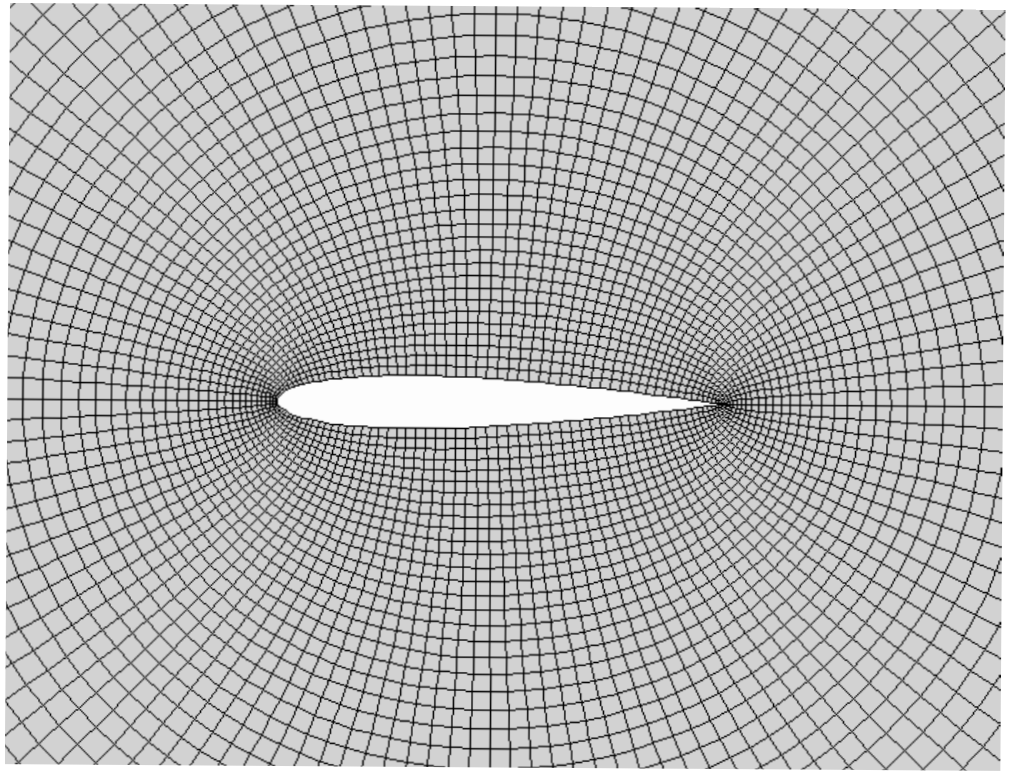
POLYTECHNIQUE MONTRÉAL

MEC6602E : Transonic Aerodynamics

Development of a 2D structured Euler solver

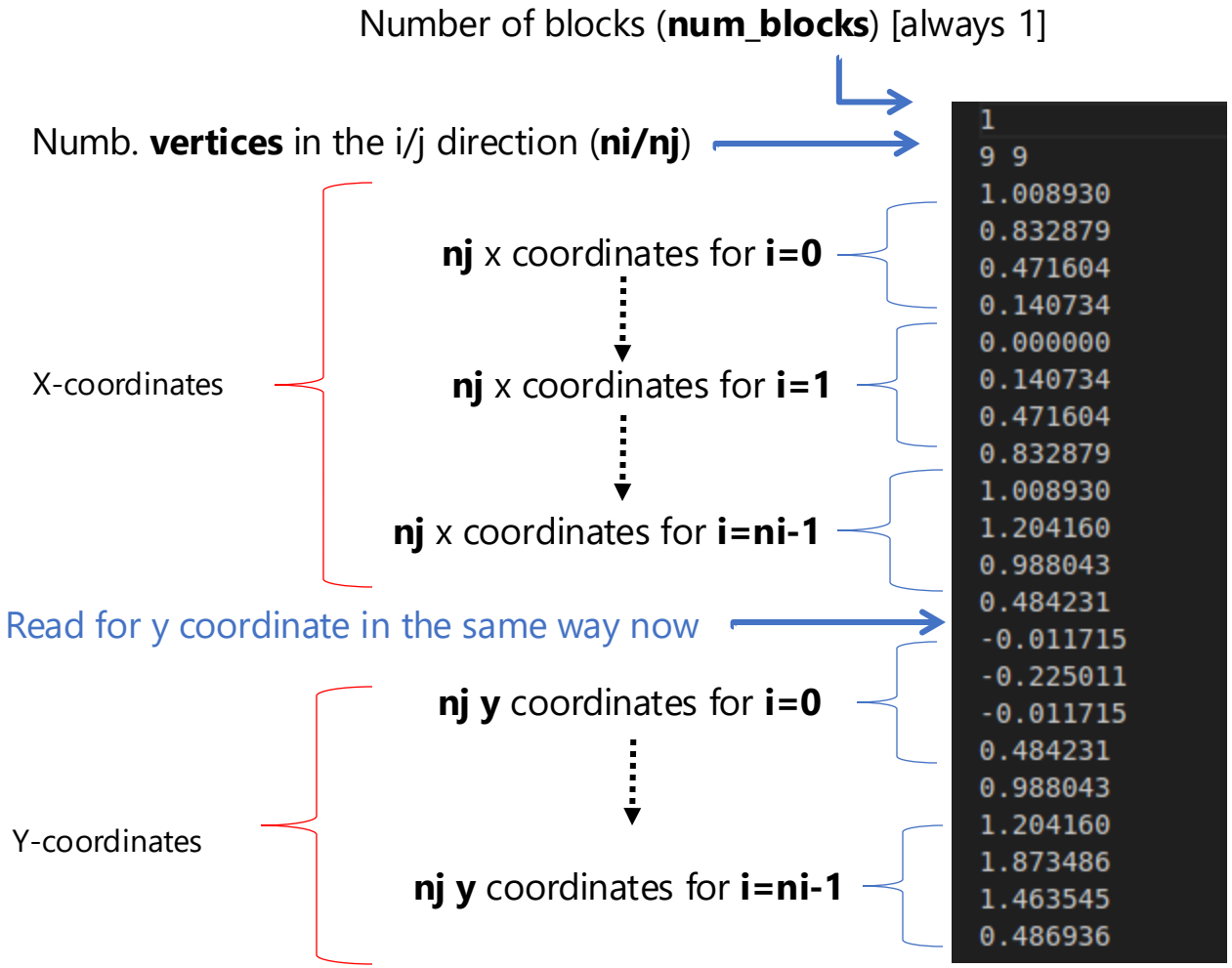
Developer guide



Latimier Corentin – Defer Tristan – Paramvir Lobana

Mesh.c : reading grid and compute grid metrics

- Grids
  - Located in folder ‘’NACA0012grids”
  - 2D structured grid (i,j)
  - Format plot3D (see example)



Example of a plot3D file

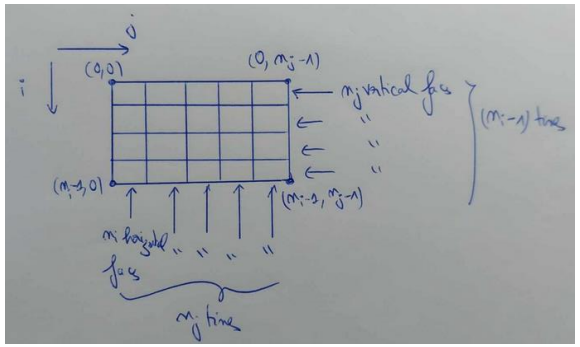
Mesh.c : reading grid and compute grid metrics

- How to store the grid information efficiently ?

  ↳ Objective of the data structure **MESH (defined in mesh.h)**

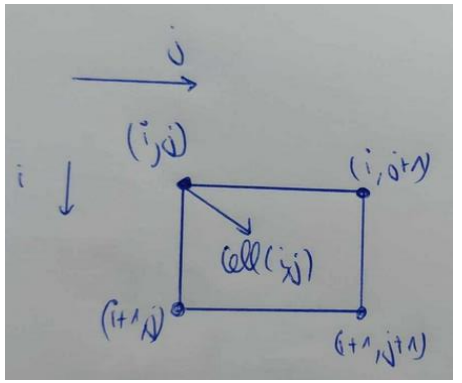
```
typedef struct{
    int ni;
    int nj;
    int nVFaces;
    int nHFaces;
    int num_blocks;
    VerticeCoordinate** coordinates;
    double** CellVolume;
    Face** VerticalFace;
    Face** HorizontalFace;
    char* filepath;
}MESH;
```

number of vertical/horizontal faces : **nV = nj(ni-1); nH = ni(nj-1)**



Matrix (ni x nj) whose entry (i,j) is (x,y) coordinate of vertice (i,j)

Matrix ([ni-1] x [nj-1]) whose entry (i,j) is the volume of cell (i,j)



Matrices of size either **([ni-1] x [nj])** (*vertical faces*) or **([ni] x [nj-1])** (horizontal faces) whose entry (i,j) has the information **(normal and face length)** of the face going from **(i,j) to (i+1,j)** (V. faces) or **(i,j) to (i,j+1)** (H. faces)

Mesh.c : reading grid and compute grid metrics

- Other data structures

```
typedef struct{
    float x;
    float y;
}VerticeCoordinate;


typedef struct {
    float nx;
    float ny;
} Normal2D;

typedef struct
{
    Normal2D normal;
    float ds;
}Face;
```

VerticeCoordinate : contains x,y coordinate of a given vertice *('coordinates' attribute of MESH structure is actually a matrix of objects VerticeCoordinate)*

Normal2D : contains x,y component of the normal for a given face

Face: contains face length and normal of a given face *(VerticalFace and HorizontalFace attributes of MESH structure are actually matrices of objects Face)*

Mesh.c : reading grid and compute grid metrics

- Functions

MESH LoadMESH(char* filepath)* → path to the grid to read

→ Returns a pointer to a MESH object **completely initialized**

void ReadHeaderPlot3D(MESH mesh) : read **num_blocks, ni, nj** and initialize their values*

→ Pointer to the mesh structure to modify it into the function directly

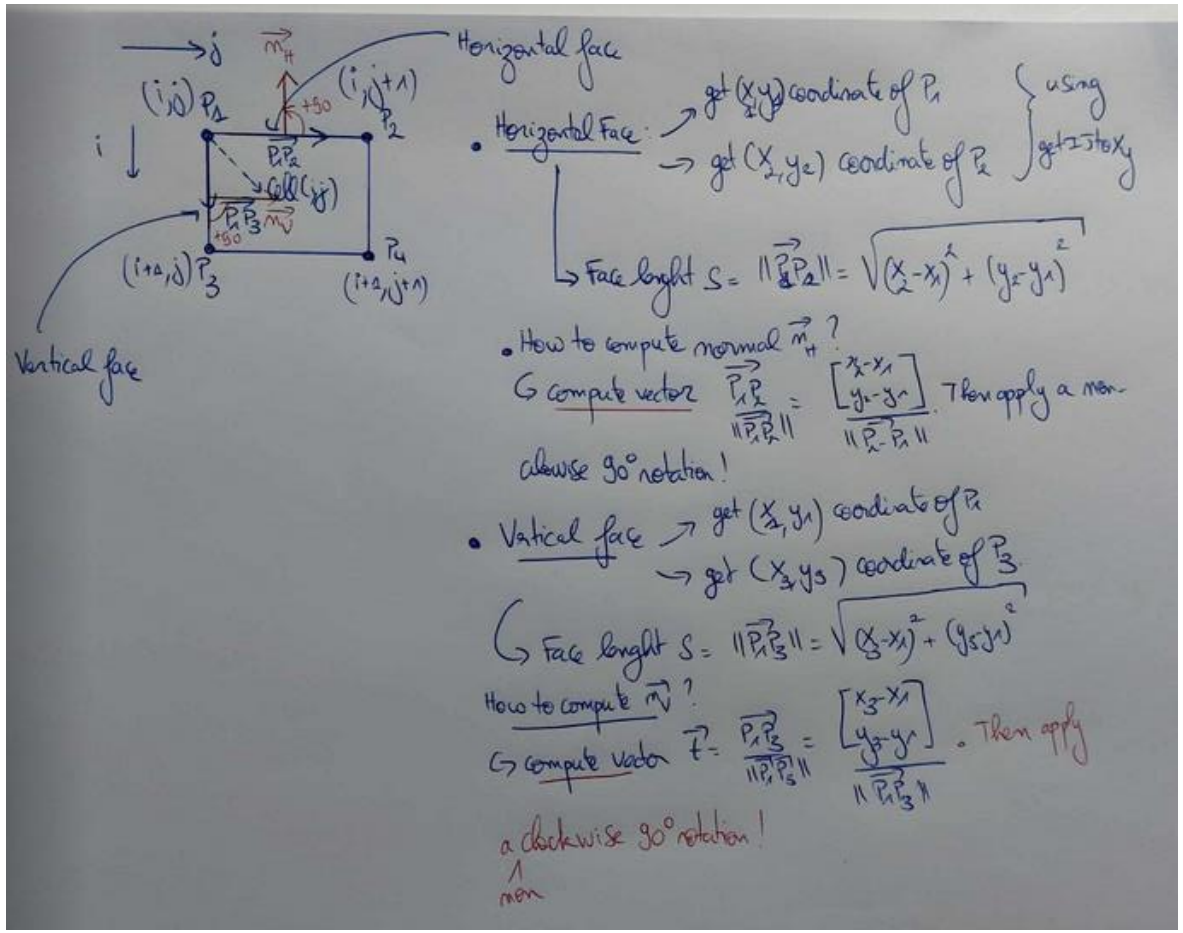
void ReadCoordinates(MESH mesh) : read x,y coordinates (described slide 2) and fill the '**coordinates**' attribute of mesh.*

→ Pointer to the mesh structure to modify it into the function directly

void InitializeFaces(MESH mesh) : compute normals and face lengths for each faces in the mesh : two loops are implemented (first for Vertical Faces, then Horizontal Faces).*

Mesh.c : reading grid and compute grid metrics

- Function '*initializeFaces*' : *how to compute face lengths and face normals ?*

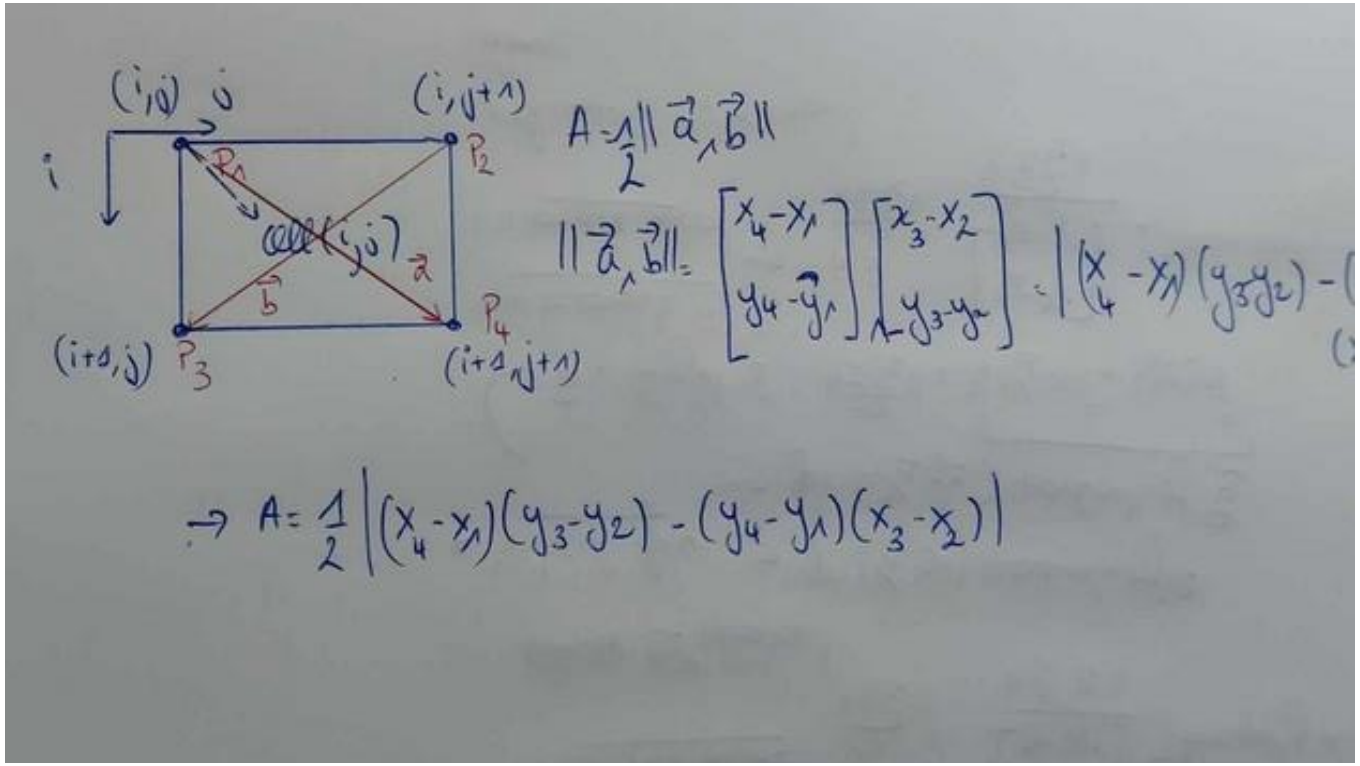


Remarks :

- ***Normal to vertical faces goes from j to j+1***
- ***Normal to horizontal faces goes from i to i-1***

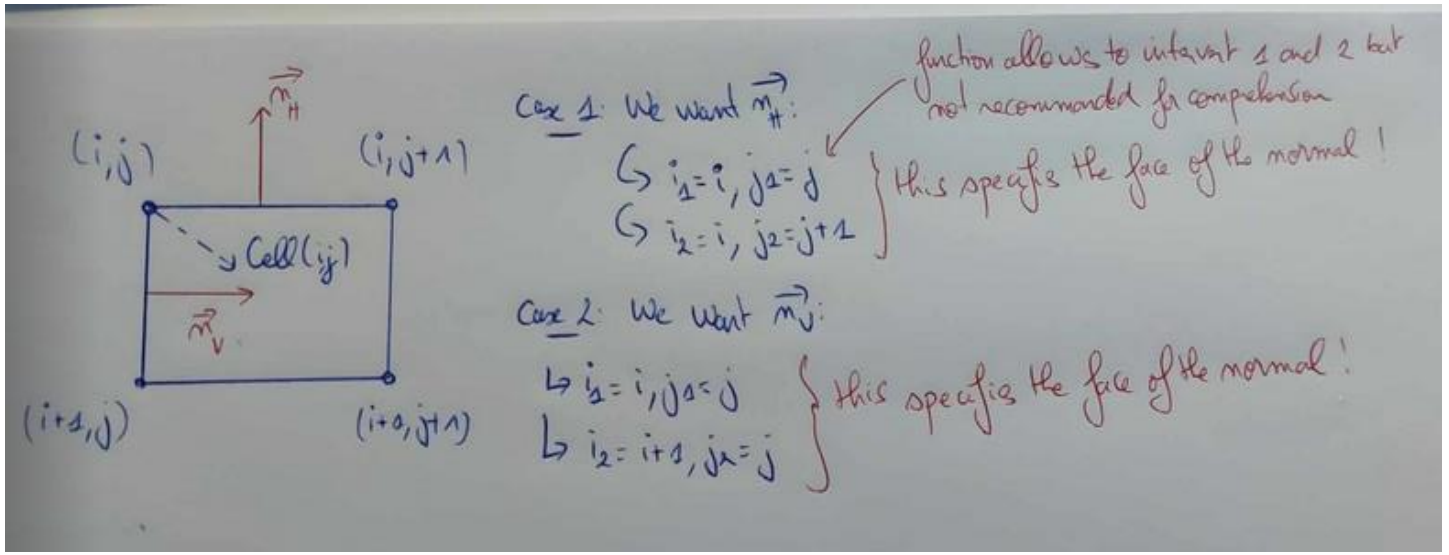
Mesh.c : reading grid and compute grid metrics

- Functions
  - *void ComputeCellVolumes(MESH* mesh) : compute area of each cell and initialize the CellVolume attribute of mesh object*



Mesh.c : reading grid and compute grid metrics

- Functions
  - *Normal2D getNormal(int i1, int i2, int j1, int j2, MESH* mesh) : for a given face, returns the normal*
    - *How to use it ? :*



  - *Float getDS (int i1, int i2, int j1, int j2, MESH* mesh) : for a given face, returns the face length*
    - *How to use it ? : similar to getNormal*

ConservedVariables.c : storing conserved variables

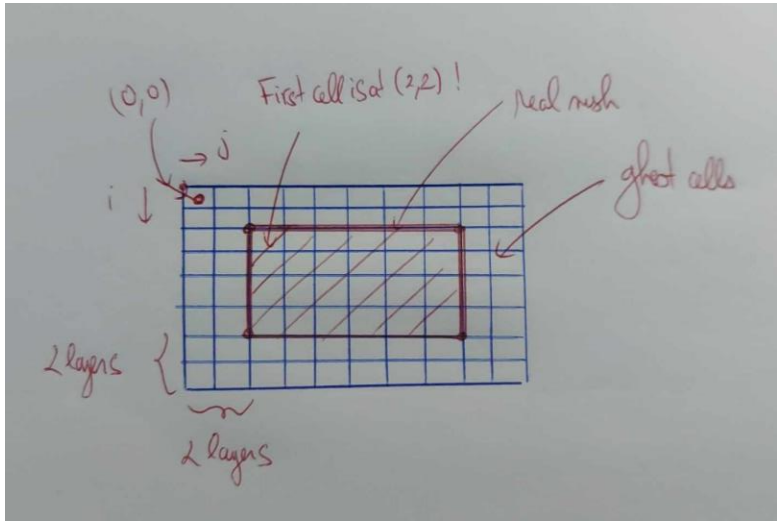
```
typedef struct
{
    int n_ghost_layers;
    int nCells_i;
    int nCells_j;
    float** rho;
    float** rhou;
    float** rhov;
    float ** rhoE;
    float ** p;
}ConservedVariables;
```

Number of ghost layers

*Number of cells in i direction : ni-1+2*n_ghost_layers*

*Number of cells in j direction : nj-1+2*n_ghost_layers*

Matrices of size (n_cells_i x n_cells_j) that contains conserved variables



BoundaryCondition.c

- How to store boundary conditions informations ?

  ↳ Objective of the data structure **BoundaryCondition** **(defined in BoundaryCondition.h)**

```c
typedef struct {
    BoundaryConditionType type;
    BoundaryDirection direction;
    int index_i;
    int index_j;
    int start_index;
    int end_index;
} BoundaryCondition;
```

3 possibilities : WALL, FARFIELD, CONNECT

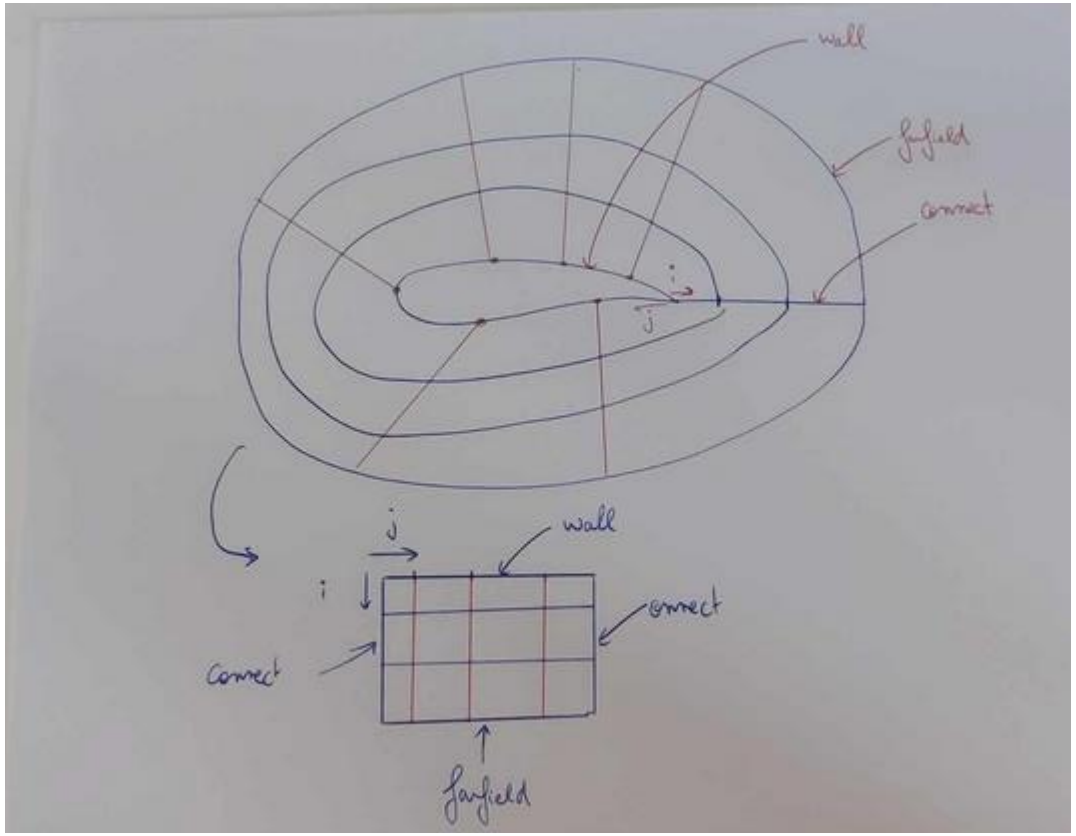
2 possibilities : I_DIRECTION / J_DIRECTION : gives the information is the condition is at constant i or constant j index (cell index)

Must be used only for WALL and FARFIELD (set –1 for both if CONNECT) ***give the constant index*** *(if I_DIRECTION, set index_j to –1, if J_DIRECTION, set index_i to –1)*

Must be used only for CONNECT (set –1 for both if not) ***give the two indexes that are connected (cell indexes)***

BoundaryCondition.c

- Boundary conditions for the NACA 0012 problem
  - **Wall : cells at i=0**
  - **Farfield : cells at i=ni-2 (we start at i=0)**
  - **Connect : cells at j=0 and j=nj-2**




```
// 3. Create boundary conditions
BoundaryCondition* bc_wall = createBoundaryCondition(WALL, I_DIRECTION, 0, -1, -1, -1);
BoundaryCondition* bc_farfield = createBoundaryCondition(FARFIELD, I_DIRECTION, MESH->ni-2, -1, -1, -1);
BoundaryCondition* bc_connect = createBoundaryCondition(CONNECT, J_DIRECTION, -1, -1, 0, MESH->nj - 2);
```