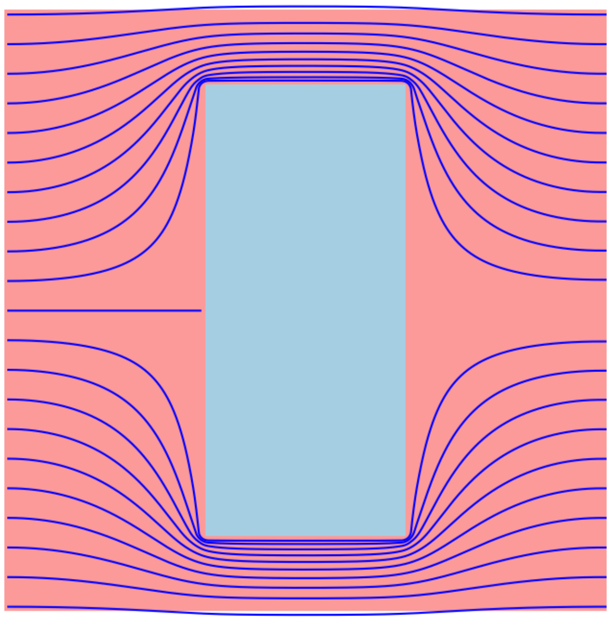
University of Franche-Comté, master P2N, 1st year
Academic year 2020-2021

Numerical project

Planar potential flows



Contact : Vincent Ballenegger
Office 312C, Institut UTINAM, Université de Franche-Comté
`vincent.ballenegger@univ-fcomte.fr`

1 Introduction and description of the project

Many fluid flows of practical interest are **potential flows**, i.e. incompressible ($\operatorname{div}\vec{v} = 0$) and irrotational ($\overrightarrow{\operatorname{rot}}\,\vec{v} = 0$). An ideal incompressible fluid impinging an obstacle with a uniform velocity is for example a potential flow. The theory of potential flows plays an essential role in aerodynamics to calculate the air flow around a wing.[1]

The goal of this project is to write a program to calculate several physical properties of potential flows in various geometries. In particular, we want to determine the velocity field, the pressure field and the streamlines. For the sake of simplicity, we consider only **planar** potential flows and we admit the following hypotheses:

- the fluid is ideal, i.e. non viscous
- the fluid is incompressible
- the flow is irrotational: the fluid particles do not rotate on themselves
- the flow is two-dimensional in the plane $Oxy \Rightarrow$ translational invariance in direction z.

The velocity field of a potential flow derives from a **velocity potential function** $\phi(x, y)$ trough

$$\vec{v} = -\overrightarrow{\nabla}\phi(x, y). \tag{1}$$

We can deduce from this relationship that the fluid velocity is perpendicular to the contours of the ϕ function (the velocity has indeed a null component in the direction where ϕ does not vary). The velocity potential function obeys Laplace's equation (see undergraduate Fluid Mechanics course)

$$\Delta\phi(x, y) = 0 \qquad \forall\,(x, y) \in D \tag{2}$$

for any point (x, y) in a domain $D \subset \mathbb{R}^2$. So, we have to solve Laplace's equation in a specified domain, with conditions on the solution ϕ adequate at the boundaries of the domain D. Such boundary conditions describe how the fluid behaves at the boundaries. They can be

- fixed velocity: $\vec{v}$ known
- impermeable wall: $\vec{v} \cdot \vec{n} = 0$ ($\vec{n}$: vector orthogonal to the boundary and oriented outwards)
- uniform flow with unknown velocity.

One has to specify which of these 3 boundary condition applies at each point of the boundary.

We will consider different geometries for the flow, that is different shapes for the D domain:

1. a straight channel
2. a channel with a widening or a shrinkage
3. a channel with an elbow
4. a straight channel containing an obstacle (the shape of this obstacle is left to your choice).

The first geometry, the straight channel, does not present any notable physical interest, but allows to test the correct operation of the program in a very simple case.

The numerical resolution of the Laplace equation will be performed by using the **finite difference method**. In this method, one introduces a regular grid, composed of $N = N_x \times N_y$ points, covering the whole D domain. The continuous equation (2) is then discretized on this grid by replacing the sum of second partial derivatives $\Delta\phi = \frac{\partial^2}{\partial x^2}\phi + \frac{\partial^2}{\partial y^2}\phi$ by differences of ϕ between neighboring points on the grid. The discretization of equation (2) leads to a system of N coupled linear equations for the values of ϕ at each grid point, which can be written as $\mathrm{A}\vec{x} = \vec{b}$, where A is a matrix of size $N \times N$. The solution of this system can be obtained by using a solver for linear systems of equations from a software library.

[1]Viscous effects are confined to a small layer towards the surfaces and the potential flow theory can be corrected to take these effects into account. Since the air speed remains typically small compared to the speed of sound, the flow of air is incompressible.

2 Specifications

The aim is to create a program with the following functionalities:

Version alpha The program calculates the flow of a fluid in the case of geometry 2 (channel with a widening or shrinkage) and displays the following results:

1. a representation of the geometry of the domain D
2. the potential field showing isopotential lines (contour lines)
3. a plot of the velocity field, and numerical values of the input and output velocities of the fluid
4. a plot of the streamlines (via the command `streamplot()` of matplotlib)
5. a plot of the pressure field with isobaric lines.

The specification of the velocity of the fluid at the inlet, the number of grid points in the x and in the y directions, and the mesh spacing h (in meters) (or the L_x and L_y extension of the physical domain under study) is done via easily identifiable variables in the program.

Version beta
The program

1. allows to study flows in the 4 geometries listed on page 2. The various parameters of these geometries (width and length of the various sections of the channel, dimension and position of the obstacle) must be easily adjustable by modifying variables in the program
   To optimize the readability of the code, functions should be introduced to build the different geometries.
2. calculates the streamlines by performing an explicit integration of the trajectories for a few fluid particles
3. works correctly in the case of a non-square physical domain ($N_x \neq N_y$).

For each geometry, the program displays, in addition to the plots of the alpha version,

1. the plot of the velocity profile in one or more sections deemed of interest
2. a plot of the streamlines obtained by integrating a few trajectories
   (Don't use the command `streamplot()` *anymore, but plot instead your calculated trajectories.)*
3. a plot of the pressure exerted on the obstacle or on the elbow deflector. The net force exerted by the fluid on the elbow deflector or on the obstacle is moreover computed.

Version gold
Such a version can feature

1. Input of various parameters (geometry, fluid velocity at the inlet, grid size, ...) as arguments to the program. (Default values should be used if no value is given.)
   Example: program invoked by a command like `./potential_flow geometry=1 vx=2 Nx=60 Ny=60`
2. Plots with a nice aesthetics and legibility
3. On the streamlines plot, markers for points with zero speed (stopping point, stagnation point)
4. Integration of streamlines using a more precise integration method than Euler's method
5. Plots showing the influence of some parameters, for example the influence of the grid spacing h on the computed streamlines, or of the time step used when computing trajectories.

→ All plots generated by the program must be saved as PDF files with a descriptive file name.
→ The program must work correctly even when the grid used for discretization contains very few points.
→ Take care of the presentation and readability of the code.

Development environment

The program must be written in **Python**. It will make use of the numerical libraries **numpy** (multidimensional arrays) and **matplotlib** (plotting library). The package **scipy** may prove useful, although not indispensable.

3 Tasks

1. Understand the structure of the system of equations resulting from the discretization of Laplace's equation as well as the conditions on $\phi(x, y)$ to be applied to the domain boundaries: see later sections 4 et 5.

2. Learn how to use the commands `pyplot.imshow()` et `numpy.meshgrid()`: see Appendix.

3. Write the Python program according to the specifications.

4. In the technical report:
   • List the limitations of the program: conditions to be respected when defining the geometry of the flow domain, cases that are not implemented, etc..
   • Include a critical analysis of the results: observed phenomena, interpretation, etc.

4 Theoretical background

4.1 Discretization by finite differences

We present the finite difference method which allows to discretize the equation (2) onto a grid to obtain a system of linear equations. The idea is to replace, in the Laplace equation, at each point P of the grid, the derivatives in $\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2}$ by potential differences with the potentials at points adjacent to the point P.

First of all, one has to discretize the fluid domain by introducing a regular grid covering the whole domain. In this project, we consider a grid with square grid cells with dimension $h \times h$. The point (i, j) on the grid has thus Cartesian coordinates $(x, y) = (ih, jh)$. At a point (x, y), the first derivative in the x direction can be estimated by one of the following finite difference formulas

$$v_x(i,j) = -\left.\frac{\partial\phi}{\partial x}\right|_{i,j} \simeq -\begin{cases} \dfrac{\phi_{i+1,j}-\phi_{i,j}}{h} & \text{(forward difference)} \\ \dfrac{\phi_{i,j}-\phi_{i-1,j}}{h} & \text{(backward difference)} \\ \dfrac{\phi_{i+1,j}-\phi_{i-1,j}}{2h} & \text{(centered difference with step } 2h) \\ \dfrac{\phi_{i+\frac{1}{2},j}-\phi_{i-\frac{1}{2},j}}{h} & \text{(centered difference with step } h\text{: involves off-grid points !)} \end{cases} \tag{3}$$

The second derivative can be approximated by applying twice the formula of a centered difference with step h.

$$\left.\frac{\partial^2\phi}{\partial x^2}\right|_{i,j} \simeq \frac{\left.\frac{\partial\phi}{\partial x}\right|_{i+1/2,j} - \left.\frac{\partial\phi}{\partial x}\right|_{i-1/2,j}}{h} \simeq \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{h^2}. \tag{4}$$

Similar formulas are used to obtain derivatives in the y direction.

The centered finite difference formulas given above allow one to approximate the Laplace equation $\Delta\phi = 0$ at an (i, j) point on the grid by

$$\frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} = 0 \qquad \Longrightarrow \qquad \frac{\phi_{i-1,j} - 2\phi_{i,j} + \phi_{i+1,j}}{h^2} + \frac{\phi_{i,j-1} - 2\phi_{i,j} + \phi_{i,j+1}}{h^2} = 0 \tag{5}$$

$$\Longrightarrow \qquad \phi_{i,j} = \frac{\phi_{i-1,j} + \phi_{i+1,j} + \phi_{i,j-1} + \phi_{i,j+1}}{4}. \tag{6}$$

The potential at an interior point (i, j) of the grid thus corresponds to the average of the potentials of the 4 adjacent points. Since the formula (6) applies only to the inner points, how can one calculate the potential at the edges of the grid? The answer follows from imposing adequate boundary conditions.

4.2 Boundary conditions

It is necessary to specify how the fluid particles behave at the boundaries of the domain. At some boundary, is there an opening through which the fluid can flow (either entering or leaving the domain) or is this boundary closed by a wall? Is the fluid velocity through an opening fixed to some value or free to adjust itself ? This specification of the physical problem under consideration translates into mathematical conditions imposed on the potential function ϕ for all points located at the boundaries of domain D.

The boundary conditions can be of two types: one can impose the value of the function (Dirichlet's condition) or its derivative (Neumann's condition).

Neumann's boundary condition A Neumann boundary condition imposes the value of the gradient of the function. Since the gradient of ϕ gives the velocity of the fluid (up to the sign), such a condition is equivalent to imposing the velocity of the fluid.

Let's suppose that we want to impose an incoming (or outgoing) velocity v_x along the x axis on the *left* edge of the domain. We then introduce a Neumann condition

$$-\left.\frac{\partial\phi}{\partial x}\right|_{x=0} = v_x \qquad \Longrightarrow \qquad -\frac{\phi_{i+1,j}-\phi_{i,j}}{h} = v_x \qquad \Longrightarrow \qquad \phi_{i,j} = \phi_{i+1,j} + v_x h. \tag{7}$$

To impose a velocity on the *right* edge, one would proceed in the same way, but using the backward finite difference formula.

An impermeable wall condition ($\vec{v}\cdot\vec{n} = 0$) results in the imposition of a null gradient of ϕ in the direction orthogonal to the wall. For example, if the lower edge of the grid is in contact with an impermeable wall, we impose the condition $\frac{\partial\phi}{\partial y}\big|_{y=0} = 0$, which can be discretized in

$$\left.\frac{\partial\phi}{\partial y}\right|_{y=0} = 0 \qquad \Longrightarrow \qquad \frac{\phi_{i,j+1}-\phi_{i,j}}{h} = 0 \qquad \Longrightarrow \qquad \phi_{i,j+1} = \phi_{i,j}. \tag{8}$$

According to eq. (8), the two points of the grid just above the wall are at the same potential. In practice, one should not actually implement eq. (8) exactly in this way, for the reason given in section 5.1, but as an average similar to the eq. (6) (see eq. (12)). The formulas (7) and (8) [or rather (12)] show how to treat grid points on the edges of the domain in the case of an impermeable wall or an in or out flow with a fixed velocity.

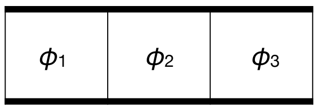
Dirichlet Boundary Condition A Dirichlet condition imposes the value of the function at the boundary. This is equivalent to imposing a reference value ϕ_{ref} on the potential:

$$\phi_i = \phi_{\text{ref}} \tag{9}$$

where ϕ_i is the potential at a point on the boundary. Since the fluid velocity is perpendicular to equipotential lines $\phi = cst$, the stream lines will cross perpendicularly a line characterized by a constant velocity potential ϕ_{ref}. A Dirichlet boundary condition applied to an outlet (region where the fluid leaves the domain) imposes therefore a uniform fluid flow, without specifying the fluid velocity.

Uniqueness of the solution It is necessary to impose a Dirichlet condition at at least one boundary of the domain. Indeed, since Laplace's equation is an equation involving only second derivatives, the solution would only be determined to within a constant if only Neumann type conditions were imposed. Imposing a Dirichlet condition amounts to imposing a reference level for the ϕ function.

To illustrate the importance of setting a reference level, let us consider the case of a domain discretized in 3 cells (3 grid points) on which one solves the Laplace equation in 1 dimension with only Neumann conditions imposing zero derivatives at both ends.



We then have the following relationships

- Laplace eq. for the inner point: $\phi_2 = \frac{\phi_1+\phi_3}{2}$ (average of the 2 adjacent cells)
- Neumann's condition for cell 1: $\phi_1 = \phi_2$
- Neumann's condition for cell 3: $\phi_2 = \phi_3$

The equations of this system are linearly dependent, so it cannot be solved uniquely. To avoid this problem, a Dirichlet condition must be imposed on at least one of the two ends.

4.3 How to compute the flow

System of equations for computing the velocity potential field $\phi(i,j)$

The first step consists in discretizing the domain on a regular grid ($N_x \times N_y$ square cells). Some cells are filled with fluid, while others are inaccessible to the fluid ("wall cells").
The second step is to assign a variable ϕ_m to each cell that is filled with fluid ($m = 1, ..., N_{\text{fluid}}$), where N_{fluid} is the number of cells occupied by the fluid.
Third, one constructs a system of linear equations $\mathrm{A}\vec{x} = \vec{b}$ by applying eq. (6) at all interior points, and by applying also the boundary conditions (Dirichlet or Neumann) for all cells at the boundaries of the domain. Each row of the A matrix contains at most 5 non-zero terms (see for instance eq. (13) below). The components of the vector $\vec{x}$ are the unknown values ϕ_m and the components of the vector $\vec{b}$ are all null, except when the index m corresponds to a cell obeying a Dirichlet or Neumann condition.
The last step is to solve the linear system to obtain the solution for the velocity potential at each point of the grid occupied by fluid. This step is easy because one can use a standard method for solving a linear system of equations, for example the solver provided by the command `numpy.solve()`. The tricky points lie more in the first 3 steps than in the resolution of the system of equations.

Calculation of the velocity field

When the potential $\phi(x,y)$ for the velocities is known at each point of the grid, the components (v_x, v_y) of the velocity can be obtained, at each point, by the formula $\vec{v} = -\vec{\nabla}\phi$. Of course, the gradient should be approximated by a finite difference formula, see eq. (3).
Note: The function `numpy.gradient()` *can be useful.*

Calculation of the pressure field

In a planar potential flow, the energy per unit volume $P + \frac{1}{2}\rho v^2$ (where $P = P(x,y)$ is the fluid pressure) remains constant at any point in the flow, and not only along a flow line as in Bernoulli's theorem (see undergraduate course in Fluid Mechanics). If the velocity and pressure of the fluid are known at the entry, and the velocity field has been determined, the formula $P + \frac{1}{2}\rho v^2 = cste$ allows to calculate the pressure at any point in the flow.

Calculation of a streamline

In a *stationary* flow, i.e. a flow that does not dependent on the time, the streamlines coincide with the trajectories of the fluid particles. The trajectory $\vec{r}(t) = (x(t), y(t))$ obeys the equation

$$\frac{d}{dt}\vec{r}(t) = \vec{v}(x(t), y(t)), \tag{10}$$

by definition of the velocity field. A streamline is obtained by integrating numerically this first-order differential equation, taking into account the initial condition (i.e. that the particle is located at a point (x_0, y_0) at time $t = 0$). Within the framework of this project, this integration can be performed by using Euler's method. *A brief description of Euler's method and the corresponding algorithm can be found for example at* §4.2.1 *of the document* `https://femto-physique.fr/omp/pdf/euler.pdf`.

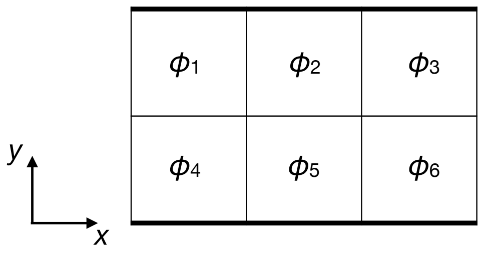
The calculation of a streamline requires to know the speed of the fluid at any point in the domain. This velocity can be obtained by interpolation[2] of the known velocities at the grid points. The command `scipy.interpolate.interp2d()` is useful to perform such an interpolation.

[2]for example bi-linear interpolation, see `https://fr.wikipedia.org/wiki/Interpolation_bilin%C3%A9aire`

5 Practical considerations

5.1 Discretization of the domain and impermeable wall condition

Let us take as an example the discretization of a channel with a constant section, length 3 m and width 2 m. We choose to discretize the domain occupied by the fluid into a very coarse grid composed of 3×2 cells.



The Laplace equation should be discretized **at the center of the different grid cells**, and not at the points where the grid lines intersect (see drawing above). Here, all 6 cells are at the edges of the flow domain. At the entrance of the channel (cells ϕ_1 and ϕ_4), we can apply the Neumann boundary condition (7) to impose the velocity of the fluid. At the outlet (cells ϕ_3 and ϕ_6), a Dirichlet condition must be imposed: $\phi_3 = \phi_6 = \phi_{\rm ref}$, where $\phi_{\rm ref}$ is an arbitrary value. By applying the condition of impermeable wall [eq. (8)] to the 6 squares in contact with a wall, we obtain the equations: $\phi_1 = \phi_4$, $\phi_2 = \phi_5$ and $\phi_3 = \phi_6$. The resulting system of equations cannot be solved, because nothing determines the common value of $\phi_2 = \phi_5$.[3]

To solve this problem in a general way, we should discretize the Neumann condition $\partial\phi/\partial y|_{y=0} = 0$ a bit differently, using a backward difference rather than the forward difference in eq. (8). For cell 5, we then obtain that ϕ_5 is equal to $\widetilde{\phi}_5$ where $\widetilde{\phi}_5$ is the potential in a fictitious cell located outside the domain, below cell 5. The equality $\phi_5 = \widetilde{\phi}_5$ can be interpreted as imposing a null gradient between these two cells, and thus a null gradient at the true location of the wall. With the addition of the fictitious cell $\widetilde{\phi}_5$, the cell ϕ_5 can be treated as an inner box. According to eq. (6), we then have

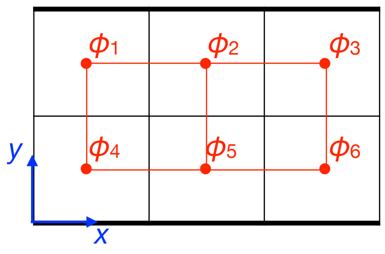
$$\phi_5 = \frac{\phi_4 + \phi_2 + \phi_6 + \widetilde{\phi}_5}{4} \qquad \text{where} \qquad \widetilde{\phi}_5 = \phi_5, \tag{11}$$

that is

$$\phi_5 = \frac{\phi_4 + \phi_2 + \phi_6}{3}. \tag{12}$$

The potential ϕ_5 is thus given by the average of the potentials of the 3 adjacent fluid cells. For a corner cell in contact with 2 impermeable walls, one would similarly find that the potential is given by the average of the 2 adjacent cells. We conclude that the cells in contact with one or more wall(s) can be treated like the inner cells, i.e. by an average of the form (6), but where the average includes only the adjacent (true) fluid cells.

As we are trying to determine the potential for the fluid velocity at the centers of the squares, it is convenient to work not with the grid represented in black, but with the grid represented in red in the figure below:



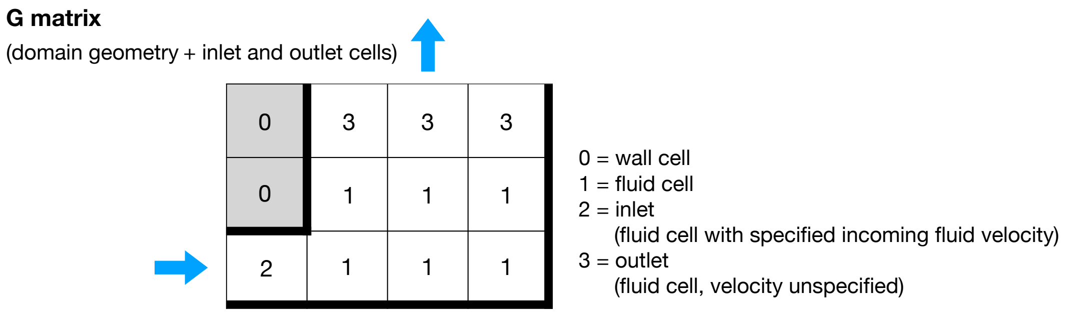
[3]Note that this problem would not occur if the grid contained at least 3 cells in the y direction.

The 6 potentials to be determined then correspond to the nodes of this grid, whose origin is located at $(x, y) = (h/2, h/2)$ where h is the grid step. In Python, the potential field can be represented by a 2-dimensional array.

`numpy` *adopts the same convention for the indexes of the components $M_{i,j}$ of a M matrix as in linear algebra: the first index of a matrix element* `M[i,j]` *corresponds to the row number, and the second index to the column number. With the Cartesian axes shown in the above figure, the axis* `i` *corresponds to the $-y$ direction, and the* `j` *axis to the x direction. It is therefore necessary to take into account any possible difference in orientation between the "matrix axes" and the Cartesian axes when writing the program. Note that it is possible to invert the orientation of the matrix axis* `i` *when representing the content of the matrix as an image. Alternatively, one can choose to orient the Cartesian axes differently (the technical report must specify the chosen orientation).*

5.2 Matrix describing the flow geometry

It is useful to introduce a two-dimensional array describing the geometry of the studied domain and the boundary conditions. The dimension of this array (named, for example, `G`) is identical to that of the matrix containing the $\phi(i, j)$ values. An example of the contents of array `G` describing a channel with a right-angle bend is shown below.

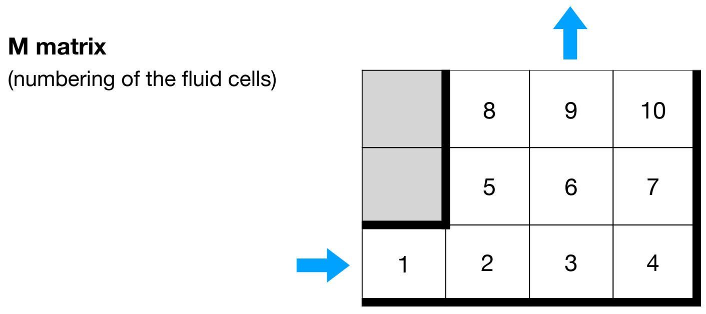

G matrix
(domain geometry + inlet and outlet cells)
0
3
3
3
0
1
1
1
2
1
1
1
0 = wall cell
1 = fluid cell
2 = inlet
(fluid cell with specified incoming fluid velocity)
3 = outlet
(fluid cell, velocity unspecified)

The element `G[i,j]` thus contains a number between 0 and 3 depending on the content of the box and on the type of boundary condition to be applied there. The locations of the walls, shown above by thick lines, can be deduced from the `G` matrix: a wall is present whenever a cell border is adjacent to a cell filled with a wall and when is it located at the boundary of the domain unless this boundary allows fluid to flow in or out.

5.3 Construction of matrix A and vectors $\vec{x}$ and $\vec{b}$

The dimension of matrix A is $N_{\text{fluid}} \times N_{\text{fluid}}$, where N_{fluid} is the number of grid cells containing fluid. To build the matrix A, we must first count the fluid cells and assign a unique number $m = 1, 2, 3, ..., N_{\text{fluid}}$ to each of them. The velocity potential of the m^{th} cell is denoted ϕ_m. The components of the vector $\vec{x}$ are these velocity potentials $\phi_1, ..., \phi_{N_{\text{fluid}}}$. One must be able to easily make the link between a cell number m and the coordinates (i, j) of the cell. Conversely, one must also be able to link a pair of coordinates (i, j) to the number m of that cell.

To make these associations, one can introduce, on one hand, an array `cell_coords(`N_{fluid}`, 2)` *in which the m^{th} row contains the coordinates i and j of cell number m, and, on the other hand, a matrix* `M` *in which the matrix element* `M[i,j]` *contains the number m.*



The general structure (and thus the construction) of the matrix A is then easily understood from an example. Consider the case of a channel turning at 90° with fluid cells numbered as above. Applying the formulas introduced in sections 4.1, 4.2 and 5.1 leads to the following linear system of equations:

$$\begin{bmatrix} -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -3 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -2 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & -3 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & -4 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & -3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \\ \phi_6 \\ \phi_7 \\ \phi_8 \\ \phi_9 \\ \phi_{10} \end{bmatrix} = \begin{bmatrix} -hv_x \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \phi_{\text{ref}} \\ \phi_{\text{ref}} \\ \phi_{\text{ref}} \end{bmatrix} \begin{array}{l} \}\ \textit{inlet: eq. (7)} \\ \left.\begin{array}{l} \\ \\ \\ \\ \\ \\ \end{array}\right\} \begin{array}{l}\textit{standard fluid cell:} \\ \textit{eqs. (6) or (12)}\end{array} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \textit{outlet: eq. (9)} \end{array} \tag{13}$$

The i^{th} equation in this system corresponds to an equation for the i^{th} cell. The 6th equation in the above system reads for instance

$$\phi_3 + \phi_5 - 4\phi_6 + \phi_7 + \phi_9 = 0, \tag{14}$$

in agreement with eq. (6). The first equation in the system corresponds to a Neumann boundary condition which sets the fluid velocity in this cell to v_x (see eq. (7)). The last 3 equations are Dirichlet boundary conditions for the outlet. The other equations correspond to averages on the adjacent fluid cells (see eq. (6) and (12)). For fluid cells other than those at the inlet or outlet, the number on the diagonal of the matrix A is equal to the opposite of the number of adjacent fluid cells. To fill-in the matrix A and the vector $\vec{b}$, one goes therefore through all fluid cells and write

- equation (7) if the fluid cell is an inlet
- equation (9) if the fluid cell is an outlet
- equation (6) or (12) otherwise, depending on the number of adjacent fluid cells.

6 Structure of the program to be written

The structure of the program should follow the different steps explained in section 4.3. From section 2 (specifications) and section 5 (practical considerations), the program will involve the following objects:

- matrix `G`: describes the geometry of the domain and the boundary conditions, i.e. the locations of the walls, inlet(s) and outlet(s).
- matrix `M` : numbering of the fluid cells (link `(i,j)` $\to$ `m`)
- array `cell_coords(`N_{fluid}`,2)` (link `m` $\to$ `(i,j)`)
- matrix A and vector $\vec{b}$ of the linear system of equations $\mathrm{A}\vec{x} = \vec{b}$
- matrix $\phi(N_x, N_y)$ that contains the velocity potential at each grid point. This matrix is computed from the solution vector $\vec{x}$ of the linear system of equations.

In order to increase the readability of the code, it is recommended to introduce functions, whenever useful, and to use the “slice" notation to access or modify chunks of arrays.

7 Algorithms

- Method for solving a linear system of equations: `numpy.solve()`
- Calculation of the gradient of a scalar field: `numpy.gradient()` or own code using finite differences
- Interpolation of a quantity from its known values at grid points: `scipy.interpolate.interp2d()` (or own code). The command `interp2d()` only works well if the field does not contain any value `nan` (**n**ot **a** **n**umber).
- Integration of a differential equation of order 1 by the Euler method: own code
  See e.g. §4.2.1 of document `https://femto-physique.fr/omp/pdf/euler.pdf`.
- Integration of a function: own code using the method of your choice (rectangle method, Simpson method, ...) or function `scipy.integrate` of the `scipy` library.
  See `https://docs.scipy.org/doc/scipy/reference/integrate.html`.

8 Appendix

The document `Plot_matrix.ipynb` (a jupyter notebook) shows

- How a matrix can be represented graphically as a color map
- How to reverse the orientation of the vertical matrix axis `i` in a color map, if desired
- How to use the function `numpy.meshgrid()` to make a grid of points.