Team Assignment
Internal flow in a channel with a thin symmetric bump (2D Euler equations)

Kofler Joshua
61806971
joshua.kofler@tuwien.ac.at

January 27, 2025

1 Overview

1.1 2D Euler equations

The Euler equations neglect the effects of the viscosity of the fluid, which are included in the Navier-Stokes equations. The equations are a set of coupled differential equations. Below, one can see the simplified, two-dimensional, steady form of the Euler equations. There are two independent variables in the problem, the x and y coordinates of some domain. There are four dependent variables, the pressure p, the density r, and two components of the velocity vector; the component u is in the x direction and the component v is in the y direction. All of the dependent variables are functions of both x and y.
The equations are as follows:

$$\frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} = 0 \tag{1.1}$$

$$\frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} = -\frac{\partial \mathcal{P}}{\partial x} \tag{1.2}$$

$$\frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} = -\frac{\partial \mathcal{P}}{\partial y} \tag{1.3}$$

Here, x and y are the spatial coordinates, u and v are the velocity components, $\mathcal{P}$ is the pressure and ρ is the density. That we can rewrite in the following form

$$\frac{\partial U}{\partial t} + \nabla \cdot \mathbf{F} = Q \tag{1.4}$$

$$\frac{\partial U}{\partial t} + \frac{\partial f}{\partial x} + \frac{\partial g}{\partial y} = Q \tag{1.5}$$

$$U = \begin{pmatrix} \rho \\ \rho u \\ \rho v \\ \rho E \end{pmatrix} \tag{1.6}$$

$$f = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix}, \quad g = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix} \tag{1.7}$$

$$\oint_S \mathbf{F} \cdot d\mathbf{S} = \oint_{ABCD} (f dy - g dx) \tag{1.8}$$

Using a finite volume approach

$$\frac{\partial}{\partial t}\int_{\Omega} U \; d\,\Omega + \oint_{S} \mathbf{F}\cdot d\mathbf{S} = 0 \tag{1.9}$$

$$\frac{d}{dt}(U_J \Omega_J) + \sum_{faces} \mathbf{F}\cdot\Delta\mathbf{S} = 0 \tag{1.10}$$

To calculate the gradient of the fluxes one

$$\begin{aligned}
\sum_{ABCD} \mathbf{F}\cdot\Delta\mathbf{S} &= f_{AB}(y_B - y_A) - g_{AB}(x_B - x_A) && (1.11)\\
&+ f_{BC}(y_C - y_B) - g_{BC}(x_C - x_B) && (1.12)\\
&+ f_{CD}(y_D - y_C) - g_{CD}(x_D - x_C) && (1.13)\\
&+ f_{DA}(y_A - y_D) - g_{DA}(x_A - x_D) && (1.14)
\end{aligned}$$

In our case $x_A = x_B$ and $x_C = x_D$. So that the equations simplify further.

$$\begin{aligned}
\sum_{ABCD} \mathbf{F}\cdot\Delta\mathbf{S} &= f_{AB}(y_B - y_A) && (1.15)\\
&+ f_{BC}(y_C - y_B) - g_{BC}\; dx && (1.16)\\
&+ f_{CD}(y_D - y_C) && (1.17)\\
&+ f_{DA}(y_A - y_D) - g_{DA}\; dx && (1.18)
\end{aligned}$$

Now one has to compute the fluxes in each face. One can use a

To calculate the fluxes for each cell center, we define them as follows:

$$\vec{F}_{ij} = \begin{pmatrix} f_{ij} \\ g_{ij} \end{pmatrix} \quad \text{where} \quad f_{ij} = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix}_{ij}, \quad g_{ij} = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix}_{ij} \tag{1.19}$$

Using linear interpolation, the fluxes at a cell face can be defined. For the east face, for example:

$$\vec{F}_{\left(i+\frac{1}{2}\right)j} = \frac{1}{2}\left(\vec{F}_{ij} + \vec{F}_{(i+1)j}\right) \tag{1.20}$$

To calculate the flux across the face, we multiply the interpolated flux by the face-normal vector. For the east face, the normal vector is defined as:

$$\vec{n}_{(i+\frac{1}{2})j} = \vec{n}_{AB} = \begin{pmatrix} y_B - y_A \\ -(x_B - x_A) \end{pmatrix} \cdot \frac{1}{\sqrt{(y_B - y_A)^2 + (x_B - x_A)^2}}. \tag{1.21}$$

The length of the face is given by:

$$\Delta S_{AB} = \sqrt{(y_B - y_A)^2 + (x_B - x_A)^2}. \tag{1.22}$$

The combination of these two expressions results in the face-normal vector (ndS):

$$\vec{n}_{AB} \cdot \Delta S_{AB} = \Delta \vec{S}_{AB} = \begin{pmatrix} y_B - y_A \\ -(x_B - x_A) \end{pmatrix}. \tag{1.23}$$

The computation of flux can be extended to all the faces of the control volume by applying the same principles. For each face, we calculate the flux by projecting the interpolated flux vector onto the face-normal vector and scaling by the length of the face. This ensures that the contribution of the flux through every face is accurately accounted for. The flux for each face is given as follows:

$$\left(\vec{F} \cdot \Delta \vec{S}\right)_{\left(i+\frac{1}{2}\right)j} = \frac{1}{2}\left[f_{ij} + f_{(i+1)j}\right](y_B - y_A) - \frac{1}{2}\left[g_{ij} + g_{(i+1)j}\right](x_B - x_A) \tag{1.24}$$

$$\left(\vec{F} \cdot \Delta \vec{S}\right)_{i\left(j+\frac{1}{2}\right)} = \frac{1}{2}\left[f_{ij} + f_{i(j+1)}\right](y_C - y_B) - \frac{1}{2}\left[g_{ij} + g_{i(j+1)}\right](x_C - x_B) \tag{1.25}$$

$$\left(\vec{F} \cdot \Delta \vec{S}\right)_{\left(i-\frac{1}{2}\right)j} = \frac{1}{2}\left[f_{(i-1)j} + f_{ij}\right](y_D - y_C) - \frac{1}{2}\left[g_{(i-1)j} + g_{ij}\right](x_D - x_C) \tag{1.26}$$

$$\left(\vec{F} \cdot \Delta \vec{S}\right)_{i\left(j-\frac{1}{2}\right)} = \frac{1}{2}\left[f_{i(j-1)} + f_{ij}\right](y_A - y_D) - \frac{1}{2}\left[g_{i(j-1)} + g_{ij}\right](x_A - x_D) \tag{1.27}$$

The flux along the east face is then calculated as:

$$(F \cdot \vec{n}_{\text{east}} \Delta S)_{i+\frac{1}{2}\, j} = \frac{1}{2} \left[f_{ij} + f_{i+1j} \right] (y_B - y_A) - \frac{1}{2} \left[g_{ij} + g_{i+1j} \right] (x_B - x_A).$$

If we assume $x_A = x_B$, the equation simplifies to:

$$(F \cdot \vec{n}_{\text{east}} \Delta S)_{i+\frac{1}{2}\, j} = \frac{1}{2} \left[f_{ij} + f_{i+1j} \right] (y_B - y_A).$$

Similarly, for the west face, the flux is:

$$F_{i-\frac{1}{2}\, j} = \frac{1}{2} \left(F_{i-1j} + F_{ij} \right).$$

The face-normal vector for the west face is:

$$\vec{n}_{\text{west}} \Delta S = \begin{pmatrix} y_D - y_C \\ -(x_D - x_C) \end{pmatrix}.$$

The flux along the west face is:

$$(F \cdot \vec{n}_{\text{west}} \Delta S)_{i-\frac{1}{2}\, j} = \frac{1}{2} \left[f_{i-1j} + f_{ij} \right] (y_D - y_C).$$

For neighboring cells, such as the left cell with coordinates $i_{\text{left}} = i - 1, j_{\text{left}} = j$, the flux at the east face of the left cell is:

$$(F \cdot \vec{n}_{\text{east}} \Delta S)_{i_{\text{left}}+\frac{1}{2}\, j} = \frac{1}{2} \left[f_{i_{\text{left}}j} + f_{i_{\text{left}}+1j} \right] (y_C - y_D).$$

Substituting the coordinates of the right cell gives:

$$(F \cdot \vec{n}_{\text{east}} \Delta S)_{i-1+\frac{1}{2}\, j} = \frac{1}{2} \left[f_{i-1j} + f_{ij} \right] (y_C - y_D).$$

This matches the west flux of the right cell, with the opposite sign:

$$(F \cdot \vec{n}_{\text{west}} \Delta S)_{i-\frac{1}{2}\, j} = - \left(F \cdot \vec{n}_{\text{east}} \Delta S \right)_{i-1+\frac{1}{2}\, j}.$$

This symmetry allows us to calculate the fluxes for only one set of faces (e.g., east and north) and copy them to the neighboring faces (west and south) with the appropriate sign change.

2 Boundaries

This is a most critical component of any CFD code and has to be compatible with both physical and numerical properties of the problem to be solved.
The time-dependent hyperbolic system of Euler equations contains four unknown dependent variables, and we have to determine how to handle these variables at the boundaries of the computational domain. These boundaries are of three types: solid walls, inlet and outlet boundaries, and each one of them will require a dedicated treatment. It can be shown, that by adding the energy equation a fourth eigenvalue appears equal to the first one, which becomes double valued. Since, the transport properties at a surface are determined by the normal components of the fluxes, the number and type of conditions at a boundary of a multi-dimensional domain will be defined by the eigenvalue spectrum of the Jacobian matrices associated to the normal to the boundary. Hence, at a boundary surface, the behavior of the Euler system will be determined by the propagation of waves with the following speeds:

$$\lambda_1 = \vec{v} \cdot \vec{e_n} = v_n \tag{2.1}$$

$$\lambda_2 = \vec{v} \cdot \vec{e_n} = v_n \tag{2.2}$$

$$\lambda_3 = \vec{v} \cdot \vec{e_n} + c = v_n + c \tag{2.3}$$

$$\lambda_4 = \vec{v} \cdot \vec{e_n} - c = v_n - c \tag{2.4}$$

where v_n is the normal velocity component at the considered surface.

When λ is positive, the information carried by the associated characteristics, propagates from the boundary toward the interior of the flow domain and a **physical boundary condition** has to be imposed.
On the other hand, when the eigenvalue λ is negative and the propagation occurs from the interior of the domain toward the boundary, it means that the related information is determined at the boundary by the interior flow and cannot be imposed from the outside. It will have to expressed numerically, through **numerical boundary conditions**.

In summary, the number of physical conditions to be imposed at a boundary with normal vector $\vec{n}$, pointing toward the flow domain, is defined by the number of characteristics entering the domain.

2.1 Inlet

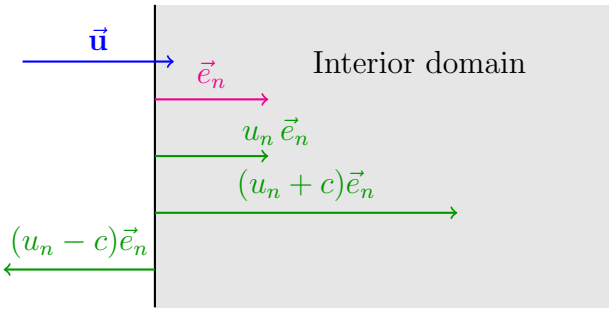

Figure 1: Characteristic propagation properties at an inlet boundary with subsonic conditions

Referring to Figure 1, for an inlet boundary with subsonic flow in the direction normal to the inlet surface, three eigenvalues are positive, while one eigenvalue, $\lambda_4 = u_n - c$, is negative. Therefore, three quantities will have to be fixed by the physical flow conditions at the inlet of the flow domain, while the remaining one will be determined by the interior conditions, through a numerical boundary condition.
A common approach, often applied to internal flows such as channels, is to specify two thermodynamic variables, such as the **upstream stagnation pressure** and **temperature**, along with the **inlet Mach number**, while allowing the inlet flow angle to be determined by the computed flow. This approach results in the mass flow not being directly specified, but rather being a consequence of the numerical solution.
To begin, the inlet flow angle can be fixed by setting the tangential velocity v_{in} to zero:

$$v_{\text{in}} = 0 \tag{2.5}$$

To determine the streamwise velocity u_{in} and the speed of sound c_{in} at the inlet, we use the Riemann invariants:

$$R^+ = u + \frac{2c}{\gamma - 1}, \quad \text{conserved along the characteristic line } \frac{dx}{dt} = u + c,$$

and

$$R^- = u - \frac{2c}{\gamma - 1}, \quad \text{conserved along the characteristic line } \frac{dx}{dt} = u - c.$$

This results in the following system of equations for determining the quantities at the inlet:

$$u_{\text{in}} + \frac{2}{\gamma - 1} c_{\text{in}} = u_\infty + \frac{2}{\gamma - 1} c_\infty, \tag{2.6}$$

$$u_{\text{in}} - \frac{2}{\gamma - 1} c_{\text{in}} = u_1 - \frac{2}{\gamma - 1} c_1 \tag{2.7}$$

where u_1 and c_1 represent the values of the respective quantities in the first cell from the previous timestep. From this system, the speed of sound at the inlet, c_{in}, can be expressed as:

$$c_{\text{in}} = \frac{\gamma - 1}{4}(u_\infty - u_1) + \frac{1}{2}(c_\infty + c_1). \tag{2.8}$$

Once c_{in} is known, the streamwise velocity at the inlet, u_{in}, is determined using:

$$u_{\text{in}} = u_1 + \frac{2}{\gamma - 1}(c_{\text{in}} - c_1). \tag{2.9}$$

For fluids in general, the speed of sound is described by the Newton-Laplace equation. In the case of an ideal gas, the inlet temperature T_{in} can then be calculated as:

$$T_{\text{in}} = \frac{c_{\text{in}}^2}{\gamma R} \tag{2.10}$$

In addition to the Riemann invariants, entropy is conserved along the characteristic line $\frac{dx}{dt} = u$, which corresponds to the streamline. From thermodynamic principles, it immediately follows that all stagnation properties, such as total temperature T_0 and total pressure p_0, remain constant along the streamlines. Consequently, by applying the isentropic relations and the equation of state for an ideal gas, the inlet pressure, p_{in}, and inlet density, ρ_{in}, can be determined.

$$\frac{p_{\text{in}}}{\rho_{\text{in}}^\gamma} = \frac{p_\infty}{\rho_\infty^\gamma}, \tag{2.11}$$

$$p_{\text{in}} = \rho_{\text{in}} R T_{\text{in}}. \tag{2.12}$$

From this system of equations, the density at the inlet, ρ_{in}, can be expressed as:

$$\rho_{\text{in}} = \left(R T_{\text{in}} \frac{\rho_\infty^\gamma}{p_\infty} \right)^{\frac{1}{\gamma - 1}} \tag{2.13}$$

The inlet pressure, p_{in}, can then be easily calculated using the equation of state for an ideal gas:

$$p_{\text{in}} = \rho_{\text{in}} R T_{\text{in}}. \tag{2.14}$$

Finally, the inlet enthalpy, H_{in}, is determined as:

$$H_{\text{in}} = c_p T_{\text{in}} + \frac{1}{2}\left(u_{\text{in}}^2 + v_{\text{in}}^2\right) \tag{2.15}$$

The calculated quantities can now be used to determine the flux vector at the west boundary of the inlet cell. Since the normal vector of the inlet cell is aligned with the streamwise direction, only the x-component of the flux vector $\vec{F} = (f, g)$ needs to be considered (see Figure 2).

$$f_w = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv + p \\ \rho uH \end{pmatrix}_{\text{in}} \tag{2.16}$$

The fluxes at the other boundaries, the east flux $\vec{F}_e$, the south flux $\vec{F}_s$, and the north flux $\vec{F}_n$ can be computed using the same method applied to the interior points, employing linear interpolation.

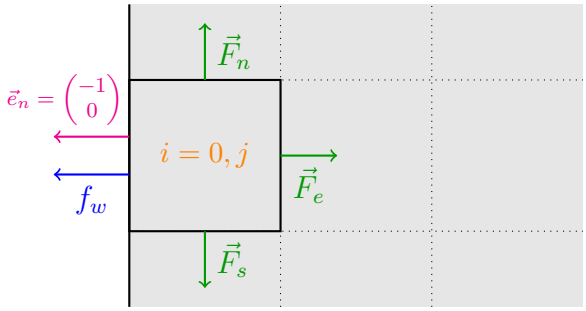

Figure 2: Illustration of the fluxes at the boundary.

2.1.1 Implementation

Initially, the values at infinity are computed under the assumption of atmospheric conditions for pressure and temperature, with $p_a = 101300$ Pa and $T_a = 288$ K, and an inlet Mach number of $M = 0.1$. To match these flow conditions, the stagnation pressure and temperature are determined using the isentropic relations:

$$T_\infty = T_a \left(1 + \frac{\gamma - 1}{2} M^2\right) = 288,576\, K \tag{2.17}$$

$$p_\infty = p_a \left(\frac{T_a}{T_0}\right)^{\frac{\gamma}{\gamma-1}} = 102010,8745\, Pa \tag{2.18}$$

Using the equation of state for an ideal gas, the Newton-Laplace equation for an ideal gas, and the definition of the Mach number, the necessary quantities at infinity are derived. These values are constant throughout the simulation and are defined at the initialization stage. The `initialize` function in `cell.py` is called at the start of the simulation, which then invokes `calculate_inlet_properties` to compute the infinity values for the fluid flow. These values are subsequently used to define the initial conditions across the entire simulation domain.

src/cell.py: Code for calculating the values at infinity

```python
def calculate_inlet_properties():
    # stagnation temperature
    gv.T_infty = ATMOSPHERIC_TEMPERATURE * (1 + (HEAT_CAPACITY_RATIO - 1)/2 * np.power(UPSTREAM_MACH_NUMBER,2))
    # stagnation pressure
    gv.p_infty = ATMOSPHERIC_PRESSURE * np.power((1 + (HEAT_CAPACITY_RATIO - 1)/2 * np.power(UPSTREAM_MACH_NUMBER,2)),
                                                 (HEAT_CAPACITY_RATIO/(HEAT_CAPACITY_RATIO-1)))
    # stagnation density
    gv.rho_infty[:] = gv.p_infty / (GAS_CONSTANT * gv.T_infty)

    gv.c_infty[:] = np.sqrt(HEAT_CAPACITY_RATIO * GAS_CONSTANT * gv.T_infty)
    gv.u_infty[:] = UPSTREAM_MACH_NUMBER * gv.c_infty

    return None
```

The expressions derived earlier for calculating the west flux at the inlet cell are implemented within the `update_flux` function in `calculate_flux.py`. After each iteration step (including every RK sub-step), the west flux is updated to reflect the changes in the values of the first cell.

src/calculate_flux.py: Code to calculate the west flux at the inlet cell

```python
    # West flux (fw) at the inlet cell (i = 0, j)
    # Set y-velocity component at inlet to 0
    v_in = 0
    # Calculate speed of sound at inlet (c_in)
    c_in = (HEAT_CAPACITY_RATIO - 1) / 4 * (gv.u_infty - gv.u[0,:,0]) + 0.5 * (gv.c_infty + gv.c[0,:])
    # Calculate inlet velocity (u_in)
    u_in = gv.u[0,:,0] + 2 / (HEAT_CAPACITY_RATIO - 1) * (c_in - gv.c[0,:])
    # Calculate inlet temperature (T_in)
    T_in = c_in**2 / (HEAT_CAPACITY_RATIO * GAS_CONSTANT)
    # Calculate inlet density (rho_in)
    rho_in = np.power(GAS_CONSTANT * T_in * gv.rho_infty**HEAT_CAPACITY_RATIO / gv.p_infty, 1/(HEAT_CAPACITY_RATIO-1))
    # Calculate inlet pressure (p_in)
    p_in = rho_in * GAS_CONSTANT * T_in
    # Calculate inlet enthalpy (H_in)
    H_in = SPECIFIC_HEAT_CP * T_in + 0.5 * (np.power(u_in, 2)) # + np.power(v_in,2))

    # West flux (fw) at the inlet cell (i = 0, j)
    gv.F[0,:,3,0] = rho_in * u_in * gv.ndS[0, :, 3, 0]                          # Mass flux (density)
    gv.F[0,:,3,1] = (rho_in * np.power(u_in, 2) + p_in) * gv.ndS[0, :, 3, 0]    # Momentum flux in x-direction
    gv.F[0,:,3,2] = 0 # rho_in * u_in * v_in * gv.ndS[0, :, 3, 0]              # Momentum flux in y-direction
    gv.F[0,:,3,3] = rho_in * u_in * H_in * gv.ndS[0, :, 3, 0]                   # Energy flux in x-direction
```

2.2 Outlet

At an outlet boundary with subsonic normal velocity, as illustrated in Figure 3, three eigenvalues are negative because the normal vector is defined as pointing inward toward the flow domain. Consequently, three numerical boundary conditions must be specified. The fourth condition, corresponding to the positive eigenvalue $\lambda_3 = -|u_n| + c$, propagates information from the boundary into the flow domain and is therefore associated with a physical boundary condition.

The most suitable physical boundary condition, particularly for internal flows and in alignment with most experimental setups, is to fix the downstream static pressure.

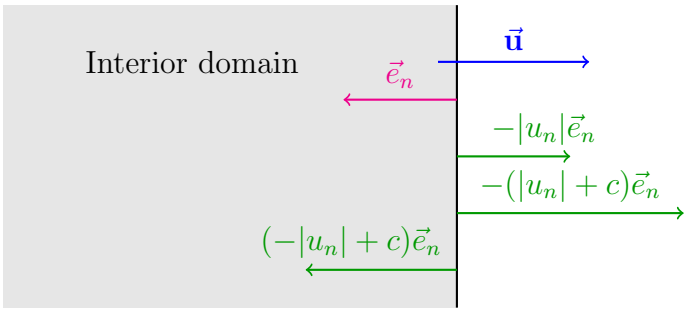

Figure 3: Characteristic propagation properties at an outlet boundary with subsonic conditions

The static pressure at the outlet boundary, p_{out}, is assumed to be equivalent to the atmospheric pressure.

$$p_{\text{out}} = p_a \tag{2.19}$$

Using the isentropic relation, the outlet density, ρ_{out}, can be determined. Here, N_x refers to the cell adjacent to the outlet boundary.

$$\rho_{\text{out}} = \rho_{N_x} \left(\frac{p_{\text{out}}}{p_{N_x}} \right)^{\frac{1}{\gamma}}. \tag{2.20}$$

Next, using the equation of state for an ideal gas, the outlet temperature T_{out} can be calculated as:

$$T_{\text{out}} = \frac{p_{\text{out}}}{\rho_{\text{out}} R}. \tag{2.21}$$

Subsequently, by applying the Newton-Laplace equation for an ideal gas, the speed of sound at the outlet, c_{out}, can be determined as follows:

$$c_{\text{out}} = \sqrt{\gamma R T_{\text{out}}}. \tag{2.22}$$

Once the speed of sound at the oulet ist known, the Riemann invariant R^+, which is conserved along the characteristic line $\frac{dx}{dt} = u + c$, can be utilized. This enables the calculation of the streamwise velocity u_{out} as:

$$u_{\text{out}} = u_{N_x} + \frac{2}{\gamma - 1} \left(c_{N_x} - c_{\text{out}} \right). \tag{2.23}$$

The tangential velocity component at the outlet, v_{out}, is given directly by:

$$v_{\text{out}} = v_{N_x}. \tag{2.24}$$

Finally, the outlet enthalpy, H_{out}, can be calculated as follows:

$$H_{\text{out}} = c_p T_{\text{out}} + \frac{1}{2} \left(u_{\text{out}}^2 + v_{\text{out}}^2 \right). \tag{2.25}$$

As with the inlet, the calculated quantities can now be used to determine the flux vector at the eastern boundary of the outlet cell. Since the normal vector of the outlet cell is also aligned with the streamwise direction, only the x-component of the flux vector $\vec{F} = (f, g)$ needs to be considered (see Figure 4).

$$f_e = \begin{pmatrix} \rho u \\ \rho u^2 \\ \rho uv + p \\ \rho uH \end{pmatrix}_{\text{out}} \tag{2.26}$$

The fluxes at the other boundaries, the west flux $\vec{F}_w$, the south flux $\vec{F}_s$, and the north flux $\vec{F}_n$ can be computed using the same method applied to the interior points, employing linear interpolation.

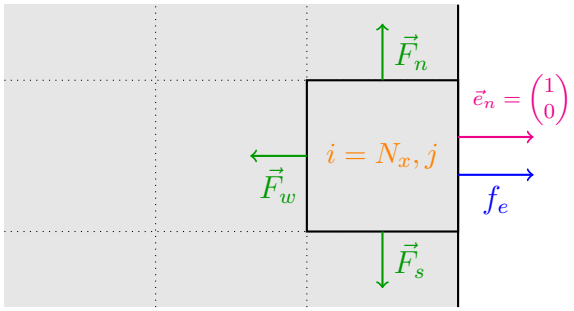

Figure 4: Illustration of the fluxes at the outlet boundary.

2.2.1 Implementation

src/calculate_flux.py: Code to calculate the east flux at the outlet cell

```
122     # East flux (fe) at the outlet cell (i = NUM_CELLS_X-1, j)
123     # Set outlet pressure (p_out) to atmospheric pressure
124     p_out = ATMOSPHERIC_PRESSURE * np.ones(NUM_CELLS_Y, 'd')
125     # Calculate outlet density (rho_out)
126     rho_out = gv.rho[-1, :] * np.power((p_out[:] / gv.p[-1, :]), 1/HEAT_CAPACITY_RATIO)
127     # Calculate outlet temperature (T_out)
128     T_out = p_out/(rho_out * GAS_CONSTANT)
129     # Calculate speed of sound at outlet (c_out)
130     c_out = np.sqrt(HEAT_CAPACITY_RATIO * GAS_CONSTANT * T_out)
131     # Calculate streamwise velocity at outlet (u_out)
132     u_out = gv.u[-1, :, 0] + 2 / (HEAT_CAPACITY_RATIO - 1) * (gv.c[-1, :] - c_out)
133     # Set y-velocity (v_out) equal to the adjacent cell's value
134     v_out = gv.u[-1, :, 1]
135     # Calculate outlet enthalpy (H_out)
136     H_out = SPECIFIC_HEAT_CP * T_out + 0.5 * (np.power(u_out, 2) + np.power(v_out, 2))
137 
138     # East flux (fe) at the outlet cell (i = NUM_CELLS_X-1, j)
139     gv.F[-1,:,1,0] = rho_out * u_out * gv.ndS[-1, :, 1, 0]                          # Mass flux (density)
140     gv.F[-1,:,1,1] = (rho_out * np.power(u_out, 2) + p_out) * gv.ndS[-1, :, 1, 0]   # Momentum flux in x-direction
141     gv.F[-1,:,1,2] = rho_out * u_out * v_out * gv.ndS[-1, :, 1, 0]                  # Momentum flux in y-direction
142     gv.F[-1,:,1,3] = rho_out * u_out * H_out * gv.ndS[-1, :, 1, 0]                  # Energy flux in x-direction
```

The expressions derived above for calculating the east flux at the outlet cell are, like those for the inlet, implemented in the `update_flux` function within `calculate_flux.py`. After each iteration step (including every RK sub-step), the east flux is updated to account for changes in the values of the last cell adjacent to the outlet boundary.

2.3 Top boundary

At a solid wall boundary, the normal velocity is zero, since no mass or other convective fluxes can penetrate the solid body. Hence only one eigenvalue is positive and only one physical condition can be imposed, namely $v_n = 0$. The other variables at the wall, in particular the tangential velocity component and the pressure have to be determined by extrapolating from the interior to the boundary.

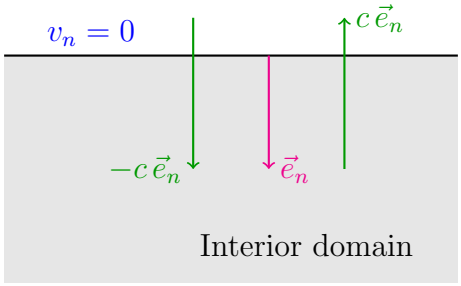

Figure 5: Characteristic propagation properties at an solid boundary

In this case, the top boundary represents a special case due to the fact that the normal vector points in the vertical direction, resulting the x-component being zero. Consequently, only the y-component of the flux vector $\vec{F} = (f, g)$ needs to be considered. Furthermore, the velocity v equals v_n. Therefore, the north flux g_n is given as:

$$g_n = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix}_{\text{top}} = \begin{pmatrix} 0 \\ 0 \\ p_{\text{top}} \\ 0 \end{pmatrix} \tag{2.27}$$

Zero-order extrapolation serves as the simplest method for determining the extrapolated value. This straightforward approach projects the value from the interior point directly onto the adjacent surface point. The method assumes that the value p is piecewise constant within the cell, implying that the boundary value is equal to the cell's mean value.

$$p_{\text{top}} = p_{N_y} \tag{2.28}$$

Here, N_y refers to the first cell adjacent to the top boundary.

The fluxes at the other boundaries, the west flux $\vec{F}_w$, the east flux $\vec{F}_e$ and the south flux $\vec{F}_s$ can be computed using the same method applied to the interior points, employing linear interpolation.

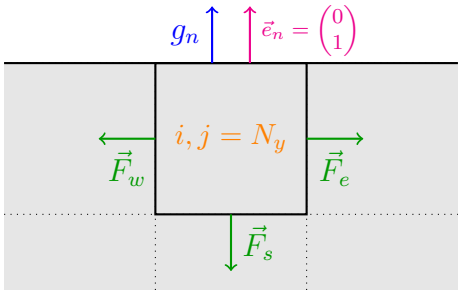

Figure 6: Illustration of the fluxes at the top boundary.

2.3.1 Implementation

Similar to the inlet and outlet boundary conditions, the top boundary condition is implemented in the `update_flux` function within `calculate_flux.py`. As a result, the north flux is updated after each iteration step, including every RK sub-step. However, unlike the inlet and outlet boundary conditions, the implementation for the top boundary is relatively straightforward.

src/calculate_flux.py: Code to calculate the north flux at the top cell

```
88    # North flux (gn) at the top wall (i, j = NUM_CELLS_Y-1)
89    gv.F[:, -1, 2, 0] = 0.0
90    gv.F[:, -1, 2, 1] = 0.0
91    gv.F[:, -1, 2, 2] = gv.p[:, -1] * gv.ndS[:, -1, 2, 1]
92    gv.F[:, -1, 2, 3] = 0.0
```

2.4 Bottom boundary

Similar to the top boundary, the normal velocity at the bottom boundary is set to zero, reflecting the physical assumption that no mass or other convective fluxes can penetrate the solid boundary.
However, since the normal vector $\vec{n}$ can point in an arbitrary direction (i.e., it does not necessarily align with one of the coordiante axes), it is necessary to account for both the x- and y-components. It is particularly important to highlight that the normal vector $\vec{n}$ is not the unit vector $\vec{e}_n$, but is instead defined as:

$$\vec{n} = \begin{pmatrix} n_x \\ n_y \end{pmatrix} = \begin{pmatrix} dy \\ -dx \end{pmatrix} = \begin{pmatrix} y_A - y_D \\ -(x_A - x_D) \end{pmatrix} \tag{2.29}$$

The flux across the bottom boundary is thus expressed as:

$$\vec{F}_s \cdot \vec{n} = f_s\, n_x + g_s\, n_y, \quad \text{with} \quad f_s = \begin{pmatrix} \rho u \\ \rho u^2 + p \\ \rho uv \\ \rho uH \end{pmatrix}, \quad \text{and} \quad g_s = \begin{pmatrix} \rho v \\ \rho uv \\ \rho v^2 + p \\ \rho vH \end{pmatrix} \tag{2.30}$$

To enforce the boundary condition that the flow does not penetrate the boundary, we impose the condition that the normal velocity is zero:

$$\vec{u}_n = \vec{u} \cdot \vec{n} = u\, n_x + v\, n_y = \vec{0} \tag{2.31}$$

Substituting this condition into the expression for the flux, the total flux simplifies to:

$$\vec{F}_s \cdot \vec{n} = f_s\, n_x + g_s\, n_y = \begin{pmatrix} 0 \\ p_{\text{bot}}\, n_x \\ p_{\text{bot}}\, n_y \\ 0 \end{pmatrix} \tag{2.32}$$

where the pressure at the boundary, p_{bot}, is determined using a zero-order extrapolation. This means that the boundary value is assumed to be equal to the mean value of the adjacent cell:

$$p_{\text{bot}} = p_0 \tag{2.33}$$

Here, the subscript 0 refers to the first cell adjacent to the bottom boundary. The fluxes at the other boundaries, the west flux $\vec{F}_w$, the east flux $\vec{F}_e$, and the north flux $\vec{F}_n$ can be computed using the same method applied to the interior points, employing linear interpolation.

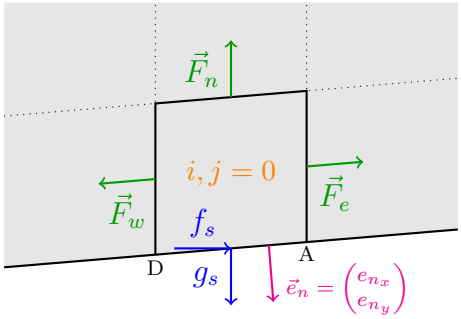

Figure 7: Illustration of the fluxes at the bottom boundary.

2.4.1 Implementation

As with the other boundary treatments, the bottom boundary condition is implemented in the `update_flux` function within `calculate_flux.py`. As a result, the south flux is recalculated and updated after every iteration step, including every RK sub-step.

src/calculate_flux.py: Code to calculate the south flux at the bottom cell

```
94    # South flux (fs, gs) at the bottom wall (i, j = 0)
95    gv.F[:, 0, 0, 0] = 0.0
96    gv.F[:, 0, 0, 1] = gv.p[:, 0] * gv.ndS[:, 0, 0, 0]
97    gv.F[:, 0, 0, 2] = gv.p[:, 0] * gv.ndS[:, 0, 0, 1]
98    gv.F[:, 0, 0, 3] = 0.0
```