* → *Start*:

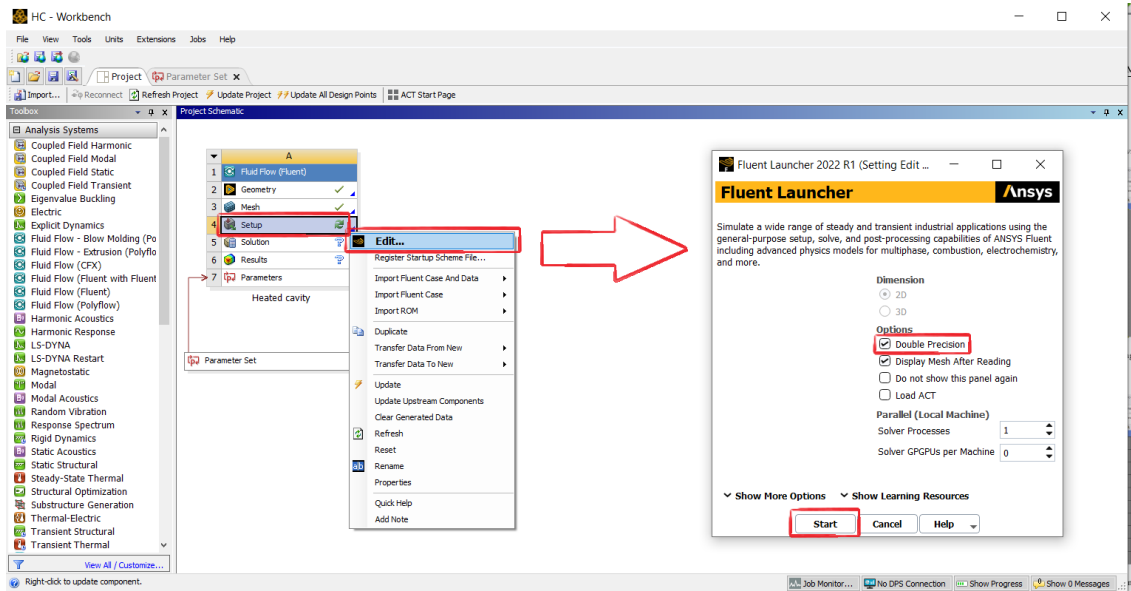


Figure 15: Starting Fluent from WB and Fluent settings.

- Tick *Gravity* → *Gravitational Acceleration Y* ( $-g$ ) → *New Input Parameter* → rename the new input parameter as *Ra*, the Rayleigh number (note the  $m/s^2$  unit)

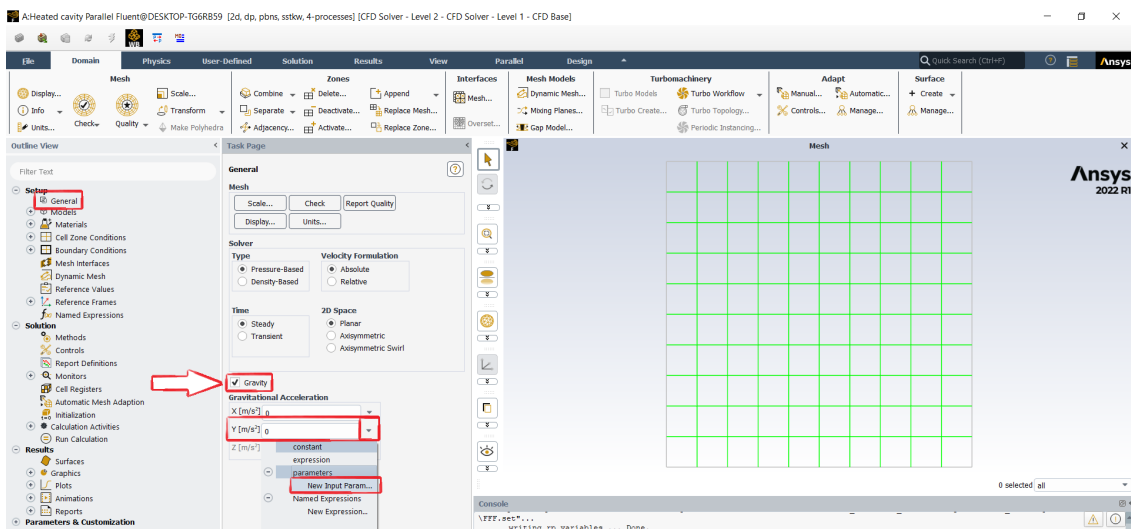


Figure 16: Fluent settings, gravity.

- Change the sign of *Gravitational Acceleration Y* from *Ra* to  $-Ra$ :
- *Models* → *Energy*: on, *Viscous*: Laminar



- *Materials* → *Fluid* → right click on *air* → *Edit*
- Set the following settings:
  - *Name*: cavity-fluid
  - *Density* ( $\rho_0$ ): boussinesq,  $1 \text{ kg/m}^3$
  - *Cp* (*Specific Heat*,  $c_p$ ): constant,  $1 \text{ J}/(\text{kg}\cdot\text{K})$
  - *Thermal Conductivity* ( $k$ ): constant,  $1 \text{ W}/(\text{m}\cdot\text{K})$
  - *Thermal Expansion Coefficient* ( $\beta$ ): New Input Parameter → rename the new input parameter as *Pr*, the Prandtl number (note the  $\text{K}^{-1}$  unit)
  - *Viscosity* ( $\nu$ ): Expression,  $\text{Pr} * 1 \text{ [K kg m}^{-1} \text{ s}^{-1}]$  (to match units)
- Click on *Change/Create* → *Change/Create mixture and Overwrite Air?* → *Yes*

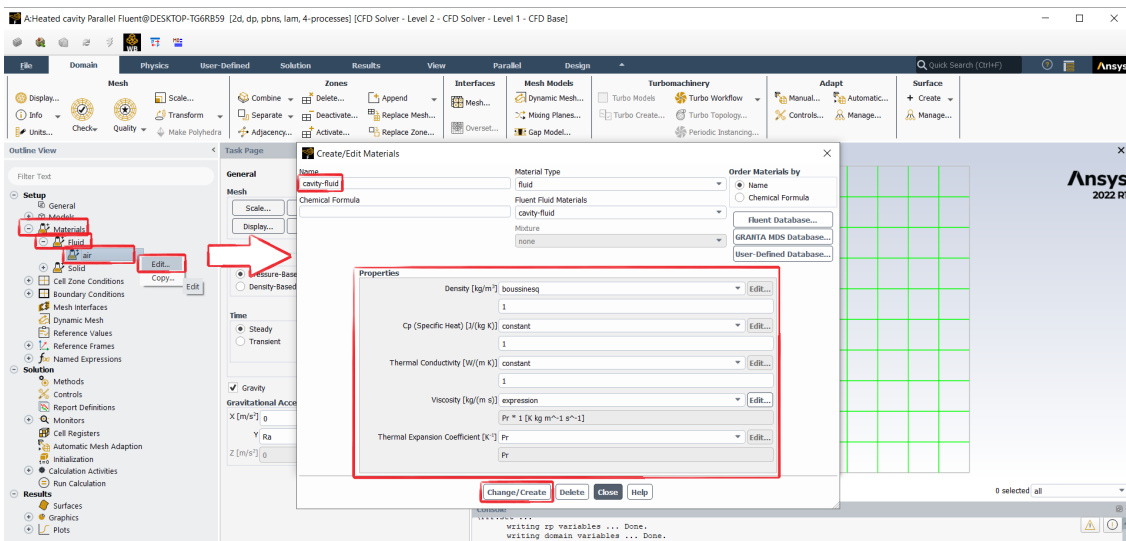


Figure 17: Fluid properties.

- *Boundary Conditions* → *Wall* → right click on *cold* → *Edit*
- *Thermal* → *Temperature* ( $T_c$ ):  $0 \text{ K}$  (negative temperatures are not allowed)

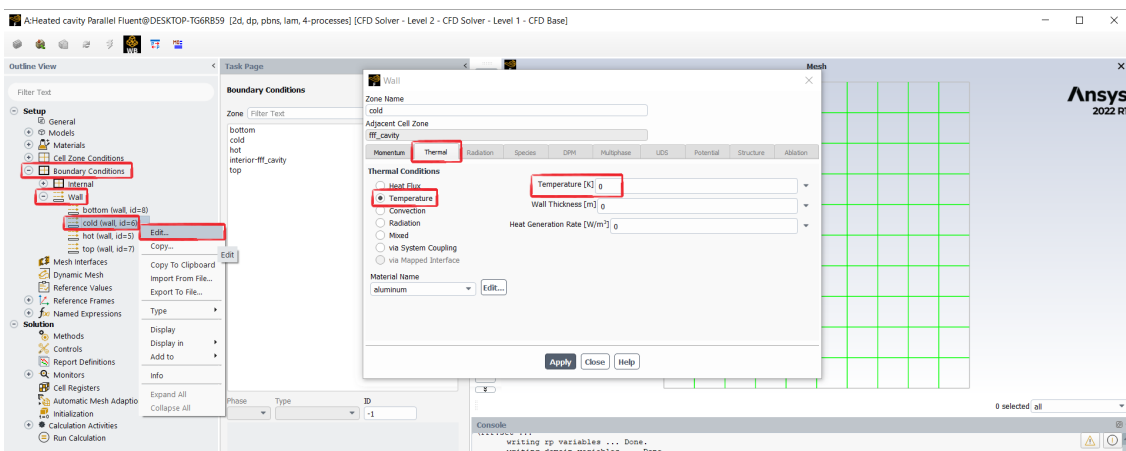


Figure 18: Isothermal boundary condition for the *cold* wall.

- Repeat the same for the *hot* wall: *Temperature* ( $T_h$ ):  $1 \text{ K}$

– *Physics* → *Operating Conditions* → *Operating Temperature* ( $T_0 = (T_h + T_c)/2$ ): 0.5 K

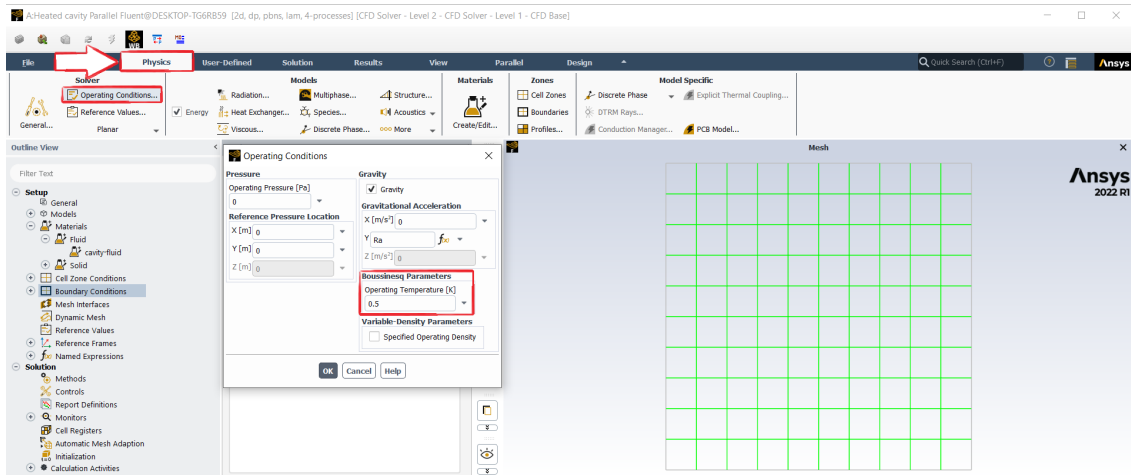


Figure 19: Setting the operating temperature.

– *Setup* → *Reference Values* → set *Length* to 1 m and *Temperature* to 0 K.

– *Console* → type solve set expert

– Allow selection of all applicable discretization schemes? → type yes

– *Methods* → *Spatial Discretization* → select *Central Differencing* for both *Momentum* and *Energy*:

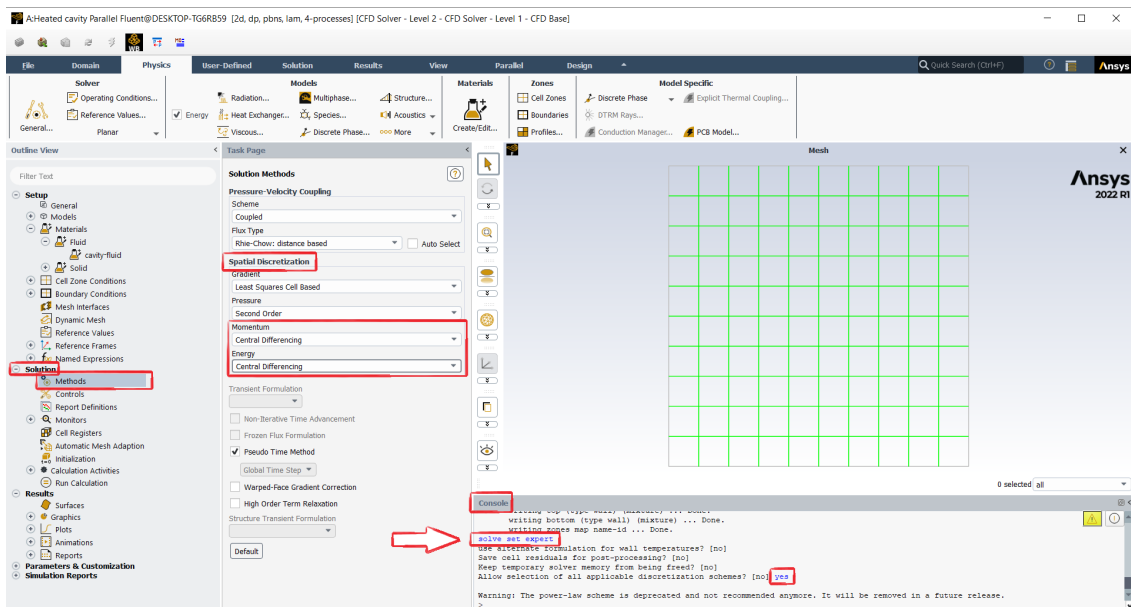
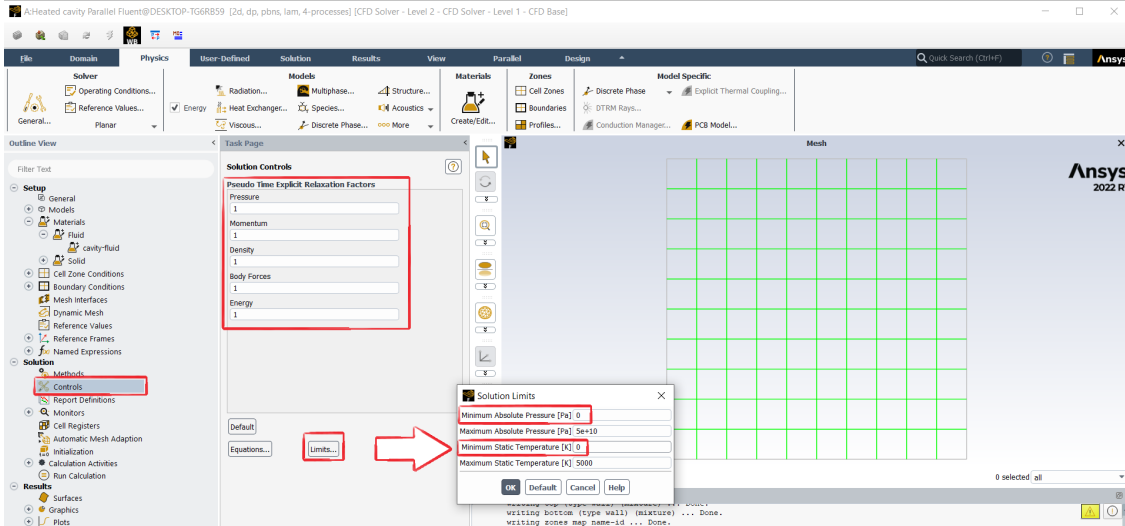


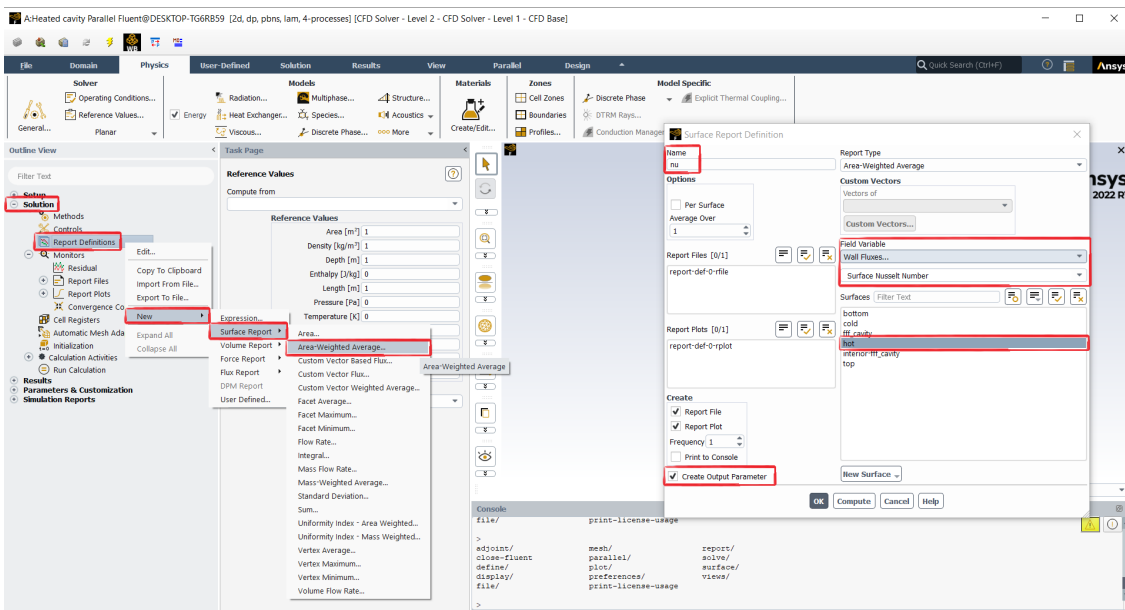
Figure 20: Setting the discretization schemes.

- *Controls* → set all relaxation factors to 1
- *Limits...* → set all minimum values to 0



**Figure 21:** Setting the relaxation factors and lower limits for pressure and temperature.

- Right click on *Report Definitions* → *New* → *Surface Report* → *Area-Weighted Average...* for the computation of the mean Nusselt number  $\bar{Nu}$ :
  - *Name* → *nu*
  - *Field Variable* → *Wall Fluxes...* → *Surface Nusselt Number*
  - *Surfaces* → *hot*
  - tick *Create Output Parameter*



**Figure 22:** Computation of  $\bar{Nu}$ .



– *Residual* → *Show Advanced Options* → *Convergence Criterion* → *none* since we'll use a fixed number of iterations:

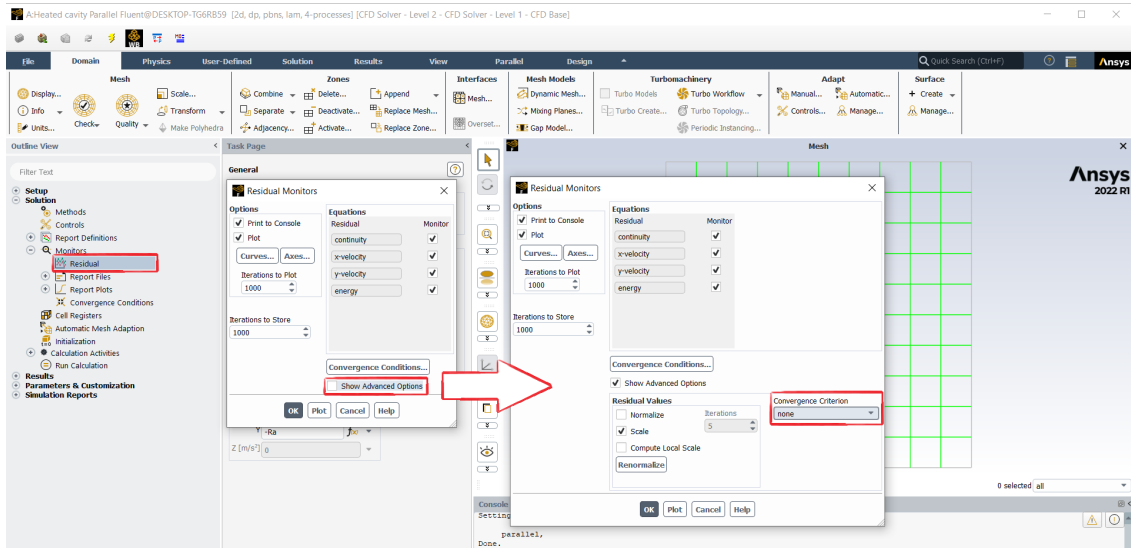


Figure 23: Convergence conditions.

- *Initialization* → *Initialize*
- Double click on *Calculation Activities* → *Create/Edit...* to save the results to file:
  - *Defined Commands:* 1
  - tick on Active
  - *Every:* 250 Iterations
  - *Command:* `file/export/ascii cell_center_values.txt () yes temperature pressure y-velocity x-velocity () yes`

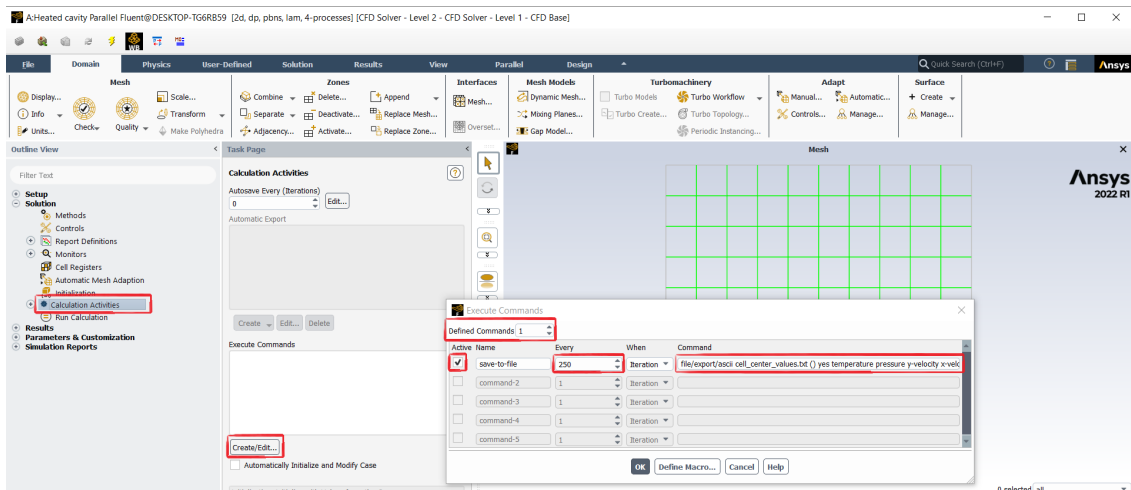


Figure 24: Save results to file.

- *Run Calculation* → *Length Scale Method:* Aggressive, *Number of Iterations:* 250
- *File* → *Save Project* and close Fluent.

- In WB, double click on *Parameter Set* and specify the values of the parameters:
  - *Edge Sizing Number of divisions*: 10,20,40,80,....,640
  - *Ra*:  $10^5$  ( $1e5$ )
  - *Pr*: 0.71 ( $0, 71$ )

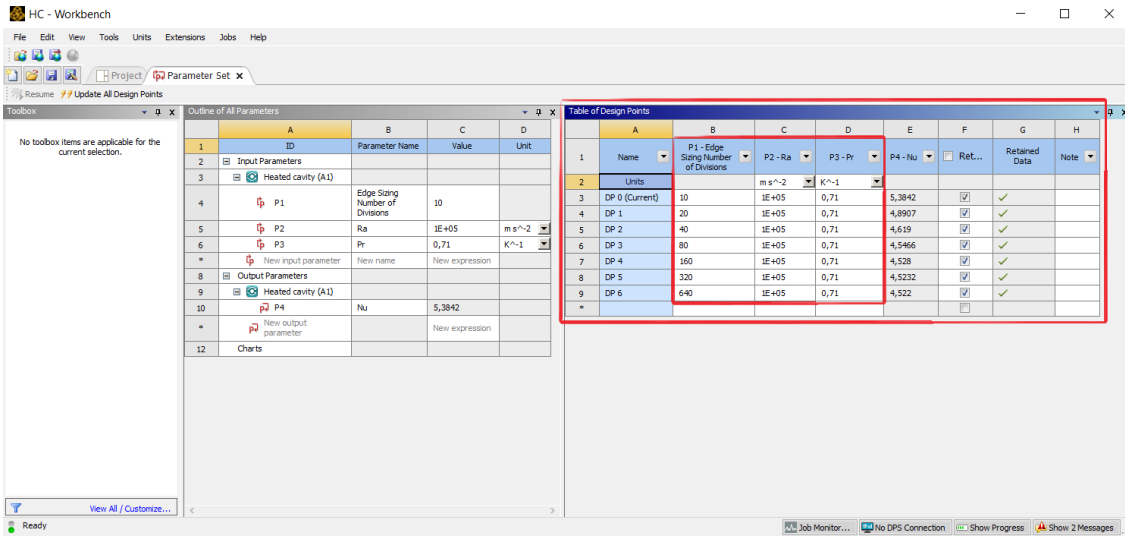


Figure 25: Parameters.

- Click on *Update all Design Points* to run the simulations for the whole set of parameters
- The values of the mean Nusselt number  $\bar{Nu}$  are displayed as output parameters
- Using file manager (File explorer/Finder/etc.) go to the project directory, i.e., where your WB project file `HC.wbpj` has been saved
- `HC_files\dp0\FFF\Fluent` is the location of the files for Design Point 0 (DP0)
- `cell_center_values.txt` is the formatted text file containing the results of the simulation, i.e., the variables at the centroid of each cell:

```

cellnumber,    x-coordinate,    y-coordinate,    x-velocity,    y-velocity,    pressure,    temperature
1, 9.500000000E-01, 5.000000000E-02, -1.046026133E+01, -5.431995278E+01, 3.941059612E+03, 3.813460988E-02
2, 9.500000000E-01, 1.500000000E-01, -1.333146451E+01, -2.096147773E+01, 5.525523339E+02, 9.241430683E-02
3, 8.500000000E-01, 5.000000000E-02, -2.880271365E+01, -4.176336734E+01, 3.287822903E+03, 9.738570710E-02
4, 9.500000000E-01, 2.500000000E-01, -5.669191625E+00, -6.513594303E+01, -1.885156534E+03, 1.616692471E-01
5, 8.500000000E-01, 1.500000000E-01, -3.219779372E+01, -4.399573009E+00, -5.219206826E+02, 2.040169185E-01
6, 7.500000000E-01, 5.000000000E-02, -4.225412273E+01, -3.020070824E+01, 2.391560976E+03, 1.303767968E-01
7, 9.500000000E-01, 3.500000000E-01, -5.073024206E+00, -6.853919810E+01, -3.819253019E+03, 2.138247619E-01
8, 8.500000000E-01, 2.500000000E-01, -1.917700808E+01, -4.329798046E+01, -2.604515360E+03, 3.207748617E-01
9, 7.500000000E-01, 1.500000000E-01, -4.421236949E+01, 1.471949386E+01, -1.109332743E+03, 2.322885827E-01
10, 6.500000000E-01, 5.000000000E-02, -3.695429154E+01, -2.769231322E+01, 1.852409333E+03, 1.491388631E-01
11, 9.500000000E-01, 4.500000000E-01, 5.003548954E-01, -7.630208568E+01, -4.496550953E+03, 2.659435075E-01
...

```

## 3 MATLAB post-processing

### 3.1 Structure of the main script

In MATLAB we'll read the results of the simulations in order to compute:

- the Nusselt number along the hot wall,  $Nu_y$ , equation (15)
- the mean Nusselt number,  $\overline{Nu}$ , equation (16)
- the streamfunction,  $\psi$ , as the solution of  $\nabla^2\psi = -\omega$
- the maximum value of  $|\psi|$ , i.e.,  $\psi_M$ , and the corresponding location  $(x_M, y_M)$

The structure of the main MATLAB script (that should be put in the same directory of the WB project file `HC.wbpj`) is the following:

```

                                PostProcessing.m
1  addpath('MATLAB_functions') ;
2
3  design_points = 0:6 ;
4  results_variables = {'Nu', 'psi_M', 'x_M', 'y_M'} ;
5  results = initialize_table(design_points, results_variables) ;
6
7  for dp = design_points
8
9      % Reading the ascii formatted file of 2D results
10     CCV_file = sprintf('HC_files\dp%d\FFF\Fluent\cell_center_values.txt', dp) ;
11     CCV      = readmatrix(CCV_file) ;
12     [~, x, y, u, v, p, T] = extract_columns(CCV) ;
13     K = matrix_of_linear_indices(x,y) ;
14     [ x, y, u, v, p, T] = vector_to_matrix(K, x, y, u, v, p, T) ;
15
16     % Nuy = -dT/dx at the hot (left) wall
17     Nuy = -( T(1,:) - 1 ) / ( x(1,1) - 0 ) ;
18
19     % Mean Nu
20     results.Nu(dp+1) = mean(Nuy) ;
21
22     % Streamfunction
23     w = vorticity(x, y, u, v) ;
24     psi = Poisson_2D_solver(x, y, -w) ;
25
26     % Maximum of |psi|
27     [psi_M, k] = max( abs(psi(:)) ) ;
28     results.psi_M(dp+1) = psi_M ;
29     results.x_M(dp+1) = x(k) ;
30     results.y_M(dp+1) = y(k) ;
31 end
```

– Line 1: definition of the directory where the required [user-defined functions](#) are stored, i.e., `MATLAB_functions\`.

– Lines 3-5: we define the design points to read, the names of the quantities that we want to compute for each design, and then we initialize the table used to store these quantities.

– Lines 7-31: the main `for` loop over each design point.

– Lines 10-14: we read the formatted text file containing the results, i.e., the coordinates  $x$  and  $y$  of the centroids of the cells, the velocities  $u$  and  $v$ , pressure  $p$  and temperature  $T$  at the centroids. Some manipulation is required to rearrange the unstructured order of the cells in Fluent into structured data, i.e., 2D matrices. After line 14,  $x$ ,  $y$ ,  $u$ ,  $v$ ,  $p$  and  $T$  are  $N_x \times N_y$  matrices, where  $N_x$  and  $N_y$  are the number of cells along  $x$  and  $y$ , respectively.

– Line 17: computation of the vector  $Nu_y$  of the values  $Nu_y$  at the hot wall through a finite difference, equation (15).

- Line 20: computation of  $\overline{Nu}$  through a simple average of vector  $Nu_y$ , equation (16), and storage in the `results` table.
- Lines 23-24: computation of vorticity  $\omega$  and streamfunction  $\psi$  at cell centers.
- Lines 27-30: computation of the maximum value of  $|\psi|$  and its corresponding location, and their storage in the results.

### 3.2 Some user-defined functions

The most important user-defined functions employed in the main script are listed as follows.

`matrix_of_linear_indices`: this function takes the vectors  $x$  and  $y$  of the coordinates of the cell centroids and returns the matrix  $K$  such that, given a vector  $v$  of any flow variable computed by Fluent,  $v(K)$  is the corresponding structured matrix.  $K$  is therefore the  $N_x \times N_y$  matrix containing the indices of the cells in structured order. The vectors  $x$  and  $y$  must be obtained from a cartesian structured grid (not necessarily uniform nor square):

`matrix_of_linear_indices.m`

```

1 function K = matrix_of_linear_indices(x,y)
2 [y, K] = sort(y) ;
3 Nx = find( y-y(1) > 1e-6, 1) - 1 ;
4 Ncells = length(y) ;
5 Ny = Ncells / Nx ;
6 K = reshape(K, Nx, Ny) ;
7 for j = 1 : Ny
8     x_j = x( K(:,j) ) ;
9     [~, i] = sort(x_j) ;
10    K(:,j) = K(i,j) ;
11 end
12 end

```

- Line 2: the vector  $y$  is sorted with increasing order, i.e., the cells are sorted along  $y$ .  $K$  is the vector of indices for this sort. Since the grid is structured, we are now left with  $N_y$  horizontal rows of cells, each of which requires a sort along  $x$ .
- Line 3-5: the number of cells  $N_x$  and  $N_y$  along  $x$  and  $y$  is computed together with the total number of cells  $N\_cells = N_x \cdot N_y$ .
- Line 6:  $K$  is reshaped into a  $N_x \times N_y$  matrix in order to  $x$ -sort each row of cells, corresponding to matrix columns, separately from one another.
- Line 7-11: the `for` loop over each row of cells.
- Line 8-10: each row of cells is sorted with increasing order along  $x$ , modifying matrix  $K$  accordingly.

`vorticity`: this function takes the matrices of coordinates  $x,y$  and the matrices of velocities  $u,v$  and computes the matrix  $w$  of the vorticity  $\omega$  at the cell centers by using central finite differences with constant step  $2\Delta x$  along  $x$  and  $2\Delta y$  along  $y$ :

`vorticity.m`

```

1 function w = vorticity(x, y, u, v)
2 dx = x(2,1) - x(1,1) ; % only for uniform grids, ie, dx = constant
3 dy = y(1,2) - y(1,1) ; % only for uniform grids, ie, dy = constant
4
5 [u, i, j] = add_ghost_cells(u) ;
6 [v, i, j] = add_ghost_cells(v) ;
7

```

```

8   % w = dv/dx - du/dy
9   w = ( v(i+1,j) - v(i-1,j) ) / (2*dx) - ( u(i,j+1) - u(i,j-1) ) / (2*dy) ;
10  end
11
12  function [Z, i, j] = add_ghost_cells(M)
13      [Nx, Ny] = size(M) ;
14      Z = zeros(Nx+2, Ny+2) ; % the new matrix with ghost cells
15      i = 2 : (Nx+1) ;
16      j = 2 : (Ny+1) ;
17      Z(i, j) = M ; % original cells
18      Z(i, 1) = -Z(i, 2) ; % ghost cells, south side, 0 Dirichlet
19      Z(i, end) = -Z(i, end-1) ; % ghost cells, north side, 0 Dirichlet
20      Z(1, j) = -Z(2, j) ; % ghost cells, west side, 0 Dirichlet
21      Z(end, j) = -Z(end-1, j) ; % ghost cells, east side, 0 Dirichlet
22  end

```

- Line 2-3: the (constant) cell size  $[\Delta x, \Delta y]$  is obtained from the coordinates  $x$  and  $y$ .
- Line 5-6: ghost cells are added to matrix of velocities  $u$  and  $v$  according to zero Dirichlet boundary condition  $u = 0$  at the walls. This operations is done by [add\\_ghost\\_cells](#) defined at lines 12-22. Ghost cells are added in order to simplify the computation of  $\omega$  at the boundary cells.
- Line 9: the matrix  $w$  of vorticity is computed through central finite differences in a single line by using vector indexing through vectors  $i$  and  $j$ , allowing therefore block matrix operations.
- Lines 12-22: function  $[Z, i, j] = \text{add\_ghost\_cells}(M)$  adds ghost cells to matrix  $M$ , returning  $Z$ . This function returns also vectors  $i$  and  $j$  containing the indices for the internal cells in matrix  $Z$ .
- Lines 13-14: the new matrix with 2 additional rows and columns is initialized.
- Lines 15-16: vectors  $i$  and  $j$  contain the row and columns indices of the internal cells in matrix  $Z$ :  $Z(i, j)$  is therefore the original matrix  $M$ , which is copied in the same positions of  $Z$  at line 17.
- Lines 18-21: the values of ghost cells are set according to zero Dirichlet boundary conditions. This is done by using vectors  $i$  and  $j$  for each side of the square boundary.

[Poisson\\_2D\\_solver](#)( $x, y, b$ ): a FV solver for a Poisson equation on a cartesian  $N_x \times N_y$  uniform rectangular grid, i.e.,  $\Delta x$  and  $\Delta y$  are constant but not necessarily equal.  $x$ ,  $y$  and  $b$  are the  $N_x \times N_y$  matrices of  $x$  and  $y$  coordinates of the centroids of the cells, and right hand side term  $b$  at the centroids, respectively. This solver has already been illustrated during a previous MATLAB lesson on solving PDEs.

### 3.3 Results

By running the main script `PostProcessing.m`, the results can be displayed by typing results in the MATLAB Command Window:

```
>> results
results =
  7x4 table
      Nu      psi_M      x_M      y_M
  _____  _____  _____  _____
  5.3842    8.1345      0.35      0.65
  4.8907    9.1764      0.725     0.375
  4.619     9.5061      0.7125    0.3875
  4.5466    9.5884      0.71875   0.39375
  4.528     9.6101      0.28437   0.60313
  4.5232    9.6151      0.28594   0.60156
  4.522     9.6164      0.28516   0.60078
```

These results can be compared with the benchmark values obtained in [2]:  $\overline{Nu} = 4.519$ ,  $\psi_M = 9.612$  at  $(x_M, y_M) = (0.285, 0.601)$ . We can also see that the  $\overline{Nu}$  values,  $Nu$ , are identical to those directly computed by Fluent, Figure 25.

Since the simulations have been performed on a sequence of grids obtained by doubling the number of cells along each dimension, i.e., with grid sizes  $h, h/2, h/4, h/8, \dots$ , the Richardson extrapolation can be easily applied to any computed quantity  $A$ :

$$\tilde{A}(h) = A + Ch^p + O(h^{p+1}) \quad (17)$$

where  $\tilde{A}(h)$  is the approximation of  $A$  computed on a grid with size  $h$ ,  $p$  is the convergence rate of the employed method ( $p = 2$  for the standard second order FV discretization, as the one employed by Fluent) and  $C$  is an unknown constant. By writing equation (18) for a grid size  $h/2$  and solving for  $A$ , we obtain the Richardson extrapolation:

$$A = \frac{2^p \tilde{A}(h/2) - \tilde{A}(h)}{2^p - 1} + O(h^{p+1}) \quad (18)$$

which has convergence rate  $p + 1$ . The process can then eventually be iterated to obtain convergence rates  $p + 2, p + 3, \dots$ . The MATLAB implementation of the iterative Richardson extrapolation in this case is quite easy, here applied to  $A = \overline{Nu}$ :

RichardsonExtrapolation.m

```
Richardson = @(v,p) ( 2^p * v(2:end) - v(1:end-1) ) / ( 2^p - 1 ) ;
A = results.Nu ;
for p = 2:4
    A = Richardson(A,p)
end
```

which gives the following extrapolations:

$p = 2$	$p = 3$	$p = 4$
<pre>A = 4.726242 4.528405 4.522511 4.521764 4.521653 4.521638</pre>	<pre>A = 4.500142 4.521670 4.521657 4.521637 4.521636</pre>	<pre>A = 4.523105 4.521657 4.521636 4.521636</pre>

It is possible to verify the convergence rate  $p = 2$  of the results obtained with Fluent by computing the error between the  $\overline{Nu}$  values and their best approximation available, i.e., the last value obtained by the Richardson extrapolation:

```
A_best = A(end) ;
FV_error = abs( results.Nu - A_best ) ;
```

Since we want to plot the convergence curve, the grid size  $h = 1/N_x$  is needed:

```
Nx = 10 * 2 .^ design_points ;
h = 1 ./ Nx ;
```

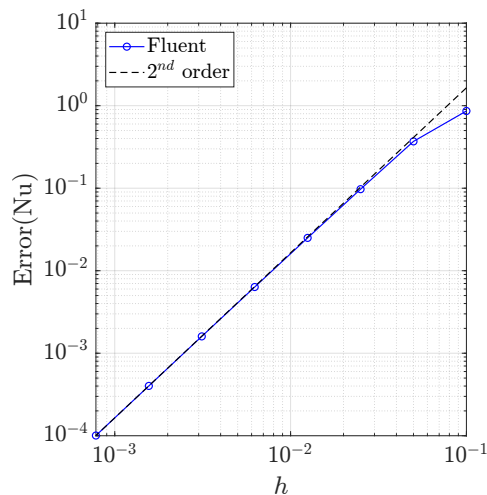
A reference  $2^{nd}$  order curve is useful for graphical comparison:

```
II_order = FV_error(end) * (h/h(end)).^2 ;
```

Plot of both curves with log-log scales, with some graphical embellishment:

```
hold on ;
plot(h, FV_error, 'o-b', 'LineWidth', 1) ;
plot(h, II_order, '--k', 'LineWidth', 1) ;
ax = gca ;
ax.XScale = 'log' ; ax.YScale = 'log' ;
ax.XGrid = 'on' ; ax.YGrid = 'on' ;
ax.XLabel.String = '$h$' ;
ax.YLabel.String = 'Error(Nu)' ;
ax.TickLabelInterpreter = 'latex' ;
ax.XLabel.Interpreter = 'latex' ;
ax.YLabel.Interpreter = 'latex' ;
ax.Box = 'on' ;
ax.FontSize = 18 ;
leg = legend('Fluent', '$2^{nd}$ order', 'Location', 'northwest') ;
leg.Interpreter = 'latex' ;
exportgraphics(gcf, 'Error_Nu.pdf') ;
```

which produces the following figure, confirming the  $2^{nd}$  order accuracy of Fluent:



**Figure 26:** Convergence curve for  $\overline{Nu}$ .

## References

- [1] G. De Vahl Davis. Natural convection of air in a square cavity: A bench mark numerical solution. *International Journal for Numerical Methods in Fluids*, 3(3):249–264, 1983.
- [2] G. De Vahl Davis and I. P. Jones. Natural convection in a square cavity: A comparison exercise. *International Journal for Numerical Methods in Fluids*, 3(3):227–248, 1983.
- [3] P. Le Quéré and M. Behnia. From onset of unsteadiness to chaos in a differentially heated square cavity. *Journal of Fluid Mechanics*, 359:81–107, 1998.