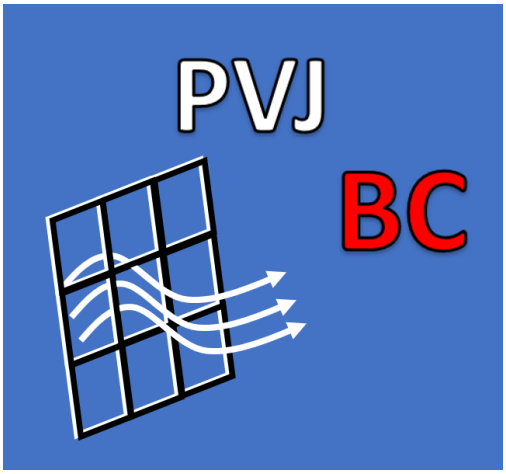
PVJ:
a pressure-velocity-jump boundary condition for OpenFOAM

Version 1.0



July 1, 2024

Contents

1 About PVJ

Thank you for your interest in *PVJ*. *PVJ* provides a modified cyclic boundary condition able to represent the presence of permeable surfaces, taking into account pressure-jumps as well as eventual velocity deflections. It is developed at the Laboratory of Computational Mechanics of University of Bologna, primarily targeting Computational Wind Engineering applications. For further info visit the website `https://site.unibo.it/cwe-lamc/en` and do not hesitate to write us in case you have any doubt.

1.1 Main features

The CFD modelling of permeable surfaces, characterized by pores which are remarkably smaller than the overall size of the immersed body, is extremely challenging, due to the inherent multi-scale nature of the problem. The *PVJ* boundary condition is a generalization of the well-known pressure-jump approach, in which forces tangential to the surface, responsible for the deflection of the velocity field, are also considered. Additionally, *PVJ* allows for a more tight control of the aerodynamic forces dependency with the angle of attack, with respect to standard approaches. *PVJ* is suitable for modelling a wide range of permeable surfaces, from perforated sheets to lamellar screens, see Fig. 1. The model is described in [1].

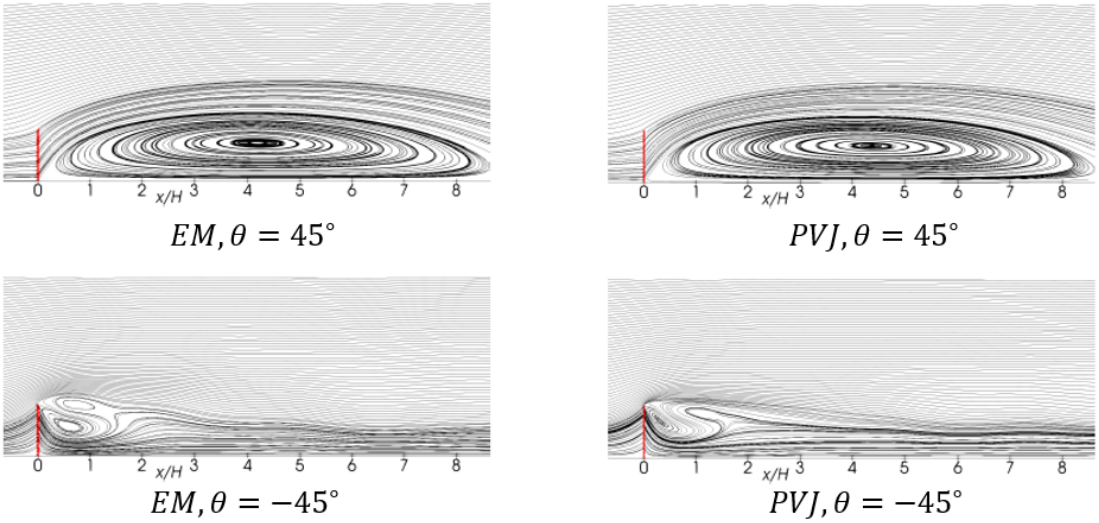


Figure 1: Example of use of the *PVJ* boundary condition to model a ground mounted lamellar barrier with different lamellae inclinations, θ. Comparison between explicit modelling of the geometry, *EM* and *PVJ* (adapted from [1]).

1.2 Licensing and compatibility

PVJ is released under the GNU GENERAL PUBLIC LICENSE Version 3. The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of merchantability, fitness for a particular purpose and non infringement. In no event shall the authors or copyright holders be liable for any claim, damages or other liability, whether in an

action of contract, tort or otherwise, arising from, out of or in connection with the software or the use or other dealings in the software.

The present implementation has been developed within OpenFOAM® v2112 and it is expected to work also for subsequent ESI-OpenCFD releases, while adaptation to the OpenFOAM Foundation version requires modifications of the source code.

2 Getting started

2.1 Set up your case

Setting up your *PVJ* BC usually requires three simple steps, reported in Fig. 2.

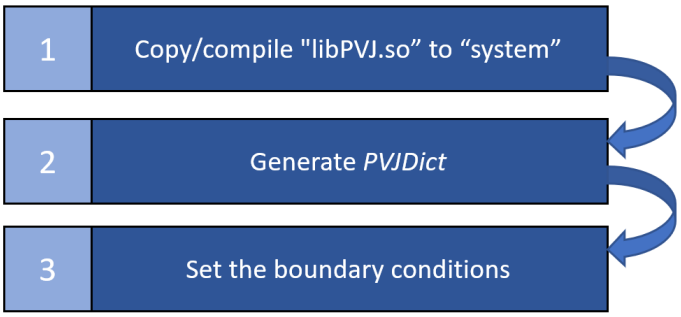


Figure 2: Steps needed to use the *PVJ* boundary condition.

Firstly, copy the *libPVJ.so* into the case main folder. If you are using it for the first time, you find it in the provided examples or in the provided *CodeOF* folder. It is anyway advisable to recompile the code for your system following the instructions provided in Section 2.2, it takes just a few seconds. Then, add the following line at the beginning of your *controlDict*:

```
libs("./libPVJ.so").
```

Once *libPVJ.so* is in your case main folder and you added it in the *controlDict*, run *PVJPreprocess.py* to generate the *PVJDict*, as described in Sec. 4.3. Finally, in the *p* and *U* files in the *0* folder, respectively select the *PJ* and *VJ* boundary conditions as shown below (left for *PJ* and right for *VJ*). In this case, *patchA* and *patchB* are here the cyclic patches pair to which *PVJ* is applied.

```
patchA {
type PJ;
patchType cyclic;
PVJDict PVJDict;
jump uniform (0);
point (100 0 0);
t1d (0 1 0);
}

patchB {
type PJ;
patchType cyclic;
PVJDict PVJDict;
jump uniform (0);
point (100 0 0);
t1d (0 1 0);
}
```

```
patchA {
type VJ;
patchType cyclic;
PVJDict PVJDict;
jump uniform (0 0 0);
point (100 0 0);
t1d (0 1 0);
}

patchB {
type VJ;
patchType cyclic;
PVJDict PVJDict;
jump uniform (0 0 0);
point (100 0 0);
t1d (0 1 0);
}
```

The meaning of the keywords is detailed below:

type: [*required*] select *PJ* for the pressure boundary condition and *VJ* for the velocity boundary conditions;
patchType: [*required*] always keep *cyclic*;
PVJDict: [*required*] the name of the *PVJDict* dictionary file that contains the data for calculating the pressure and velocity jumps based on the aerodynamic characterization of the permeable surface. The dictionary must be put in the *system* folder and it is usually generated using *PVJPreprocessing.py* as described in Sec. 4.2;
jump: [*required*] initial value of the velocity/pressure jump, use default values as reported above;
point: [*required*] coordinates of a point located along the positive normal direction of the local reference system. Further details are provided in Sec. 4.1;
t1d: [*required*] a vector identifying an approximate first tangent vector of the local reference system. Further details are provided in Sec. 4.1;

NB 1 : For each *PVJ* patches pair, a file reporting at each time step the forces acting on the permeable surface is created. The file is located in the *postProcessing* *PVJ* folder.
NB 2 : t1d: identifies an approximate tangent vector, the actual one is calculated in order to actually lay on the tangent plane of the permeable surface.
NB 3 : For cases involving more than one permeable surface typology, it is possible to use more than one *PVJDict*, all to be placed in the *system* folder. The name can be changed freely and it is here assumed to be *PVJDict* for the sake of simplicity.
NB 4 : In numerous cases, the aerodynamic properties to be considered traversing the barrier in one direction or the opposite one are the same, so that the *point* location does not have effects. In all other cases, such parameter must be correctly chosen. Examples are provided

in Fig. 3.

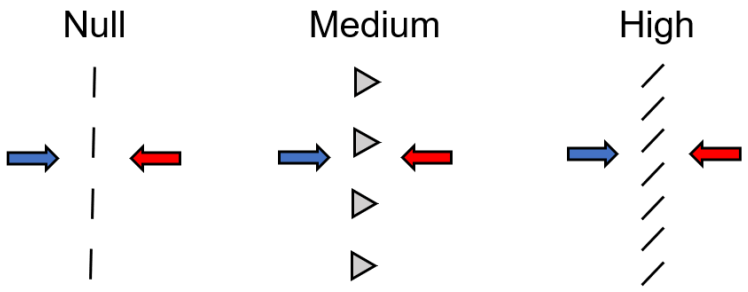


Figure 3: Expected effects induced by the crossing direction (blue and red).

2.2 Compiling *libPVJ.so*

In order to compile *libPVJ.so* for your system, open a terminal and go to the provided *CodeOF* folder. Then execute the command:

```
wmake libso
```

Then copy the newly created *libPVJ.so* into the case main folder.

3 Theoretical background

Figure 4 (a) reports a sketch of an elementary portion of a permeable surface immersed in a flow characterized by velocity $\boldsymbol{u}$ and impinging with incidence angle α with respect to the barrier normal direction, $\boldsymbol{n}$. We here assume a two-dimensional case, being the three dimensional extension of the present formulation detailed in Sec. 4.1.
For an incompressible fluid, mass conservation reads

$$u_{ni} - u_{no} = 0, \tag{1}$$

being u_{ni} the velocity component along $\boldsymbol{n}$ measured at i, while momentum conservation in the $n-$direction requires

$$\rho(u_{ni}u_{ni} - u_{no}u_{no}) + p_i - p_o - f_n = 0. \tag{2}$$

Substituting Eq. (1) in Eq. (2), we obtain

$$p_o - p_i = -f_n, \tag{3}$$

where f_n is the aerodynamic force for unit area acting on the permeable surface projected along the normal direction. The equation shows that the pressure-jump measured at the two sides of the barrier is equal, and opposite, to the force acting on the barrier in the normal direction.
Analogously, momentum conservation in the $t-$direction requires

$$\rho(u_{ti}u_{ni} - u_{to}u_{no}) - f_t = 0. \tag{4}$$

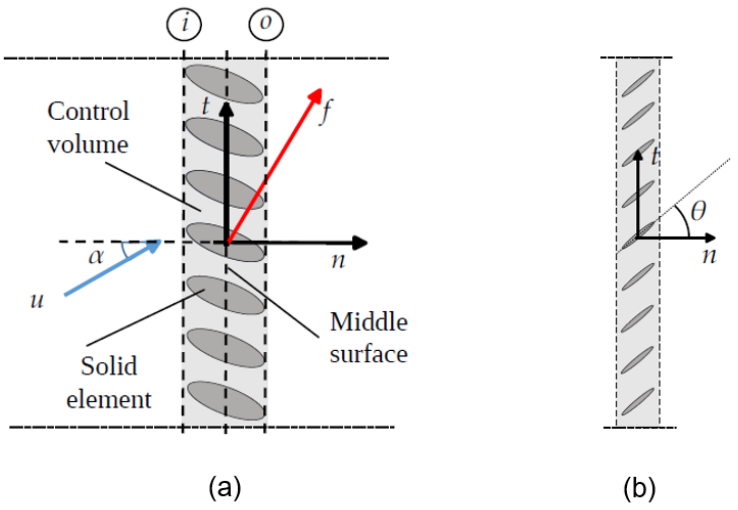


Figure 4: Sketch and main quantities involved in the *PVJ* formulation: (a) general elementary permeable surface element and (b) lamellar screen (adapted from [1]).

where the subscript t denotes projection along the tangential direction. Substituting Eq. (1) in Eq. (4), we obtain

$$\rho u_{ni}(u_{ti} - u_{to}) - f_t = 0, \tag{5}$$

so that

$$u_{to} - u_{ti} = -\frac{f_t}{\rho u_{ni}}, \tag{6}$$

which allows to calculate the jump of the tangential velocity component based on the exchanged forces and the mass flux across the permeable barrier.
Then, for all inflow angles, α, we denote the aforementioned force as $\boldsymbol{f}(\alpha) = [f_n(\alpha), f_t(\alpha)]^T$. By defining the velocity versor as $\hat{\boldsymbol{u}} = \boldsymbol{u}/|\boldsymbol{u}|$, the aerodynamic forces are assumed to be expressed as:

$$\boldsymbol{f}(\alpha) = \frac{1}{2}\rho|\boldsymbol{u}|^2|\hat{u}_n|^\gamma \boldsymbol{c}(\alpha), \tag{7}$$

where γ is a coefficient and $\boldsymbol{c}(\alpha)$ are expressed using a Fourier series as

$$\boldsymbol{c}(\alpha) = \begin{bmatrix} c_n \\ c_t \end{bmatrix} = \begin{bmatrix} b_{n0} + b_{n1}\cos(\alpha) + b_{n2}\sin(\alpha) + b_{n3}\cos(2\alpha) + b_{n4}\sin(2\alpha)... \\ b_{t0} + b_{t1}\cos(\alpha) + b_{t2}\sin(\alpha) + b_{t3}\cos(2\alpha) + b_{t4}\sin(2\alpha)... \end{bmatrix}, \tag{8}$$

in which b_{ni}, b_{ti} with $i = 0, 1...$ are model parameters which can be obtained from fitting of wind tunnel/CFD data or, for some limit cases, analytically [1].

4 Usage

As already stated, using the *PVJ* boundary condition requires to provide a dictionary which contains all the data needed to calculate the pressure and velocity jumps, here named *PVJDict*. Such dictionary is usually automatically generated relying on *PVJPrepocessing.py*, which has been developed using Python 3.9. Once generated, such dictionary must be placed in the case *system* folder.

4.1 Reference system

Before proceeding it is useful to clarify that two kind of reference systems can be used in order to obtain from the two-dimensional formulation reported in Sec. 3 a three-dimensional one.

In particular, two options are available and Fig. 5 provides a sketch of the differences between them. The first one, named **`locRef`** is useful for cases in which porosity is essentially derived by the extrusion of a two-dimensional section (e.g. lamellar screens), while the second one, named **`velRef`**, is useful for cases showing cylindrical symmetry (isotropy in the tangential directions, e.g. usual perforated porous screens).

For **`locRef`**, the positive normal direction is firstly chosen, ensuring that it points in the direction of the **`point`** mentioned in Sec. 2.1. Then, $\boldsymbol{t_1}$ is built in order to lay on the tangent plane and be oriented as closely as possible the provided **`t1d`** (see Sec. 2.1). Then $\boldsymbol{t_2}$ is obtained by vector product. In this case the velocity component acting along $\boldsymbol{t_2}$ is disregarded (i.e. no axial flow effects are considered).

For **`velRef`**, the positive normal direction is firstly chosen, ensuring that it points in the direction of the **`point`** mentioned in Sec. 2.1. Then, $\boldsymbol{t_1}$ is built in order to lay along the local velocity tangential component. In this case the provided **`t1d`** is ignored. Finally, $\boldsymbol{t_2}$ is obtained by vector product.

In both cases, tangential forces are always assumed to lay along $\boldsymbol{t_1}$, which represents in three dimensions the analogous of $\boldsymbol{t}$ used in Sec. 3 for the two-dimensional case.

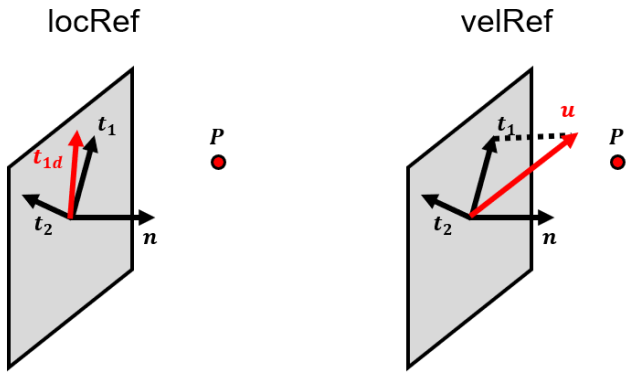


Figure 5: Possible reference systems adopted by *PVJ*.

4.2 *PVJDict*

The keywords appearing in *PVJDict* are discussed below.

`pvj_ref:` [*required*] choosing the kind of reference system to be used, see Sec. 4.1. Possible choices are **`locRef`** or **`velRef`**, as detailed in Sec. 4.1.
`pvj_gamma:` [*required*] the coefficient γ appearing in Eq. (7).
`pvj_bn:` [*required*] the Fourier coefficients appearing in the calculation of c_n in Eq. (8), following the format detailed below.
`pvj_bt1:` [*required*] the Fourier coefficients appearing in the calculation of c_t in Eq. (8), following the format detailed below.

The Fourier coefficients are stored in two-dimensional arrays for which each line contains in the order: a flag to identify cosinus (0) or sinus (1), the harmonic number, the Fourier coefficient. As an example, the line $[0, 2, 0.3]$ identifies $0.3\cos(2\alpha)$ while $[1, 3, 0.2]$ identifies $0.2\sin(3\alpha)$.

4.3 *PVJPreprocessing.py*

The routine allows to generate the *PVJDict*, relying on the library *PVJLib.py*. The first lines of the code shall be modified by the user in agreement with the following. Then the code is run, generating the *PVJDict* to be copied in the *system* folder. Only the following variables shell be edited before running:

`outDictionaryName:` name of the produced output file, generally *PVJDict*.
`inputDataFile:` path to a text file collecting data to be compared/used (see below) to calibrate/compare the Fourier expansion appearing in Eq. (8). The file shall have three columns representing: approaching flow angle, force in the $\boldsymbol{n}$ direction and force in the $\boldsymbol{t_1}$ direction. Attention, forces shall be intended as normalized by the fluid density, $\boldsymbol{f}(\alpha)/\rho$. Do not make them non-dimensional using the reference kinetic pressure.
`U:` undisturbed upstream velocity for the data stored in **`inputDataFile`**.
`area:` area of the permeable surface for the data stored in **`inputDataFile`**. This is intended as gross area of the middle surface (see Fig. 4).
`pSurfaceDict:` python dictionary collecting data needed to generate the *PVJDict*. There are three approaches available in order to characterize the permeable surface: *fromData*, *fullyDeflective* and *fromPorosity*. Depending on the selected approach, the dictionary shall be filled with different keywords/values as follows [1]:

1. *fromData* estimates the Fourier coefficients appearing in Eq. (8) fitting the forces exchanged by the permeable surface when impinged from different angles of attack, measured in wind tunnel tests or numerical simulations (usually periodic conditions). The required keywords/values are:
   `approach:` value must be set to *'fromData'*.
   `order:` maximum order of the Fourier series expansion of Eq. (8).
   `data:` data used to calibrate the Fourier coefficients.
   `ref:` choice regarding the reference system as from Sec. 4.1.

2. *fullyDeflective* estimates the Fourier coefficients appearing in Eq. (8) assuming that the flow will be completely deflected and forced to have a predefined outgoing angle. It can be used to approximate densely spaced lamellar screens, for which the lamellae tend to force the flow direction. The required keywords/values are:
   `approach:` value must be set to *fullyDeflective*.
   `theta:` deflection angle with respect to the normal, see Fig. 4.

3. *fromPorosity* estimates the Fourier coefficients appearing in Eq. (8) assuming that only forces along $\boldsymbol{n}$ are produced and calculating the pressure drop according to [2]. It can be used to approximate perforated plates. The required keywords/values are:

approach: value must be set to *fromPorosity*.
beta: permeable surface porosity (ranging from 0 to 1).
ref: choice regarding the reference system as from Sec. 4.1.

NB 5 : For the *fromData* approach, forces imported from the *inputDataFile* should be normalized by the fluid density, ρ, i.e. $A\boldsymbol{f}(\alpha)/\rho$, being A the gross area of the middle surface. Notice that these are forces per unit density not normalized by the reference area (such last normalization is performed automatically).
NB 6 : When using *fromData*, the value of `pvj_gamma` is forced to be 1, being the parameter actually redundant.
NB 7 : The folder "Data" collects examples of aerodynamic forces at different attack angles measured in numerical simulations. In particular, *lamellae_theta_45.txt* collects results (two-dimensional URANS) for a lamellar permeable surface oriented at $\theta = 45°$, while *perforated_beta_0p5.txt* collects results for a two-dimensional perforated surface with porosity $\beta = 0.5$. In both cases, $U = 10$ and area $= 0.6$.

4.4 Tips and tricks

Here are some useful tips and tricks:

1. Porous surfaces characterized by very low porosity lead to strong pressure-jumps which might render the simulation unstable. Usually it is difficult to set up stable computations when porosity is lower than 10%-15%;

2. The fact that **`t1d`** is provided but the actual $\boldsymbol{t_1}$ is calculated ensuring that it lies on the tangent plane, often allows to provide a unique **`t1d`** even when $\boldsymbol{t_1}$ varies from point to point because the permeable surface is curved;

3. It is possible to have more than one *PVJ* patches pairs and use a single *PVJDict*. This is useful in case the same porous surface type is present in more than one instance or it is curved.

5 Example

The provided example considers the computational domain reported in Fig. 6. The upper and lower patches are of periodic type while *PVJ* is applied at a cyclic patch which splits vertically the domain in two parts, so representing the presence of a permeable surface which extend over the whole domain cross-section. We assume such permeable surface to be a closely spaced lamellar screen, so we decide to model it as a fully deflective permeable surface. The lamellae are assume to be oriented so that $\theta = 45°$.

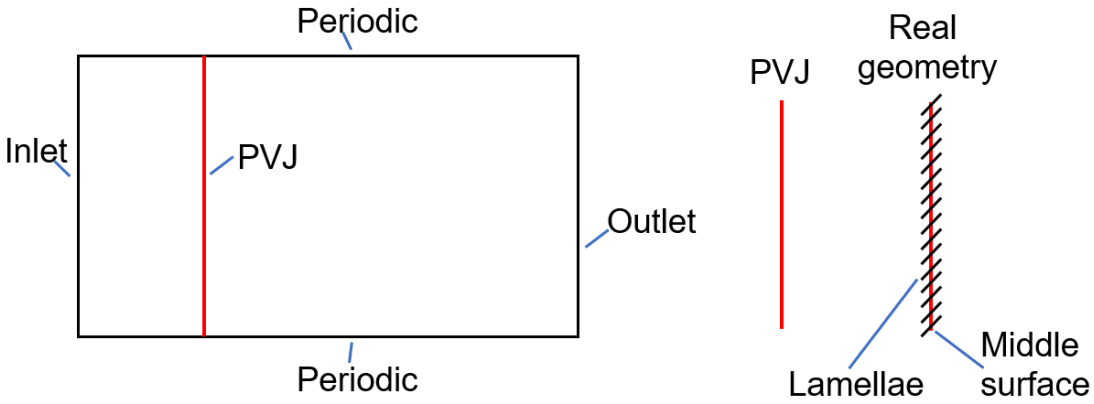


Figure 6: The computational domain and the lamellar screen with $\theta = 45°$.

A comparison between the aerodynamic forces predicted by the fully deflective permeable surface and the results of a URANS simulation, in which numerous lamellae have been explicitly modelled, is reported in Fig. 8.

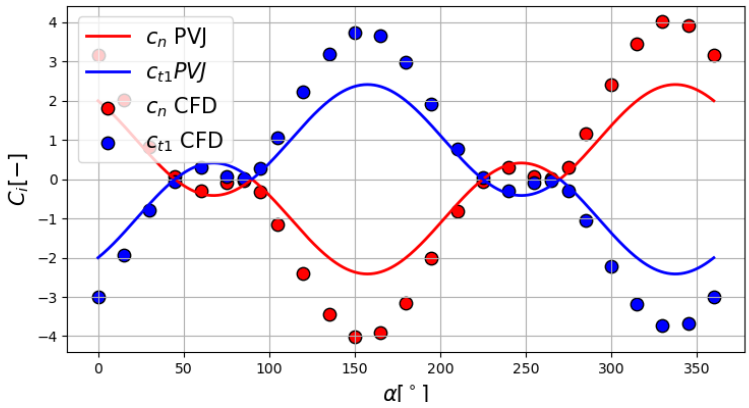


Figure 7: Aerodynamic forces acting on a lamellar screen with $\theta = 45°$.

As it can be seen, accuracy is not high, due to the fact that the actual deflection angle is not aligned with the lamellae, due to flow detachments (see [1]). However, trends are correctly captured and the approach often suffices in order to have a rough representation of the permeable surface presence. If aerodynamic forces at each angle of attack are evaluated by means of wind tunnel tests or numerical simulations, it is always possible to calibrate the PVJ form data.

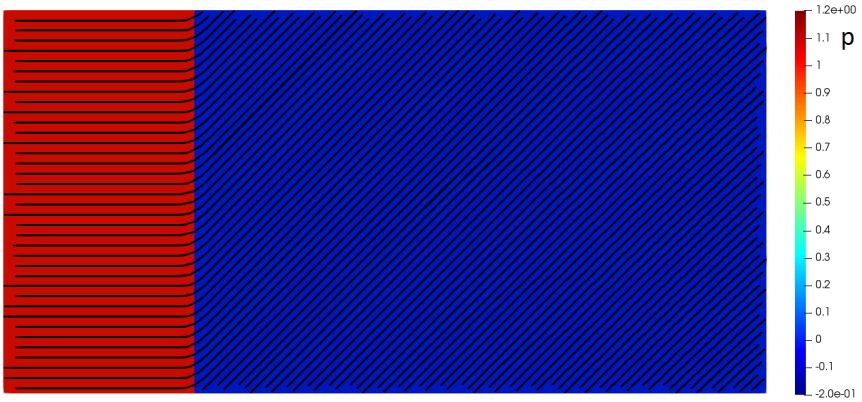


Figure 8: Flow field obtained for the lamellar screen with $\theta = 45°$: streamlines and pressure contour (cell values).

As it can be seen, in correspondence of the PVJ patch the flow is correctly deflected at 45° and an overpressure upstream of it is produced, as expected.

6 Version history

v1.0 - February 01, 2023 - First release.

References

[1] Mao Xu, Luca Patruno, and Stefano de Miranda. A pressure-velocity jump approach for the cfd modelling of permeable surfaces. *Journal of Wind Engineering and Industrial Aerodynamics*, 233:105317, 2023.

[2] Mao Xu, Luca Patruno, Yuan-Lung Lo, and Stefano de Miranda. On the use of the pressure jump approach for the simulation of separated external flows around porous structures: A forward facing step. *Journal of Wind Engineering and Industrial Aerodynamics*, 207:104377, 2020.