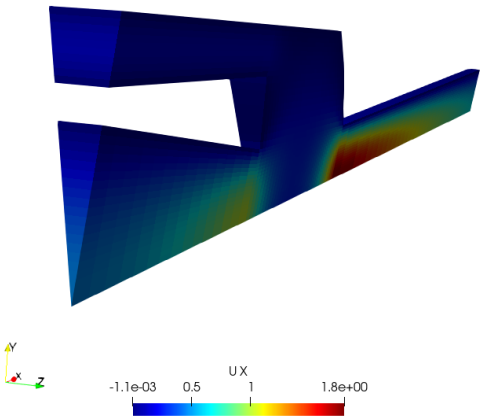
Informe N°2:
Estudio de caso axisimétrico



Resumen:

Preparación de un caso con axisimetría utilizando la malla generada con *blockMesh* en el informe anterior. Comparación de resultados utilizando ambas mallas.

Autor: Guillermo Rolle
Supervisor: Dr. Ing. César Pairetti

Diciembre 2019

Contents

1 Introducción

En este trabajo se mostrará cómo modificar una malla creada con *blockMesh* para realizar un estudio de un problema axisimétrico.

Modificaremos la malla creada en el Informe01 para que represente tubos de sección circular. No es necesario crear toda la geometría en 3D porque existen herramientas de OpenFOAM para aprovechar la simetría del problema. Solamente habrá 1 celda en la dirección del eje Z, pero esta vez la sección tendrá forma de "porción de pizza" para representar una sección cilíndrica.

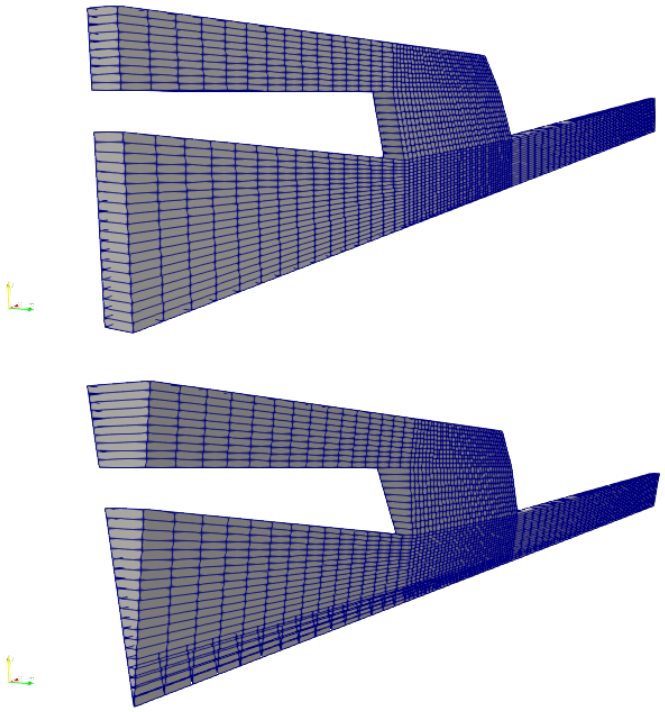


Figure 1: Comparación de la geometría. Original (arriba). Modificado (abajo)

1.1 Información preliminar del problema

Utilizaremos exactamente los mismos parámetros que el caso anterior para independizar mejor los cambios debido a la simetría de la malla. Las presiones a las entradas y salida son las mismas y también lo son las tolerancias de los métodos numéricos. El caso final preparado correspondiente a este informe se encuentra en el repositorio.

2 Preparación del caso

Comenzaremos generando una copia del caso 01 y eliminamos el directorio con toda la información correspondiente a la malla en `./constant/polyMesh`

OpenFOAM tips: Recordar que debe estar cargado el entorno OpenFOAM en la terminal

```
$ cd casos_cfd
$ mkdir 02
$ cp -r ./01 ./02
$ cd 02
$ rm -r constant/polyMesh
```

Para realizar la simetría, es necesario que las caras "frontAndBack" definidas en el diccionario blockMeshDict sean 2 diferentes.

Es decir, modificar esto:

```
empty frontAndBack
(
(0 1 12 13)
(1 2 5 12)
(2 3 4 5)
(12 5 6 11)
(11 6 7 8)
(10 11 8 9)

(14 27 26 15)
(15 26 19 16)
(16 19 18 17)
(26 25 20 19)
(25 22 21 20)
(24 23 22 25)
)
```

Por esto

```
empty front
(
(0 1 12 13)
(1 2 5 12)
(2 3 4 5)
(12 5 6 11)
(11 6 7 8)
(10 11 8 9)
)
empty back
(
(14 27 26 15)
(15 26 19 16)
(16 19 18 17)
(26 25 20 19)
(25 22 21 20)
(24 23 22 25)
)
```

2.1 Mallado Inicial

Crearemos la malla con *blockMesh*. El resultado será la misma malla que el caso anterior pero si lo observamos en *ParaView*, las caras "front" y "back" son ahora 2 componentes distintas.

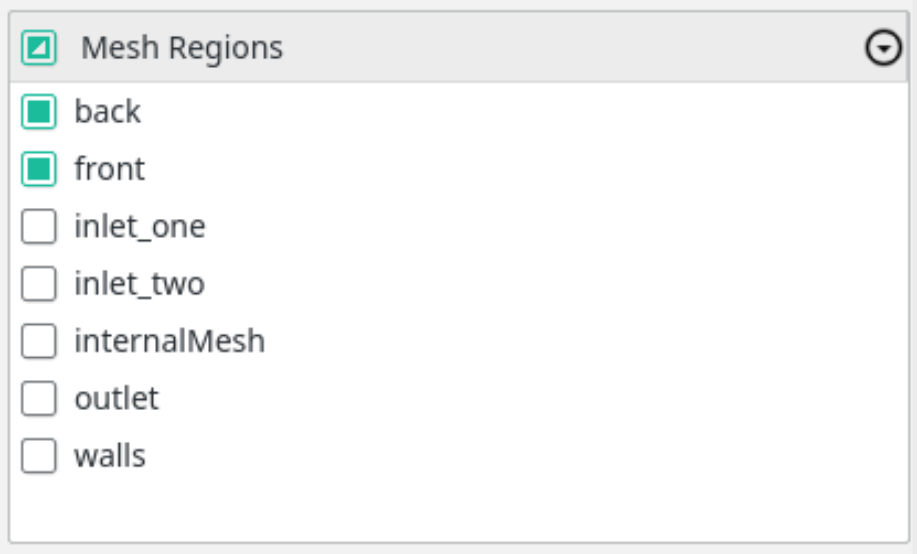


Figure 2: Regiones de malla en ParaView

Para crear la simetría utilizaremos la función extrudeMesh. Necesitamos crear un diccionario en la carpeta *system* llamado *extrudeMeshDict*. Podemos copiar uno de los casos tutorial de *OpenFOAM* o mejor podemos descargar uno directamente del repositorio en cuestión: extrudeMeshDict.

En este diccionario se define las caras de "cuña" (wedge), el eje de simetría y el ángulo de la extrusión.

```
constructFrom patch;
sourceCase "$FOAM_CASE";
sourcePatches (front);

// If construct from patch: patch to use for back (can be same as
    sourcePatch)
exposedPatchName back;
....
//- Linear extrusion in point-normal direction
extrudeModel      wedge;

point (0 0 0);
// punto de paso del eje
axis (1 0 0);
// direccion del eje de simetria
```

```
angle 15;
// angulo de la seccion circular
....
```

Ahora podemos ejecutar la utilidad *extrudeMesh*

```
$ extrudeMesh
```

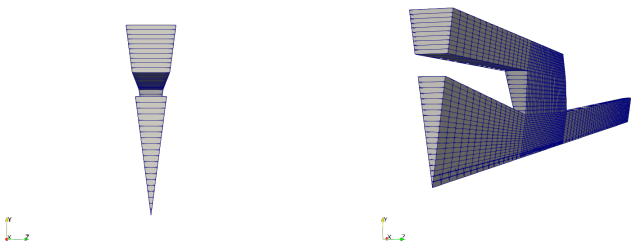


Figure 3: Sección con forma de 'porción de pizza' en ParaView

Sería interesante observar el resultado de *checkMesh* para encontrar errores en la malla. Podemos ver que hay errores debido a que, por ejemplo, hay ciertas caras que resultaron ser "aplastadas" (las del eje central) en la extrusión en cuña. Estas caras tienen área nula, lo que es problemático a la hora de resolver las simulaciones.

Linux tips: Es posible escribir un script de comandos de Linux y ejecutarlos con `./script.sh`. Ver *comandos.sh*

2.2 Reparacion de la malla

Podemos utilizar la funcionalidad collapseEdges para reparar la malla. Esta utilidad colapsa aristas pequeñas y combina aristas que estén en línea.

```
$ collapseEdges -overwrite
```

Una vez más, comprobamos con *checkMesh* para verificar que no haya inconvenientes con la malla:

```
$ checkMesh
...
Checking geometry...
Overall domain bounding box (0 0 -0.00104421) (0.1 0.00793156 0.00104421)
Mesh has 2 geometric (non-empty/wedge) directions (1 1 0)
Mesh has 3 solution (non-empty) directions (1 1 1)
Wedge front with angle 7.50001 degrees
Wedge back with angle 7.50001 degrees
All edges aligned with or perpendicular to non-empty directions.
Boundary openness (-5.20681e-21 -2.03166e-15 2.13322e-15) OK.
Max cell openness = 2.62994e-16 OK.
Max aspect ratio = 22.9592 OK.
```

```
Minimum face area = 2.91172e-09. Maximum face area = 2.2737e-06. Face
    area magnitudes OK.
Min volume = 1.24767e-12. Max volume = 5.46506e-10. Total volume =
    3.41533e-07. Cell volumes OK.
Mesh non-orthogonality Max: 51.1811 average: 16.3359
Non-orthogonality check OK.
Face pyramids OK.
Max skewness = 0.930108 OK.
Coupled point location match (average 0) OK.

Mesh OK.

End
```

3 Preparación de la simulación

Vamos a utilizar las mismas condiciones iniciales y mismos diccionarios que en el otro caso. Sin embargo, debemos actualizar los archivos `0/U` y `0/p` con los parches *front* y *back* como las definimos anteriormente.

También, debemos modificar el tipo de los parches de *empty* a *wedge*. Ver más información sobre boundaries

Las modificaciones en los archivos son (en ambos):

```
    front
{
type           wedge;
}
back
{
type           wedge;
}
}
```

Ver los archivos completos: 0/U; 0/p

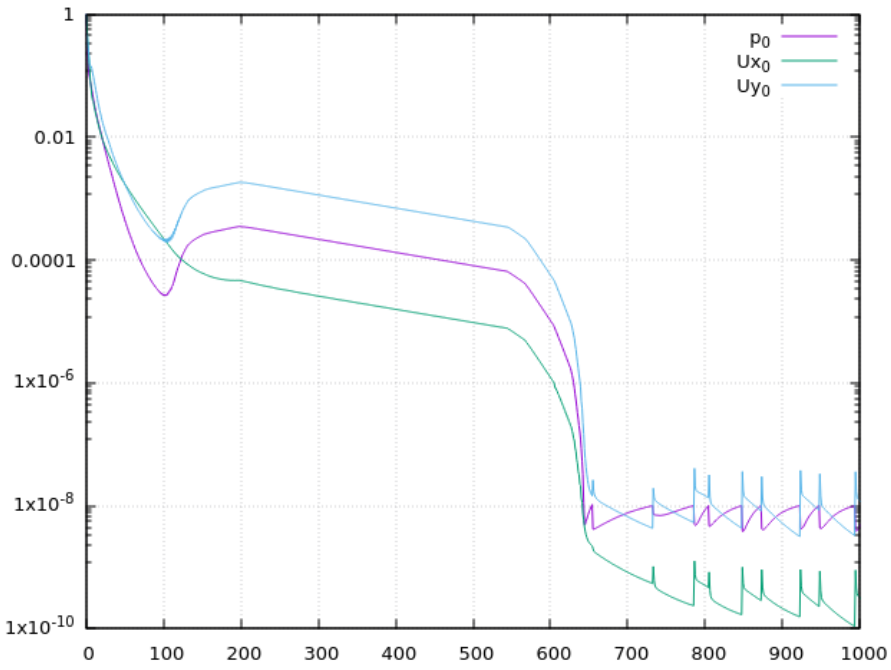
Los otros diccionarios en *system*, como *fvSolution* y *fvSchemes*, los dejamos entonces sin modificar como en la simulación anterior.

4 Simulación: *simpleFoam*

En este caso, la simulación no converge. Podemos ver que calcula hasta la última iteración definida en el *controlDict*. Volvemos a ejecutar la simulación y procesamos los archivos log para verificar los residuales. Luego los graficamos entrando a la carpeta `./logs` con *gnuplot*.

```
$ simpleFoam | tee log.simpleFoam
$ foamLog log.simpleFoam
$ gnuplot
```

```
> load 'graficar_res.gp'
```



gnuplot tips: Es posible escribir un script de comandos de gnuplot y ejecutarlos con `gnuplot <script.gp>` o desde gnuplot con `'load <script.gp>'`

Figure 4: Residuales no convergentes

Esto es así porque *simpleFoam* está intentando resolver para Uz también por la simetría y ese valor no converge. Para solucionarlo vamos a evitar controlar por Uz por fines prácticos. Para eso, debemos modificar en el archivo `system/fvSolution`:

```
Esta linea: U               1e-5;
Por esta:  "(Ux|Uy)"        1e-5;
```

Ver archivo completo: system/fvSolution

Podemos ejecutar la simulación nuevamente con los mismos comandos anteriores y vemos los residuales nuevamente. Esta vez, con Uz también:

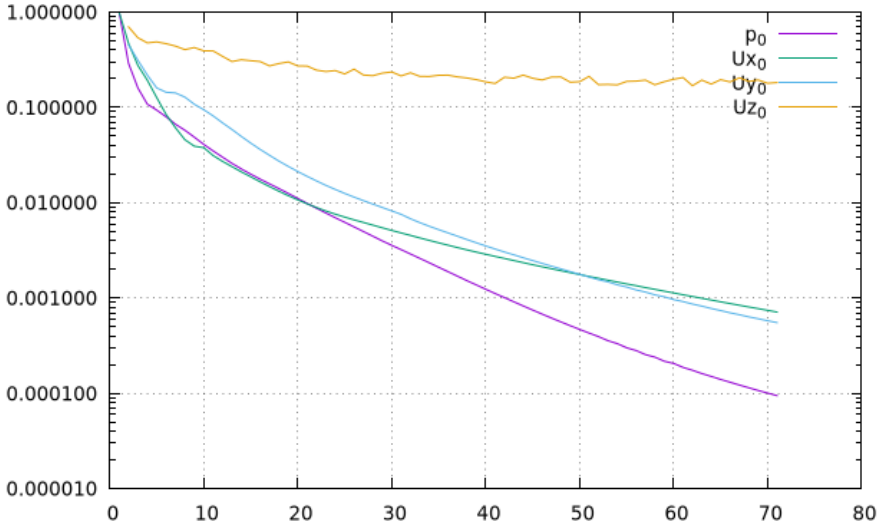


Figure 5: Residuales convergentes

Nótese que no se controla por Uz.

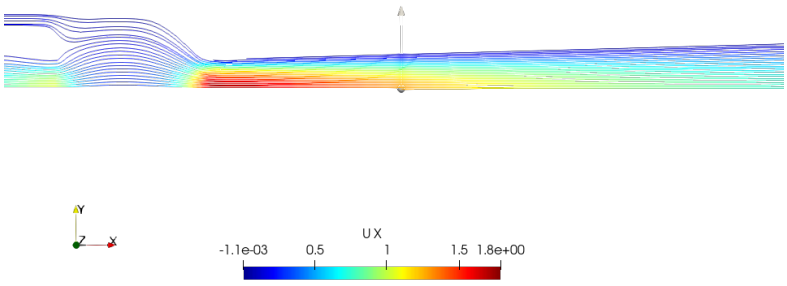


Figure 6: Velocidad X

5 Preparación del caso RTD

Como base para la simulación del transporte escalar copiaremos otro de los tutoriales en una carpeta nueva. En la carpeta principal del caso:

```
$ mkdir RTD
$ cp -r $FOAM_TUTORIALS/basic/scalarTransportFoam/pitzDaily ./RTD
$ cd RTD
```

Tendremos la siguiente estructura de archivos:

```
RTD
|-- 0
|   |-- T
|   `-- U
|-- constant
|   `-- transportProperties
`-- system
    |-- blockMeshDict
    |-- controlDict
    |-- fvSchemes
    `-- fvSolution

3 directories, 7 files
```

Copiamos la malla del caso anterior en el caso RTD.

```
$ cp -r ../constant/polyMesh ./constant
```

Como antes, podemos comprobar que la malla esté creada correctamente ejecutando

```
$ paraFoam
```

5.1 Condiciones iniciales

Para el campo de velocidad usaremos el obtenido en la simulación anterior. En este caso, la última iteración es la 110. Por lo tanto

```
$ cp ../110/U ./0
```

La concentración de agroquímico será representada en este caso por la variable T. Un valor de 1 indicaría que el fluido es agroquímico puro y un valor de 0 indicaría que el fluido en ese lugar es únicamente agua.

Entonces, para el *inlet_ one* pondremos una condición tipo *fixedValue* con valor 0 y para *inlet_ two* un valor de 1. Para el *outlet* pondremos condición *zeroGradient*.

Por ejemplo, para *inlet_ one*:

```
inlet_one
{
       type   fixedValue;
       value  uniform 0;
}
```

Ver el archivo completo: *RTD/0/T*

5.2 Propiedades del fluido

En la carpeta *constant* debemos definir un valor para el coeficiente de difusión del agroquímico. Propondremos un valor de 0.00005 (ojo! corroborar con el repositorio el valor más actualizado):

```
...
DT                      0.00005; // Para OpenFOAM-plus (v1906)
DT   DT [0 2 -1 0 0 0 0] 0.00005; // Para OpenFOAM-dev (7)
...
```

OpenFOAM tips: En la versión *dev* es necesario definir las unidades de la difusividad mientras que en la *plus* no.

Ver el archivo completo: *RTD/constant/transportProperties*

5.3 Diccionarios en *system*

Nuevamente, estos diccionarios contienen información sobre los solvers, pasos de tiempo, métodos numéricos, etc. El más importante es *RTD/system/controlDict*. Esta vez, como el solver *scalarTransportFoam* es un solver transitorio, las variables de tiempo sí representan tiempos y no iteraciones.

```
...
endTime 2; // en segundos ambas. Chequear ultima version en repositorio
deltaT   0.001;
...
```

Los otros dos contienen información sobre los esquemas numéricos y tolerancias.

Ver archivo: *RTD/system/fvSchemes*

Ver archivo: *RTD/system/fvSolution*

6 Simulación: *scalarTransportFoam*

Ya tenemos el caso RTD preparado y podemos ejecutar la simulación con el solver *scalarTransportFoam*. Cabe destacar que este es un solver transitorio a diferencia de *simpleFoam* que es estacionario.

```
$ scalarTransportFoam | tee log
$ paraFoam
```

Podemos cargar el archivo de estado de ParaView *RTD.pvsm* para abrir las gráficas y filtros correspondientes.

Figure 7: Concentración de agroquímico (t = 2s)

7 Post proceso

Nos interesa conocer la concentración promedio en toda el área de salida del dispositivo. En la carpeta *system* del caso RTD copiaremos el archivo *patchAverage* de la carpeta *etc* de OpenFOAM. En la carpeta principal RTD ejecutar:

```
$ cp $FOAM_ETC/caseDicts/postProcessing/surfaceFieldValue/patchAverage ./
    system
$ nano system/patchAverage
```

En este archivo debemos modificar los campos *patchName* y *field names* con nuestros valores de interés. Los cambios son los siguientes:

```
...
name  outlet;
fields (T);
...
```

Ver el archivo completo: RTD/system/patchAverage

Podemos ejecutar el post-proceso con el siguiente comando.

```
$ postProcess -func 'patchAverage'
```

Esto nos va a crear una carpeta *postProcessing/patchAverage/0* con el archivo *surfaceFieldValue_ 0.dat*. Cambiamos de directorio para analizar más fácilmente estos datos:

```
$ cd postProcessing/patchAverage/0
```

Ese archivo contiene el valor de areaAverage(T) para cada instante de tiempo dispuestos en columna. Para ver el contenido del archivo en la terminal ejecutar:

```
$ cat surfaceFieldValue_0.dat
```

Este archivo podemos leerlo directamente en *gnuplot* para realizar una gráfica de la concentración en función del tiempo.

```
$ gnuplot
gnuplot> plot 'surfaceFieldValue\_0.dat' w l title "outlet"
gnuplot> set xlabel "Tiempo [s]"
gnuplot> set ylabel "Concentracion []"
gnuplot> replot
```

Observamos que la concentración en la salida converge a un valor de aproximadamente 15%.

Figure 8: Concentración en la salida en función del tiempo

Podemos también observar el perfil de concentración sobre la salida para comprobar la homogeneidad de la mezcla. Esto se ve en rojo en la figura 9.

Figure 9: Distribución de valores sobre toda la altura del outlet