Universidad Politécnica de Cartagena

industriales
etsii UPCT

Generación automática de mallado y análisis lagrangiano del flujo oscilatorio en tubos con muelles insertados

Titulación: Ingeniería Industrial
Alumno/a: Samuel Espín Tolosa
Director/a/s: Juan Pedro Solano Fernández

Cartagena, 24 de Septiembre de 2012

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Resumen

Los reactores químicos tipo OBR (oscillatory baffled reactors) son sistemas de flujo pulsátil pensados para realizar reacciones químicas lentas en flujo continuo. Estos equipos son capaces de dar un alto tiempo de residencia para un amplio rango de valores de flujo neto. La mejor aplicación de los OBR está en convertir procesos por lotes con largos tiempos de residencia en procesos continuos, en condiciones donde los reactores tubulares convencionales darían relaciones de diámetro y longitud impracticables. Los reactores helicoidales de flujo oscilatorio generan un flujo secundario con generación de vórtices que se superpone al flujo principal. Este flujo secundario generado mejora el mezclado a bajo número de Reynolds oscilatorio, que es una zona donde otros tipos de reactores no interactúan lo suficiente con el flujo como para promover un mezclado.

Índice general

Índice de figuras

Índice de codigos

Agradecimientos

Desearia agradecer a mi familia, por el apoyo que me ha dado,a Maria Angeles por toda su paciencia que ha tenido a lo largo de estos años de carrera. A mis amigos por su ayuda y especialmente a mi amigo Felipe por ayudarme con el maquetado de la memoria, y a Juan Pedro por la libertad que me ha dado para realizar este proyecto.

Samuel Espin Tolosa

Capítulo 1

Introducción

1.1. Objetivos

1.1.1. Generales

El objetivo de este proyecto es facilitar herramientas de software para el estudio del flujo oscilatorio en tubos con muelles insertados. Las herramientas están dirigidas a la generación paramétrica de mallados en este tipo concreto de geometrías; y al desarrollo de una aplicación para el análisis mediante inyección de partículas que permita futuros estudios de tiempos residencia o de fenómenos de transporte.

1.1.2. Específicos

- Definición de los conceptos geométricos en tubos con muelles insertados, definición del concepto de simetría helicoidal y caracterización de las relaciones entre las variables.
- Aplicación de simetría helicoidal al modelo de flujo en tubo con muelle insertado.
- Diseño paramétrico de geometrías helicoidales que permitan diseñar con poco esfuerzo estudios de sensibilidad de malla o de optimización de parámetros geométricos.
- Diseño de la aplicación de análisis mediante inyección de partículas que permita aprovechar las características de simetría que posee este tipo de flujo.
- Representación lagrangiana del flujo para régimen estacionario y oscilatorio.

1.2. Reactores de flujo oscilatorio.

1.2.1. Terminología

Considérese inicialmente un flujo en un conducto liso de sección transversal circular, caracterizado por la velocidad media del flujo u, el diámetro interior del conducto D , y la viscosidad cinemática del fluido ν. Se denomina **flujo oscilatorio puro** (pure oscillatory flow, POF) a la superposición de un movimiento oscilatorio sobre dicho flujo (Figura 1.1). Para caracterizar un POF se requiere, además de los parámetros anteriores, un parámetro

adicional: la velocidad de oscilación $u_0 = x_0\omega$, siendo la amplitud de oscilación x_0 y la frecuencia angular de oscilación ω. Insertando en el conducto una serie de elementos equiespaciados se obtiene un **flujo oscilatorio con orificios deflectores** (oscillatory baffled flow, OBF) (Figura 1.1). En la caracterización de un OBF intervienen, además de los parámetros anteriores, dos parámetros geométricos: el diámetro interior del elemento deflector D y la distancia entre elementos deflectores L.

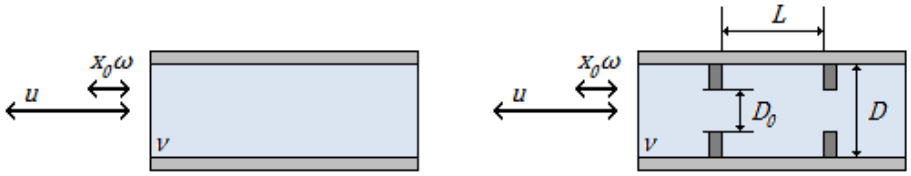

Figura 1.1: Parámetros de caracterización del flujo oscilatorio. Izquierda, flujo oscilatorio puro POF. Derecha, flujo oscilatorio con orificios deflectores OBF.

Los dispositivos de mezcla que generan en su interior un OBF se conocen como **columnas de flujo oscilatorio con orificios deflectores** (oscillatory baffled colum, OBC)(Figura 1.2)(Figura 1.3). Por lo tanto, son dos los aspectos que están siempre presentes en estos equipos de mezcla tubulares:

- Presencia de elementos insertados espaciados uniformemente.
- Existencia de un flujo oscilatorio sobre un flujo neto.

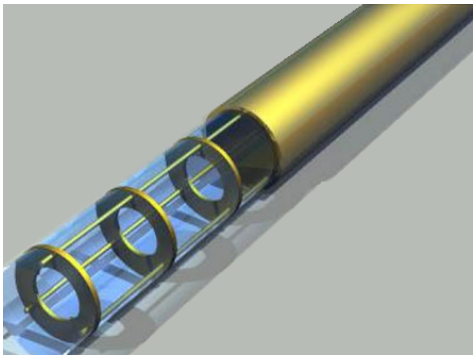

Figura 1.2: Izquierda, forma típica de una columna de flujo oscilatorio con orificios deflectores, OBC.

En la figura anterior se muestra la forma típica de un OBC, en el que los **elementos insertados** son discos con un único orificio central. Otros elementos que pueden ser también utilizados son discos con múltiples orificios, muelles helicoidales, láminas onduladas, superficies aleteadas, mallas, etc. (Figura 1.4).

Estos elementos insertados dan lugar a una aceleración local del flujo por la reducción de sección transversal de paso, que se cuantifica a través del ratio de constricción (constriction

Figura 1.3: Derecha, vista del equipo completo.

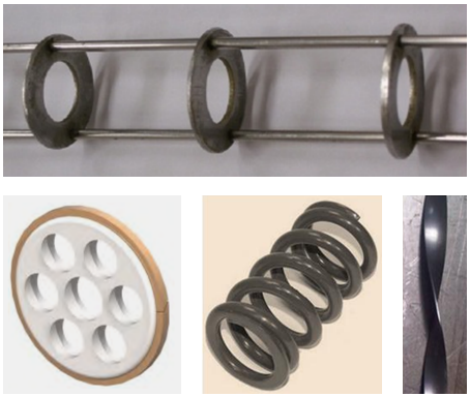

Figura 1.4: Otros elementos insertados en columnas de flujo oscilatorio. Disco con un único orificio (a), disco con múltiples orificios (b), muelle helicoidal (c), lámina ondulada (d).

ratio, CR):

Por su parte, el flujo que circula por el interior de la columna está compuesto por un **flujo neto**, caracterizado por una velocidad u_{net} constante a lo largo del tiempo, sobre el que se superpone un **flujo oscilatorio**, cuya velocidad u_{osc} sigue una ley sinusoidal con el tiempo. En la práctica, la magnitud de la componente oscilatoria es muy superior a la magnitud de la componente neta.

$$u_{osc} = 2\pi f x_0 cos(2\pi f t) \tag{1.1}$$

$$u(t) = u_{net} + u_{osc} \tag{1.2}$$

1.2.2. Ventajas y aplicaciones.

Los reactores de flujo oscilatorio OFR ofrecen interesantes ventajas respecto a los procesos tradicionales de mezclado en tanques de agitación, como son una distribución más uniforme de la fase dispersa en la fase acuosa, una mejora de la transferencia de calor y masa , una estrecha distribución del tiempo de residencia de las partículas y un preciso control de las

condiciones de mezcla . A diferencia del reactor tubular convencional, en el que se requiere una elevada velocidad superficial para obtener una buena mezcla, el OFR puede proporcionar un comportamiento de flujo pulsante en régimen de flujo laminar. Además, un OFR es ideal para la realización de reacciones largas, ya que la mezcla conseguida es independiente del gasto másico. Esto permite convertir en continuos a aquellos procesos en modo por lotes que requieren un elevado tiempo de residencia, y que obligan a diseños con alto ratio longitud-diámetro si son ejecutados con reactores convencionales.

El empleo de estos reactores de flujo oscilatorio se da, principalmente, en las industrias químicas, bioquímicas y farmacéuticas. Algunas de las aplicaciones donde actualmente se emplean OFR son la polimerización, la fermentación, la síntesis orgánica, el procesamiento o la dispersión. Quizás la aplicación más relevante sea la fabricación de polímeros mediante **polimerización en suspensión**. En este proceso se mezcla una fase acuosa, generalmente agua, con una cierta cantidad de monómero insoluble, dando lugar a la formación de gotas de monómero en la fase acuosa. El OFR permite obtener una distribución adecuada y uniforme del tamaño de las partículas de polímero a través de control óptimo de las condiciones de mezcla, especialmente de la temperatura, así como evitar la fusión de las gotas de monómero durante la reacción. Con la polimerización en suspensión se producen polímeros como el cloruro de polivinilo, el poliestireno o las resinas de intercambio de cationes y aniones, y más recientemente el metacrilato de metilo, la acrilamida y el estireno.

Una de las aplicaciones potenciales más atractivas para los reactores de flujo oscilatorio es la producción de biocarburantes. El empleo de reactores tipo tanque de agitación es una tecnología suficientemente conocida, y cuya eficiencia es ya difícilmente mejorable. Por ello se trabaja actualmente en el desarrollo de reactores intensificados: reactores estáticos, reactores con micro-canales, reactores con cavitación, reactores rotativos y reactores de flujo oscilatorio. Estos dispositivos están caracterizados por la miniaturización de los equipos tubulares, un mayor grado de conversión y una menor cantidad de catalizador, el uso de condiciones de proceso más suaves, la reducción de los costes energéticos asociados y el aumento de la capacidad de producción . En última instancia, estas nuevas tecnologías permitirán aumentar la competitividad de las instalaciones de producción de biodiesel.

Recientemente se ha demostrado la viabilidad del OFR para la producción mejorada de biodiesel a partir de semillas oleaginosas, como la jatropha curcas, y para la fermentación de bioalcoholes a partir de materias primas lignocelulósicas.

1.2.3. Mecanismo de mezcla.

El patrón de movimiento del flujo en el interior de un OFR es generado por el efecto combinado de la presencia de elementos insertados y la existencia de flujo oscilatorio superpuesto. El flujo se acelera y se desacelera según una ley sinusoidal para la velocidad en función del tiempo. Con cada aceleración del flujo se forman anillos de vorticidad aguas

abajo de los deflectores. Cuando el flujo se desacelera se forman nuevos anillos de vorticidad en el lado opuesto de los deflectores que, a su vez, desplazan a los anillos formados durante la aceleración hacia la región entre los deflectores. La interacción entre los anillos de vorticidad creados durante la aceleración y desaceleración del flujo da lugar a complejas estructuras vorticales que intensifican el proceso de mezcla. En la (Figura 1.5) se muestra esquemáticamente este mecanismo:

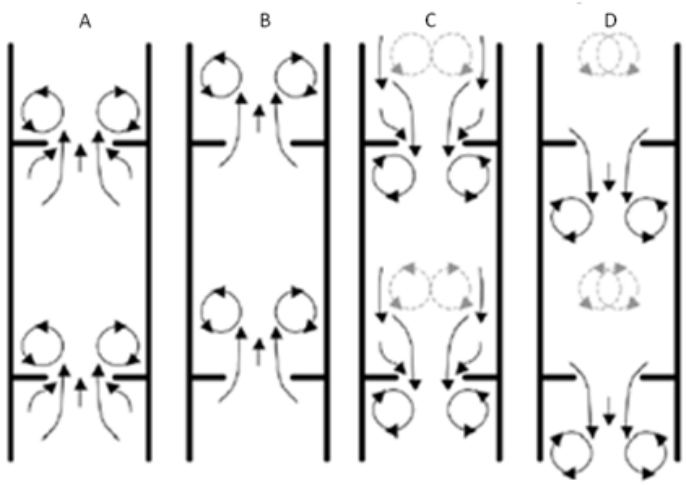

Figura 1.5: Esquema del mecanismo de mezcla en reactores de flujo oscilatorio.

Con la repetición sucesiva de ciclos de formación de anillos de vorticidad, la elevada componente de velocidad radial que se alcanza genera una mezcla uniforme en cada región entre deflectores, que se acumula a lo largo de la longitud del reactor. Este mecanismo de mezcla posee, por tanto, periodicidad espacial y periodicidad temporal, ya que las estructuras vorticales son las mismas en cada una de las regiones entre deflectores, y se repiten con cada ciclo velocidad-tiempo. En la (Figura 1.6) se muestra una visualización de este mecanismo de mezcla:

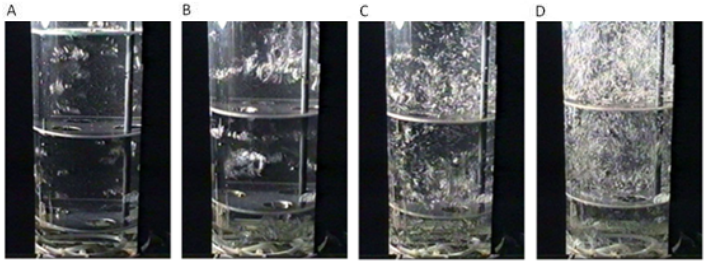

Figura 1.6: Visualización del mecanismo de mezcla. Sin oscilación (a). Comienzo de la oscilación (b). Tras un ciclo completo de oscilación (c). Tras varios ciclos completos de oscilación (d).

1.3. Definición del problema.

1.3.1. Grupos adimensionales.

El problema fluidomecánico del flujo oscilatorio en un OBR se caracteriza tradicionalmente por dos **grupos adimensionales**: el número de Reynolds oscilatorio $\Re_o$ y el número de Strouhal St El número de Reynolds oscilatorio describe la intensidad de la mezcla, mientras que el número de Strouhal mide la propagación efectiva de vórtices. La caracterización de las condiciones de operación según estos grupos adimensionales se recoge en la Tabla 1.1.

$$R_0 = \frac{Dx_0\omega}{\nu} \tag{1.3}$$

$$St = \frac{D}{4\pi x_0} \tag{1.4}$$

Condición de operación	Re	St
1	1	0.4
2	10	0.4
3	80	0.4
4	1000	0.4

Tabla 1.1: Regimenes de flujos simulados.

Capítulo 2

Estado del arte

2.1. Coordenadas helicoidales

2.1.1. Definición

Se definen las coordenadas helicoidales como un caso general de las coordenadas cilíndricas, por ello va a usarse la notación típica de estas coordenadas. En cilíndricas un punto viene representado por la terna:

$$(r, \varphi, h) \tag{2.1}$$

Igual ocurre en coordenadas helicoidales, la diferencia está en que ahora existe una relación que liga el ángulo girado φ, con la altura ascendida h. Estas ecuaciones están relacionadas con las coordenadas cartesianas (x, y, z) mediante la transformación:

$$x = rcos\left(\varphi + \frac{2\pi}{p}h\right) \tag{2.2}$$

$$y = rsen\left(\varphi + \frac{2\pi}{p}h\right) \tag{2.3}$$

$$z = h \tag{2.4}$$

El tensor métrico g_{ij} de la transformación define la métrica del espacio:

$$(g_{ij}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & r^2 & \frac{2\pi r^2}{p} \\ 0 & \frac{2\pi r^2}{p} & 1 + \left(\frac{2\pi r}{p}\right)^2 \end{bmatrix} \tag{2.5}$$

La derivada direccional $\dot{\phi_j}$ de una magnitud ϕ respecto de una coordenada generalizada

x_j se define como[1]:

$$\dot{\phi_j} := \frac{\partial \phi}{\partial x_j} = h_j^i \frac{\partial \phi}{\partial x^i} \widehat{e}_j \tag{2.6}$$

donde h_{ij} es el factor de escala de la transformación de coordenadas, y se define como:

$$h_{ij} = \frac{1}{\sqrt{g_{ij}}} \tag{2.7}$$

El operador gradiente se define como:

$$\overrightarrow{grad}\,(\circ) = h^{ij} \frac{\partial\,(\circ)}{\partial x^i} \widehat{e}_j \tag{2.8}$$

En coordenadas helicoidales quedaría como:

$$\overrightarrow{grad} = \frac{\partial}{\partial r}\widehat{e}_r + \frac{1}{r}\left(\frac{\partial}{\partial \varphi} + \sqrt{\frac{p}{2\pi}}\frac{\partial}{\partial h}\right)\widehat{e}_\varphi + \left(\frac{1}{r}\sqrt{\frac{p}{2\pi}}\frac{\partial}{\partial \varphi} + \frac{p}{\sqrt{p^2 + 4\pi^2 r^2}}\frac{\partial}{\partial h}\right)\widehat{e}_h \tag{2.9}$$

2.2. Simetría helicoidal

Cuando se dice que existe simetría helicodial se está diciendo que cualquier variable ϕ , no cámbia a lo largo de la coordenada $\widehat{e}_h$. Esto es equivalente a decir que en la solución de la variable $\phi = \phi(x_1,\ x_2,\ x_3,\ t)$, que en principio dependería de tres coordenadas espaciales, una de ellas puede ponerse en función de las otras dos porque de hecho no es independiente.

$$h = f\,(r,\ \varphi) \tag{2.10}$$

La función f , tiene el siguiente aspecto:

$$f = \frac{p}{2\pi}\varphi + h_0 \tag{2.11}$$

Esto lleva a imponer la condición de simetría siguiente:

$$\dot{\phi_h} = 0 \tag{2.12}$$

[1]Se ha utilizado el convenio de sumación de Einstein . Dos subíndices repetidos expanden la suma sobre la dimensión de ese subíndice. Si aparecen dos pares de subíndices repetidos, la expansión que tiene prioridad es la del índice superíndice, sobre el índice subíndice:

Desarrollando esta expresión, se llega a:

$$\frac{1}{r}\sqrt{\frac{p}{2\pi}}\frac{\partial\phi}{\partial\varphi}+\frac{p}{\sqrt{p^2+4\pi^2r^2}}\frac{\partial\phi}{\partial h}=0 \tag{2.13}$$

que simplificada queda como:

$$\frac{\partial\phi}{\partial h}=-\Lambda\frac{\partial\phi}{\partial\varphi} \tag{2.14}$$

$$\Lambda^2=\frac{c^2+r^2}{c\,r^2} \tag{2.15}$$

$$c=\frac{2\pi}{p} \tag{2.16}$$

Esta ecuación es una relación entre dos de las variables espaciales, de tal forma que podría sustituirse en un sistema de ecuaciones diferenciales para reducir el problema de $3D$ a un problema $2D$ en un sección transversal a la dirección axial.

2.3. Operadores diferenciales bajo condición de simetría

A continuación va a aplicarse la condición de simetría en los principales operadores diferenciales, las ecuaciones que resulten de la simplificación podrían utilizarse para ser sustituidas en cualquier ecuación diferencial escrita en función de estos operadores.

2.3.1. Gradiente

El gradiente de una magintud ϕ, bajo la condición de simetría helicoidal quedaría como:

$$\overrightarrow{grad}\,(\phi)=\frac{\partial\phi}{\partial r}\widehat{e}_r+\frac{1}{r}\left(1-\sqrt{\frac{p}{2\pi}}\Lambda\right)\frac{\partial\phi}{\partial\varphi}\widehat{e}_\varphi \tag{2.17}$$

2.3.2. Divergencia

El operador divergencia se define en coordenadas generalizadas como[2]:

$$div\,(\mathbf{v})=\frac{1}{\sqrt{|g|}}\frac{\partial}{\partial x^i}\left(\sqrt{|g|}v^i\right) \tag{2.18}$$

Aplicado a coordenadas helicoidales y teniendo en cuenta que el determinante jacobiano

[2]La derivada $\frac{\partial}{\partial x^i}\left(\sqrt{|g|}v^i\right)$, está expresada en coordenadas locales y se calcula como la derivada direccional según la coordenada x_i de la magnitud $\phi=\sqrt{|g|}v_i$.

de la transformación es:

$$\sqrt{|g|} = r \tag{2.19}$$

se obtiene:

$$div\,(\mathbf{v}) = \frac{1}{r}\frac{\partial}{\partial r}\,(r v_r) + \frac{1}{r}\frac{\partial}{\partial \varphi}\,(v_\varphi) + \sqrt{\frac{p}{2\pi r^2}}\frac{\partial}{\partial h}\,(v_\varphi) + \frac{1}{\sqrt{1+c^2\,r^2}}\frac{\partial}{\partial h}\,(v_h) + \sqrt{\frac{p}{2\pi r^2}}\frac{\partial}{\partial \varphi}\,(v_h) \tag{2.20}$$

Introduciendo la condición de simetría eliminamos la variable h, obteniendo la expresión:

$$div\,(\mathbf{v}) = \frac{v_r}{r} + \frac{\partial v_r}{\partial r} + A\,(r)\,\frac{\partial v_\varphi}{\partial \varphi} + B\,(r)\,\frac{\partial v_h}{\partial \varphi} \tag{2.21}$$

$$A\,(r) = \frac{1}{r}\left(1 - \frac{\sqrt{c^2+r^2}}{r\,c}\right) \tag{2.22}$$

$$B\,(r) = \frac{1}{r}\left(\frac{1}{\sqrt{c}} - \sqrt{\frac{c^2+r^2}{c+c^3\,r^2}}\right) \tag{2.23}$$

2.4. Ecuaciones N-S

Las ecuaciones de Navier-Stokes en coordenadas helicoidales y bajo la condición de simetría pueden expresarse con solo dos coordenadas (r, φ) y resolverse en un dominio plano bidimensional. Esto es una gran ventaja, ya que al ser un problema plano puede resolverse con un gran ahorro computacional. El objetivo final de esta sección sería llegar a implementar dichas ecuaciones en OpenFoam y resolver el problema plano con un mallado de alta densidad.

Hay que tener en cuenta que no todas las magnitudes tienen la propiedad de simetría. Por ejemplo, la temperatura o la concentración de especies no son simétricas. Ya que, si se aplica una condición de flujo de calor en la superficie, el fluido se irá calentando y la condición de simetria no se cumplirá, al menos tal y como está definida. Con la concentración de especies ocurrirá lo mismo, ya que la difusión hará que su magnitud no se conserve en la dirección de simetría.

$$\frac{\partial \rho}{\partial t} + div(\rho \mathbf{v}) \tag{2.24}$$

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t}+\mathbf{v}\cdot\overrightarrow{grad}\left(\mathbf{v}\right)\right)=-\overrightarrow{grad}\left(p\right)+\mu\Delta\left(\mathbf{v}\right) \tag{2.25}$$

$$\frac{\partial T}{\partial t}+\mathbf{v}\cdot\overrightarrow{grad}\left(T\right)=\frac{k}{\rho c}\Delta T \tag{2.26}$$

Como ejemplo de aplicación se van a sustituir los operadores diferenciales en la ecuación de continuidad no desaparece.

$$\frac{\partial \rho}{\partial t}+\frac{1}{r}\frac{\partial}{\partial r}\left(r\rho v_r\right)+A\left(r\right)\frac{\partial}{\partial \varphi}\left(\rho v_\varphi\right)+B\left(r\right)\frac{\partial}{\partial \varphi}\left(\rho v_h\right)=0 \tag{2.27}$$

Por último comentar que el hecho de reducir a dos las dimensiones espaciales del problema no implica que se tengan que reducir el número de ecuaciones a resolver. Las ecuaciones de cantidad de movimiento segurán siendo tres. La diferencia se encuentra en que desaparece una de las variables de las que depende la solución. Esto implica que el coste necesario para resolver las ecuaciones se reduce debido a que ya no es necesario dicretizar las tres dimensiones espaciales sino solo dos. Como el objetivo de este proyecto no es implementar las ecuaciones N-S bajo la condición de simetría helicoidal en OpenFoam, esta sección va a quedar inconclusa. Aunque sí que era importante sentar las bases del concepto de simetría helicodal que posteriormente será utilizado para resolver el flujo en Fluent mediante la aplicación de condiciones períodicas, que no es más que una forma limitada que posee Fluent de aplicar este tipo de simetría.

2.5. Preprocesado

En esta sección se intenta dar respuesta a la necesidad de realizar un método paramétrico de generación semiautomática de mallados para el estudio de reactores con muelles insertados. En esta seccion ha sido muy util el contenido de los siguientes enlaces:

- Gambit Modeling Guide, `http://202.118.250.111:8080/fluent/Gambit13_help/modeling_guide/mgtoc.htm` [1]
- Gambit User's Guide, `http://202.118.250.111:8080/fluent/Gambit13_help/users_guide/ugtoc.htm` [2]

El problema de este tipo de geometrías reside en que son complejas de generar y llevan mucho tiempo, con lo cual llevar a cabo un análisis de sensibilidad de malla puede ser una tarea muy costosa.

La geometría se ha generado con el software comercial Gambit 2.4.6. Concretamente se ha utilizado el modo de entrada de comandos tipo batch mediante la generación de ficheros journal que poseen las instrucciones necesarias para que el programa ejecute correctamente

la geometría. Estos ficheros tipo journal se generan en Matlab y son la base para la construcción de la geometría.

2.5.1. Construcción de la geometría en Gambit

A continuación se describe el proceso seguido en Gambit para ejecutar la geometría. Las imágenes que se muestran abajo son un resumen gráfico de los pasos más importantes que es necesario realizar. Todos estos pasos se ejecutan en el modo TUI mediante archivos tipo $*.jou$ generados en Matlab con la función *cmsh* (A.4). En (Figura 2.1) se representa los dos elemtos que definen el muelle, como son la sección transversal y el carril de paso normal a la superficie por el cual dicha sección circular avanza para generar el muelle.

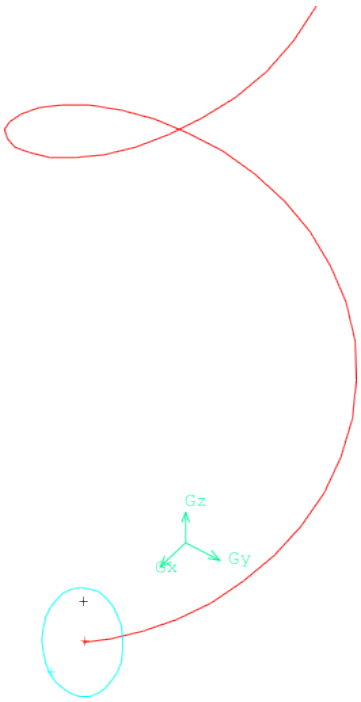

Figura 2.1: Sección normal a la helice.

Posteriormente es necesario crear el volumen base del muelle (Figura 2.2) mediante una operación de extrusión.

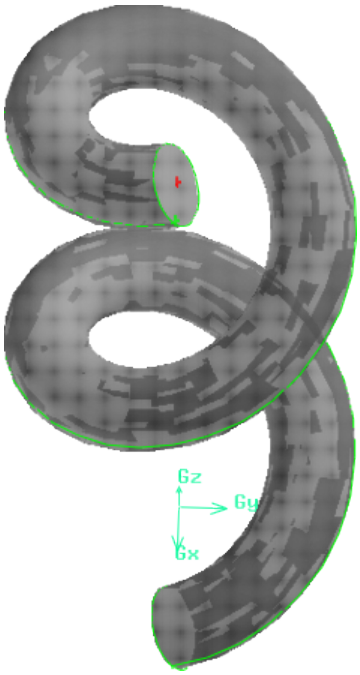

Figura 2.2: Volumen base del muelle.

El siguiente paso es ejecutar una operación booleana de intersección entre el muelle y un plano horizontal (Figura 2.3). La curva resultante de dicha operación queda contenida en el plano X-Y.

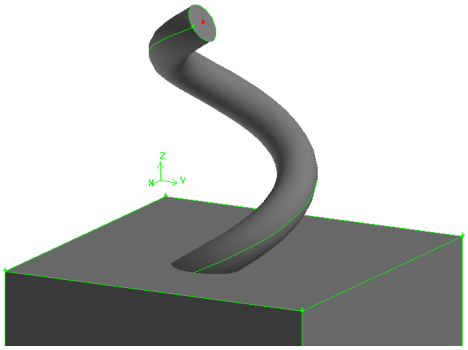

Figura 2.3: Intersección entre el muelle y un plano horizontal.

A continuación se dibuja el círculo que se obtendría de la sección entre las paredes del tubo y el mismo plano horizontal (Figura 2.4). Las secciones resultantes del muelle y del tubo se combinan nuevamente en una operación booleana de extracción para conseguir que el dominio computacional se reduzca exclusivamente al espacio libre que queda entre el muelle y el tubo (Figura 2.5), eliminando así la parte de sólido que resultó de la operación de intersección con el muelle.

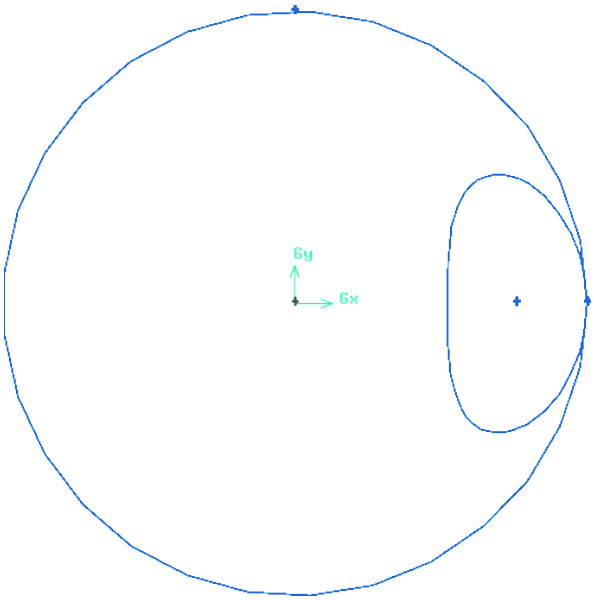

Figura 2.4: Sección transvesal de tubo y muelle.

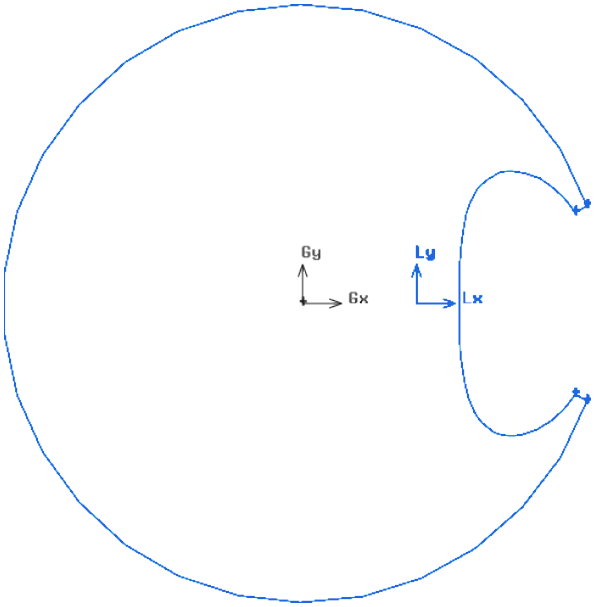

Figura 2.5: Sección transvesal tubo y muelle con contacto puntual resuelto.

A continuación es necesario resolver un problema geométrico de contacto puntual (Figura 2.4). En la realidad el tubo y el muelle se tocarían en un punto o en un dominio tan pequeño que queda muy por debajo de la resolución de cualquier mallado.

Por ello, la solución es eliminar dicho contacto puntual mediante un corte que elimine una parte insignificante a efectos del flujo que se pretende resolver.

Aunque aparentemente el trozo eliminado (Figura 2.6), es considerable no afecta en abso-

luto al fluido ya que es una zona muerta con velocidad despreciable.

En el caso de querer resolver un problema de transmisión del calor, el trozo de superficie eliminado sí introduce un error que debería ser estimado.

Esta situación es un problema relativamente frecuente que se produce siempre que exista contacto entre dos superficies con distinto radio de curvatura.

Figura 2.6: Detalle de la simplificación del contacto puntual.

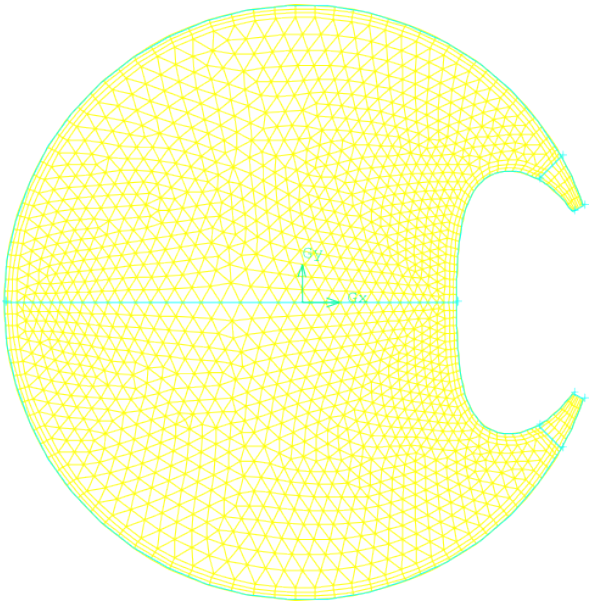

Figura 2.7: Mallado de la sección transversal.

El siguiente paso requiere del criterio del diseñador, ya que consiste en la definición del mallado plano de la sección transversal.

Para ejecutar el mallado y conseguir que cumpla con ciertas propiedades de simetría o de número de elementos.

El dominio plano se parte en tantas caras como desee el usuario, el cual solo tendrá que especificarlo posteriormente en el programa de generación de comandos para que los ficheros tipo journal funcionen correctamente.

Una operación típica de mallado requiere de la definición de una capa límite y de un mallado del resto del dominio (Figura 2.7). Esta operación puede parametrizarse en función de un tamaño de elemento si se quisiera realizar un análisis de sensibilidad de malla.

El hecho de que se defina una capa límite no implica que se piense en resolver un flujo turbulento. Sea cual sea el régimen de flujo, la capa límite ayudará a asegurar una distancia uniforme alrededor del contorno y siempre se considera este paso como una buena práctica.

El siguiente paso consiste en utilizar la función $SweepRealFaces$ para generar un volumen de pequeño espesor, que será el volumen fuente a partir del cual se generarán todos los demás (Figura 2.8).

Antes de usar la función de extrusión $SweepRealFaces$ es muy importante desactivar un bandera de la configuración de Gambit:

default set "GEOMETRY.VOLUME.SWEEP_PATH_ALIGNMENT

Si esta función no se desactiva, la extrusión con rotación que se realiza sobre cada una de las caras de la sección plana de la (Figura 2.7) se realizará alrededor del eje vertical que pasa por el centroide de cada una de las caras, de tal forma que se tendrá un eje de rotación por cada cara obteniendo un volumen inconexo.

Cuando se descativa la bandera mencionada, el eje de rotación pasa a ser el eje z, de tal forma que el volumen resultante tiene simetría helicoidal respecto del eje z, ver (Figura 2.8).

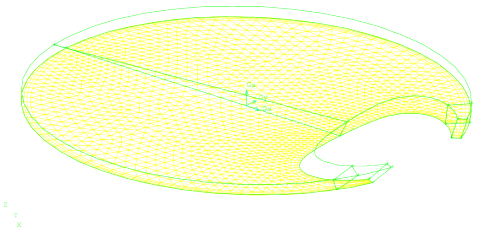

Figura 2.8: Generación de la primera capa.

Cuando se decide el tamaño de la discretización del dominio computacional con un mallado estructurado como este, es necesario definir al menos dos parámetros de discretización representativos del tamaño de los elementos.

Uno de ellos ya ha sido mencionado y se refiere al tamaño de los elementos en la sección transversal, mientras que el otro se refiere al tamaño de los elementos según la dirección axial.

La elección de este parámetro tiene implicaciones en la generación de la geometría, ya que el espesor de la capa determinará el ángulo de rotación entre sus caras superior e inferior. Esta relación viene determinada por la (Ecuación $\Delta\varphi = 2\pi\frac{\Delta h}{p}$). Y es necesario conocerla para poder definir correctamente la torsión de la geometría (Figura 2.9).

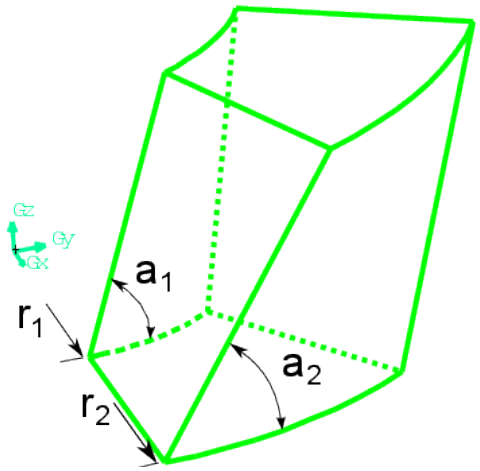

Figura 2.9: Torsión de un elemento cualquera

La relación que existe entre el ángulo de inclinación del elemento a y la distancia r al eje de torsión es

$$tan(a) = \frac{p}{2\pi r} \tag{2.28}$$

Este ángulo disminuye con el aumento de la distancia al eje de torsión y para valores inferiores a 25º la deformación del mallado empieza a estar cerca del límite de oblicuidad (skewness ratio max 0.9).

Este valor es aproximadamente el mismo que el de la geometría utilizada en esta memoria. Este hecho hace que la convegencia del solver se vea seriamente afectada y si no se utilizan parámetros de sobrerelajación muy ajustados, podría ser que la convergencia no llegue a lograrse.

Existen dos tipos de errores que se cometen al aproximar la geometría por trazos de línea recta. Estos errores dependen del tamaño h de la discretización según la dirección axial y pueden reducirse tanto como se quiera, aunque siempre va interesar que el valor de h sea del mismo orden que el tamaño de celda en el plano horizontal con el objetivo de no obtener elementos con una dimensión mucho maayor que las otras dos.

Error por reducción del volumen. Se debe al trozo de volumen que quedaría entre la geometría ideal y a la geometría discretizada. El volumen que se pierde y la consecuente reducción que se produce es proporcional al área sombreada (Figura 2.10).

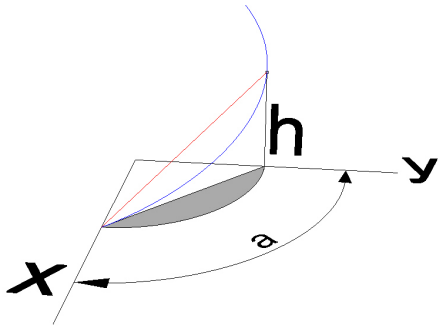

Figura 2.10: Disminución del volumen efectivo debido a la discretización.

Error por incremento del ángulo de hélice. Se debe al hecho de que al tomar un punto de cota h se produce un incremento respecto del ángulo real de inclinación de la hélice, esta diferencia queda representada por el ángulo b (Figura 2.11).

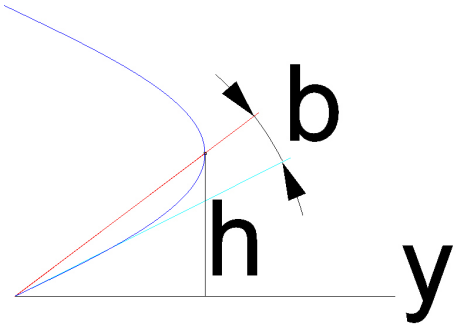

Figura 2.11: Aumento del ángulo de incidencia de la hélice debido a la discretización.

Una vez resuelta la generación del volumen diferencial, el usuario tiene que definir las paredes tipo “Wall” que delimitan el dominio computacional, así como definir las caras que forman la entrada “Inlet” al volumen y las caras que forman la salida “Outlet” del volumen, además de “Interior” para el propio volumen diferencial.

A continuación comenzaría el proceso de replicación del primer volumen diferencial. El hecho de definir previamente todas las condiciones de contorno para una sola capa se justifica por el hecho, de que las réplicas del primer volumen heredan todas las condiciones de contorno, por lo tanto ya no es necesario volver a preocuparse por ese tema, y teniendo en cuenta que un tubo puede contener del orden de cientos de capas es un aspecto a tener en cuenta.

El proceso de réplica consiste en tomar siempre como referencia el primer volumen y hacer un proceso de elevación y rotación alrededor del eje z para ir formando todo el volumen (Figura 2.12). El hecho de que se use siempre el primer volumen para replicar todos los

demás, y no se haga utilizando el último que se creó (algo que sería más fácil de programar debido a que el incremento de altura y de giro entre dos capas consecutivas es siempre constante). Se debe al hecho que de esta forma se evita propagar los errores numéricos que se generan con cada rotación especialmente; y que al cabo de cientos de capas puede llegar a tener el mismo orden que la precisión absoluta de Gambit (10^{-6}) para la salida de usuario.

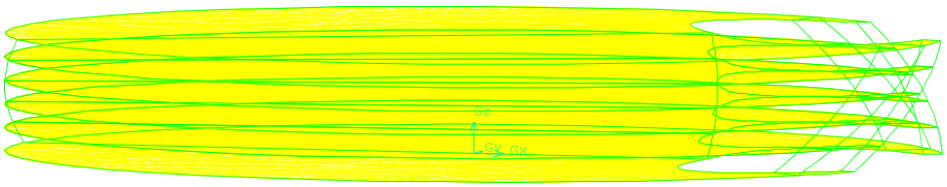

Figura 2.12: Generación de las primeras capas.

Una vez se han copiado todos los volúmenes para dar tantos pasos de muelle como se desee, es necesario conectar la geometría para que no queden solapamientos entre secciones transversales intermedias (Figura 2.13). Este problema se debe al método de generación que se ha empleado. Cuando Gambit copia un volumen no comprueba en ningún momento que dos caras de dos volúmenes distintos puedan quedar solapadas.

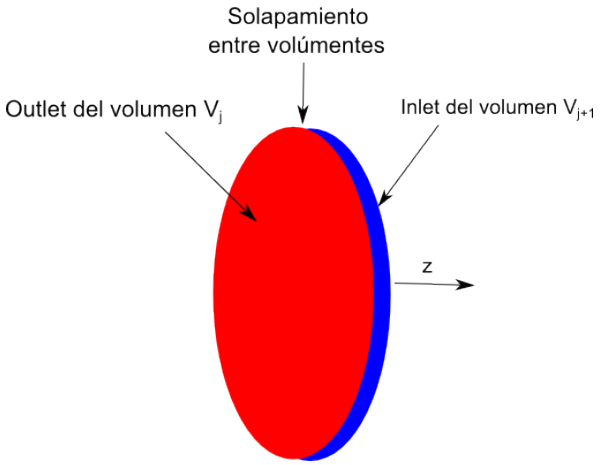

Figura 2.13: Solapamiento entre volúmenes consecutivos.

Si no se eliminan todas las caras solapadas, la geometría tendrá un error que se manifestará en la ejecución del solver, ya que cada solapamiento será exportado como una pared, quedando así el volumen del tubo dividido en infinidad de celdas inconexas.

Para resolver dicho problema, el mejor método es usar la herramienta de conectar caras solapadas, ya que Gambit chequea los solapamientos y los elimina automáticamente respetando la definición previa de los volúmenes. Este método tiene una gran dificultad, ya que se plantean dos posibilidades:

- Ordenar un chequeo total de todas las caras.

  Gambit comprueba dos a dos hasta encontrar un solapamiento. Pero sabiendo que para cada cara se va a tener que comprobar todo el volumen, este proceso se plantea como excesivamente lento e incluso inestable, ya que si el volumen tiene muchas caras; algo que podría pasar en un tubo con diez o más pasos de muelle. Probablemente se produzca un desbordamiento de memoria y el programa se salga de la ejecuación.

- Conectar en cada caso exactamente las caras solapadas.

  Esto exige conocer la numeración de las caras de la todas las secciónes transversales del tubo y el orden en que están solapadas. A priori es un problema excesiamente complejo, debido a que es fácil comprobar que Gambit no sigue un patrón de numeración de caras que sea totalmente predecible.

La mejor forma de resolver este problema es parando justo en este punto la ejecución de la línea de comandos y extrayendo un listado de todas las caras que pertenecen a los conjuntos:

- Inlet
- Outlet

Gambit proporciona esta información a través de un fichero tipo $*.trn$ o también $*.jou$ que se actualiza en tiempo real. Además la numeración de estos conjuntos de caras sí están ordenados según el momento en que se crearon por lo tanto no es muy complicado ir extrayendo los pares Inlet-Outlet de cada lista e ir formando parejas de conexión para que Gambit las conecte una a una.

De esta forma se resuelve uno de los mayores problemas que aparecen con la generación de volúmenes por replicado. Realmente este el verdadero motivo para haber definido en la primera capa las condiciones de contorno tipo “Inlet” y “Outlet”.

Después de realizar esta operación el tubo queda completamente definido y listo para exportar (Figura 2.14).

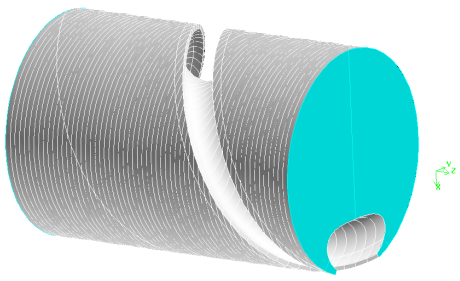

Figura 2.14: Resultado final.

2.6. Análisis lagrangiano

El desarrollo que se detalla a continuación resume los conceptos del análisis cinemático.

2.6.1. Cinemática del campo fluido

Un fluido se describe normalmente en términos de la velocidad de cada partícula. La velocidad **v** del fluido depende de la posicion del espacio **x** y del tiempo t. Los fundamentos teoricos de este apartado estan detallados en *Mecánica de Fluidos General* [3].

$$\mathbf{v} = \mathbf{v}(\mathbf{x}, t) \tag{2.29}$$

Para caracterizar el movimiento del fluido, pueden utilizarse dos puntos de vista distintos:

Modelo euleriano

El modelo euleriano se centra en estudiar el campo de velocidades que se obtiene de resolver un sistema de ecuaciones diferenciales:

$$\frac{\partial \mathbf{x}}{\partial t} = \mathbf{v}(\mathbf{x}, t) \tag{2.30}$$

Este método es muy útil cuando se pretende estudiar el movimiento del fluido en todo su dominio, o al menos en ciertas zonas, para entender los fenómenos de transferencia de calor, de caídas de presión en tubos, recirculaciones, etc.

Modelo lagrangiano

Este método se centra en el estudio del movimiento de cada partícula individualmente. Lo que se hace es obtener las trayectorias de las partículas de forma individual. Para ello se integra el campo de velocidades a través de una ecuación integral

$$\mathbf{x}_p = \int_{t_0}^{t} \mathbf{v}(\mathbf{x}, t) dt \tag{2.31}$$

Donde $\mathbf{x}_p$ es la posición instantánea de la partícula que se encontraba en el instante t en la posición $\mathbf{x}_0$

$$\mathbf{x}_p = (\mathbf{x}_0, t) \tag{2.32}$$

La integración de la ecuación de la trayectoria exige conocer en cada punto $(\mathbf{x}, t)$ del dominio la solución del campo de velocidades. Lo cual implica previamente resolver las ecuaciones de Navier-Stokes (2.24).

Por ello se puede decir que el estudio lagrangiano es posterior a la resolución del problema fluido dinámico y consistiría dar un paso más para entender el comportamiento de problemas relacionados con el transporte de especies o de los fenómenos de convección o de mezclado.

2.6.2. Aplicación del análisis lagrangiano al problema de flujo con simetría helicoidal

Como ya se ha mencionado, el problema de flujo helicoidal queda completamente definido si se conoce la velocidad en todos los puntos de una sección transversal.

La reconstrucción de la solución en todo el dominio se consigue rotando el campo de velocidad en la sección de partida un ángulo proporcional a la diferencia de cotas entre ambas secciones.

$$\Delta\varphi = -\frac{2\pi}{p}\Delta h \tag{2.33}$$

El signo negativo de esta ecuación hace referencia únicamente al hecho de que la rotación de la sección debe ser contraria al avance. La aplicación de la simetría helicoidal tiene grandes ventajas en el estudio de flujos estacionarios porque la solución del campo de velocidades no depende del tiempo.

En el caso de flujo oscilatorio el campo de velocidades es periódico en el tiempo y solo es necesario conocerlo en un ciclo completo para reconstruir totalmente la solución en cualquier otro instante de tiempo.

Capítulo 3

Programación de algoritmos

La implementación de los algoritmos se ha realizado en Matlab.

3.1. Integración de trayectorias

Para llevar a cabo la integración del campo de velocidades utilizando el método lagrangiano se ha diseñado un algoritmo (Figura 3.1) que resume los pasos más importantes.

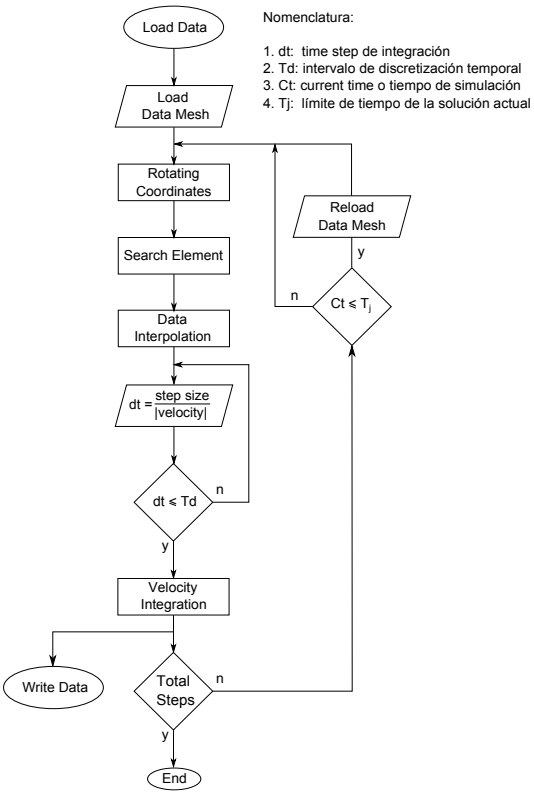

Figura 3.1: Algoritmo funcion pathline.

Como ya se sabe la reconstrucción de todo el campo de velocidades puede realizarse a partir de una sola sección transversal. Por lo tanto el algoritmo de cálculo de trayectorias utiliza este mismo concepto.

element	node i	node j	node k
150	77	54	84
157	41	54	77
158	73	54	41

Tabla 3.1: Lista de enumeracion de elementos en Gambit.

3.1.1. Load Data

El algoritmo comienza cargando los datos definidos por el usuario "Load Data", que se corresponden con la posición inicial de las partículas y la solución del campo de velocidades. (codigo A.13), los datos mínimos para que el programa funcione son:

- Posiciones iniciales de las partículas
- Datos del mallado extraido de Gambit
- Datos de la solución en los nodos extraido de Fluent

Los datos del mallado exportados por Gambit consisten en dos listas:

- Lista de elementos
- Lista de nodos

La primera lista contiene la información de la numeración de cada elemento de la sección y de los nodos que la forman, de tal manera que el orden que asigna Gambit a cada nodo de un elemento no es casual, si no que siempre están numerado en el sentido contrario a las agujas del reloj. Esto es una necesidad muy importante sobre la que se hablará posteriormente.

En (Figura 3.2) se representa un ejemplo de numeración de nodos realizado por Gambit. Como ejemplo se va a escribir el listado que Gambit devolvería para este caso.

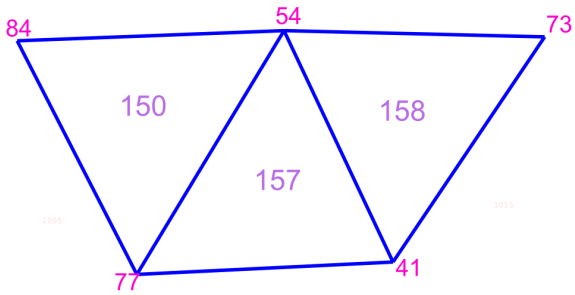

Figura 3.2: Numeracion de elementos en el mallado.

Como se aprecia en el ejemplo, la primera columna se reserva para para la numeración de elementos y las siguientes para la numeración de los nodos del elemento. Aunque en este ejemplo todos los elementos son triangulares, esto no es una condición pudiendo encontrar filas para elementos con mayor número de nodos.

Los dos tipos de elementos que podrían aparecer en el mallado son:

- Elemento triangular $T3$
- Elemento rectangular $C4$

Ambos son elementos diseñados para interpolación lineal mas informacion en el libro *El Método de los Elemento Finitos en Ingeniería* [4].

Los datos de la solución exportados por Fluent consisten en una única lista que contiene en cada fila los datos de cada nodo del mallado. Dicha lista tendrá tantas filas como nodos tenga la malla.

El problema de numeración consiste en que Gambit exporta los nodos con una numeración distinta a Fluent, de hecho, el problema reside en que Fluent no respeta la numeración pasada por Gambit y siempre que datos de entidades en nodos, comienza la numeración por 1 y prosigue consecutivamente para toda la lista.

3.1.2. Load Mesh

El siguiente paso del programa consiste en cargar todos los datos del mallado exportados tanto desde Fluent como desde Gambit. Debido al problema de numeración antes mencionado este paso no puede hacerse directamente ya que es imposible casar la numeración de las listas de Fluent y Gambit. Esto supuso un punto crítico en el desarrollo del programa pero se pudo solucionar con el desarrollo de la función "loadnode" (codigo A.9).

Esta función toma las listas de coordenadas provenientes de Fluent y Gambit y las cruza tomando como criterio el valor de las coordenadas (x, y) encontrando la relación que existe entre la numeración de ambas listas cuando coinciden las coordenadas de un nodo en una y otra lista.

Esta función también tiene en cuenta el hecho de que los datos exportados desde Fluent y Gambit no tienen porque tener la misma cota, lo que supone tener que girar todos los datos de una de las listas un cierto ángulo para que coincidan nuevamente con las coordenadas de la otra.

3.1.3. Rotating Coordinates

El siguiente paso consiste en adaptar las posiciones de las partículas en cada instante, a la sección en la que se poseen los datos de velocidad y posición de nodos.

La estrategia seguida consiste en calcular la diferencia de cotas h entre el punto y el plano de referencia al que pertenencen los datos y rotar la posición del punto alrededor del eje z con el objetivo de situar sus coordenadas sobre el plano de referencia.

La matriz de rotación usada realiza un giro alrededor del eje z:

$$\mathbf{R}(\varphi) = \begin{bmatrix} cos(\varphi) & -sin(\varphi) & 0 \\ sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.1}$$

Este proceso se realiza al comienzo de cada nuevo paso de tiempo (Figura 3.3) y la relación existente entre los puntos $\mathbf{x}_{tj}$ y $\mathbf{x}^*_{tj}$ es:

$$\mathbf{x}_{tj} = \mathbf{R}\mathbf{x}^*_{tj} \tag{3.2}$$

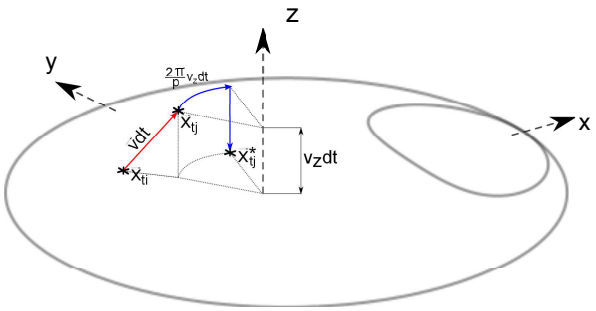

Figura 3.3: Metodo de transformacion de coordenadas al plano de referencia.

3.1.4. Search Element

Una vez se conocidas las coordenadas del punto $\mathbf{x}_p$ en el plano de referencia. Es necesario encontrar el elemento al que pertenece con el fin de utilizar los datos de los nodos para interpolar en el punto en cuestión.

Para conocer el elemento al que pertenece un punto se ha utilizado el algoritmo *whoelem* (A.20), que se describe a continuación.

La función toma como entrada las coordenadas del punto $\mathbf{x}_p$ y recorre toda la lista de elementos hasta encontrar uno que contenga al punto.

El criterio de decisión que se ha utilizado para saber si un punto pertenece o no a un elemento nace del hecho de que en Gambit al igual que la mayoría de programas de elementos finitos numera los nodos siguiendo un criterio, concretamente la numeración siempre es en sentido contrario a las agujas del reloj si el observador se sitúa desde el lado desde el que se definió la normal positiva de la cara.

El procedimiento consiste en recorrer todas las aristas o lados de un elemento en el sentido contrario a las agujas del reloj, (Figura 3.4) y si el punto siempre queda a la izquierda de la arista, para todas las aristas del elemento, es porque el punto pertenece a dicho elemento.

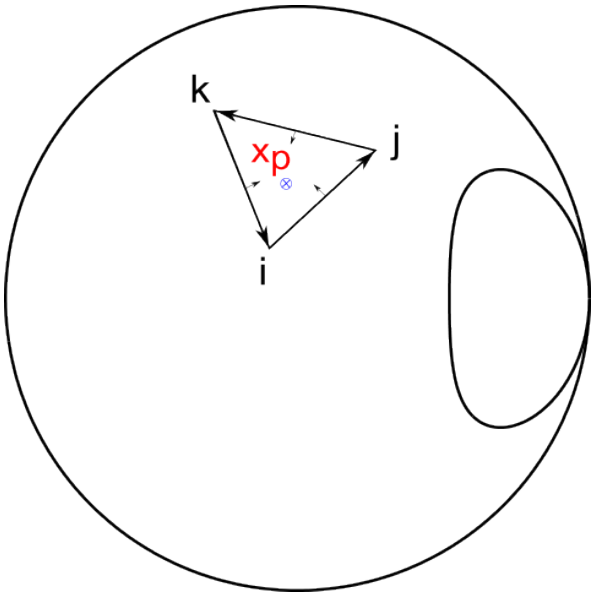

Figura 3.4: Punto interior a un elemento cualquiera del mallado.

Obviamente solo existe un elemento que cumpla dicha propiedad y cuando el algoritmo iterativo da con él, para y devuelve la numeración de dicho elemento.

El proceso de búsqueda puede llegar a suponer un gran coste porque si el elemento no se encuentra de los primeros en la lista, el algoritmo va a tardar mucho tiempo en llegar hasta su posición.

Para solucionar este problema es necesario utilizar algún algoritmo de búsqueda que optimice el tiempo. Para lo cual se ha diseñado un criterio de búsqueda específico que parte de la propiedad de que cuando Gambit numera los elementos del mallado, aunque no sigue un patrón de generación predecible si es cierto que es progresivo, de tal forma que elementos cercanos tienen numeraciones también próximas.

Si esta idea se une al hecho de que cuando se resuelve la trayectoria de una partícula los puntos por los que va pasando estarán próximos entre sí. Parece lógico pensar que un buen criterio de búsqueda consiste pasar a la función *whoelem* (A.20) el punto que se desa buscar la posición del último elemento que se encontró.

De esta forma el programa abre la lista por el elemento que encontró en la posición anterior de la trayectoria y comienza a recorrerla hacia arriba y hacia abajo desde ese punto hasta que da rápidamente con el elemento vecino (Figura 3.5).

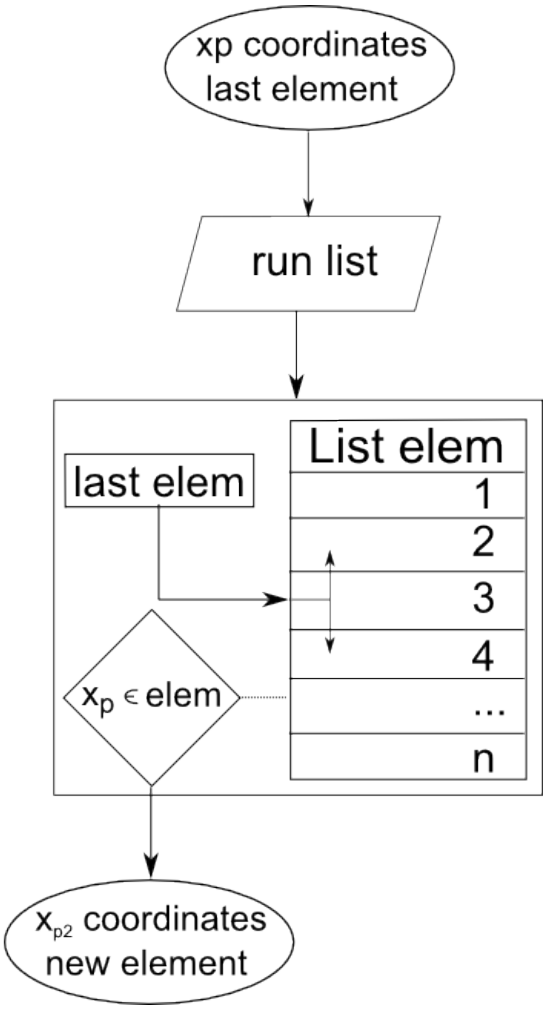

Figura 3.5: Algoritmo de busqueda de elementos.

Este criterio de búsqueda redujo el tiempo de computación en unas diez veces menos respecto de no usar ningún criterio y empezar siempre la lista por el principio.

3.1.5. Data interpolation

Una vez que el programa ha encontrado el elemento al que pertenece el punto de coordenadas $\mathbf{x_p}$. Tiene que interpolar entre los datos de los vértices o nodos de dicho elemento.

Para realizar esta operación se recurre a las funciones de forma C_0 para el elemento triangular lineal $T3$ o el elemento cuadrangular lineal $C4$, en la (Figura 3.6) se muestra como ejemplo una de las funciones de forma del elemento $C4$ para interpolar en uno de sus nodos.

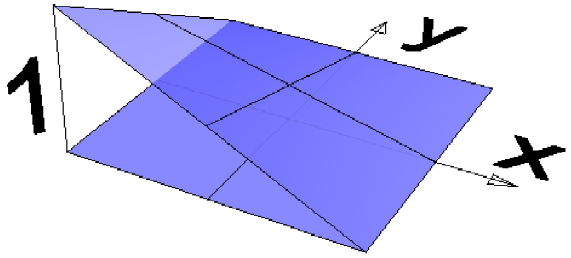

Figura 3.6: Funcion de forma para elemento cuadrangular lineal C4.

Las funciones de interpolación son *nodinterpole* (A.11) y *weigthfun* (A.19) que son las que se encargan de distinguir entre los dos tipos de elementos ($C4$ y $T3$).

Las funciones de forma para el elemento $T3$ se obtienen directamente a través de:

$$\begin{array}{lll} a_i = x_j y_k - x_k y_j & b_i = y_j - y_k & c_i = x_k - x_j \\ a_j = x_k y_i - x_i y_k & b_j = y_k - y_i & c_j = x_i - x_k \\ a_k = x_i y_j - x_j y_i & b_k = y_i - y_j & c_k = x_j - x_i \end{array} \tag{3.3}$$

$$\Delta = \frac{a_i + a_j + a_k}{2} \tag{3.4}$$

$$N_i = \frac{a_i + b_i x + c_i y}{2\Delta} \tag{3.5}$$

$$N_j = \frac{a_j + b_j x + c_j y}{2\Delta} \tag{3.6}$$

$$N_k = \frac{a_k + b_k x + c_k y}{2\Delta} \tag{3.7}$$

Y para el elemento $C4$ las funciones de forma para para los nodos $c = (i, j, k, m)$ se obtienen de la ecuación:

$$N_c = \frac{1}{4}(1 + \xi\xi_c)(1 + \eta\eta_c) \tag{3.8}$$

Donde las variables (ξ, η) representan las coordenadas locales del elemento $C4$.

El problema de las funciones de forma del elemento $C4$ es que hay que pasar las coordenadas cartesianas del punto que se quiere calcular a coordenadas locales, para lo cual es necesario resolver las ecuaciones de transformación de coordenadas. Esto lo realiza la función *weigthfun* (A.19).

Posteriormente la función *nodinterpole* (A.11) multiplica las funciones de forma por los datos de los nodos obteniendo como resultado una interpolación lineal en el interior de cada elemento para un punto determinado.

3.1.6. Interpolación temporal

Con cada paso que se resuelve el tiempo va avanzando entre los instantes t_i y t_j que son los instantes de tiempo donde se conoce la solución. Para calcular el campo de velocidades en un instante t intermedio se hace una interpolación temporal lineal:

$$x = \frac{t - t_i}{t_j - t_i} \tag{3.9}$$

$$\mathbf{v}_t = x\mathbf{v}_{t_j} + (1 - x)\mathbf{v}_{t_i} \tag{3.10}$$

3.1.7. Velocity Integration

Una vez que se ha calculado la velocidad $\mathbf{v}_j$ en el punto $\mathbf{x}_j$ considerado, el programa la integra para un Δt obteniendo la posición en el siguiente punto:

$$\mathbf{x}_{j+1} = \mathbf{x}_j + \mathbf{v}_j \Delta t \tag{3.11}$$

3.1.8. Write Data

El último paso que realiza el programa es guardar los últimos datos en una variable para su posterior procesamiento. El programa también exporta los datos a un fichero ASCII para poder cargarlos en algún programa de dibujo que lo soporte. En el siguiente apartado pueden verse algunas imágenes de ejemplo resueltas con Rhinoceros.

3.2. Configuración del solver en Fluent

Para la resolución del flujo se ha utilizado el programa de CFD Fluent 6.3.26 como se mencionó con anterioridad. En esta sección se resumen el tipo de configuración utilizada por el solver para resolver el flujo estacionario y oscilatorio.

3.2.1. Flujo estacionario

El modelo de viscosidad utilizado es “laminar” ya que se considera se considera que para los valores de Reynolds utilizados $(1, 10, 80)$ el flujo es completamente laminar.

El solver utilizado es:

- Pressure Based

- Formulation Implicit
- Absolute Velocity Formulation

Los factores de sobre-relajación utilizados son:

- Pressure: 0.3
- Density: 1
- Body Forces: 1
- Momentum: 0.5

Realmente los únicos que afectan a la convergencia son el de presión y los de cantidad de movimiento. El esquema de discretización utilizado es:

- Pressure: Standard
- Momentum: First order Upwin

Las condiciones de contorno que hay que definir para un flujo periódico en tubo sin transferencia de calor son dos:

- Boundary Conditions: Fluid-water
- Periodic Conditions: Mass-Flow-Rate

Residuales:

- Continuity: 1e-5
- X-momentum: 1e-5
- Y-momentum: 1e-5
- Z-momentum: 1e-5

3.2.2. Flujo oscilatorio

La resolución del flujo oscilatorio supone activar el solver "unsteady" y definir la ley de variación del gasto másico según las ecuaciones vistas en el capítulo de introducción.

Esta cuestión podría abordarse de dos formas distintas:

- Definición de una UDF para el gasto másico
- Definición del gasto másico a través de un journal

Realmente el problema ha sido abordado desde ambos puntos de vista, incluso con UDFs interpretadas y compiladas, sin embargo finalmente se ha optado por utilizar la línea de comandos a través de la gestión de un fichero journal porque resultaba más sencillo a la hora de cargar los casos en el servidor de la universidad: labmach.upct.es

Las funciones *OSCjou* (A.22), *OSCflow* (A.12) y *meshcicles* (A.10) que se explican a continuación sirven para generar los ficheros journal de forma automática utilizando como entrada de parámetros: $(Ciclos, Reynolds, Strouhal, fluido)$. La función *OSCflow* (A.12) genera todas las propiedades que puedan ser interesantes para caracterizar el flujo oscilatorio:

- REYNOLDS: número adimensional definido por el usuario
- STROUHAL: número adimensional definido por el usuario
- WOMERSLEY: número adimensional que caracteriza el flujo pistón
- FLUID: tipo de fluido usado
- DENSITY: densidad del fluido
- VISCOSITY: viscosidad del fluido
- FRECUENCY: frecuencia de oscilación
- TIMECICLE: periodo de oscilación
- VELOCITY: velocidad media máxima del flujo oscilatorio
- MASSFLOW: gasto másico máximo del flujo oscilatorio
- PLUNGER: desplazamiento piston máximo
- PRESSURE DROP SMH: caída de presión para un tubo liso en régimen estacionario laminar con el mismo valor del número de Reynolds que el del flujo oscilatorio simulado
- PRESSURE DROP STD: caída de presión para un tubo con muelle en régimen estacionario laminar para el mismo valor del número de Reynolds que el del flujo oscilatorio simulado
- PRESSURE DROP OSC: máxima caída de presión que se produce para flujo oscilatorio con un Reynolds y un Strouhal determinado

La función *meshcicles* (A.10) toma algunos de los datos generados por *OSCflow* (A.12) y los usa para generar la discretización temporal.

En esta función el usuario define el número de puntos que desea obtener por ciclo, y la función se encarga de espaciarlos para conseguir el en cada paso de tiempo el salto de gasto másico sea el mismo. Con esto se consigue un método sencillo de realizar una adaptación temporal a la señal oscilatoria de entrada (Figura 3.7).

La ecuación que hay que resolver para encontrar los instantes de tiempo que cumplen con la condición salto de gasto másico constante es:

$$y(t_j) - y(t_i) = k \tag{3.12}$$

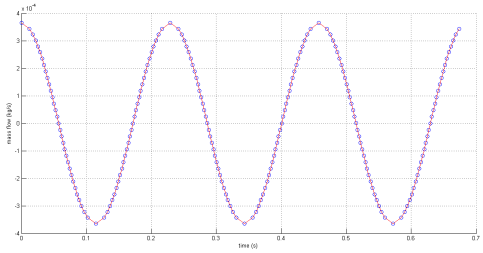

Figura 3.7: Representacion del mallado temporal para un gasto basico adaptado.

El problema se resuelve en *meshcicles* (A.10) mediante el método del punto medio. Por último, la función *OSCjou* (A.22), toma los datos obtenidos por las dos funciones anteriores y genera el fichero journal para resolver el flujo oscilatorio en Fluent.

3.3. Resultados

En este capítulo se van a representar algunas gráficas de los resultados que se han obtenido con las simulaciones y la generación de las trayectorias.

Se han representado tres imágenes representativas de los tres tipos de resultados que han obtenido.

Estas gráficas ejemplifican los resultados que se pueden obtener con los códigos desarrollados. La primera de ellas (Figura 3.8) muestra el desarrollo de las trayectorias de algunas partículas lanzadas a lo largo del eje y, y que evolucionan a lo largo del tubo para un flujo laminar de tipo estacionario.

La segunda de las imágenes (Figura 3.9) representa la trayectoria de algunas partículas lanzadas a lo largo del eje y para un ciclo de flujo oscilatorio.

La tercera imagen (Figura 3.10) representa el mezclado radial que se produce en el entorno de un punto para un ciclo de flujo oscilatorio.

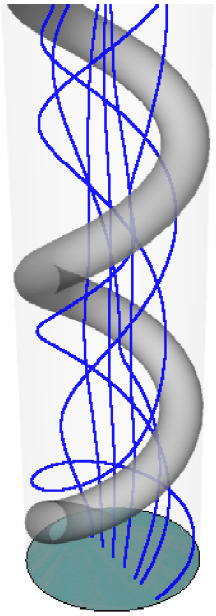

Figura 3.8: Trayectorias en flujo estacionario ($Re = 80$).

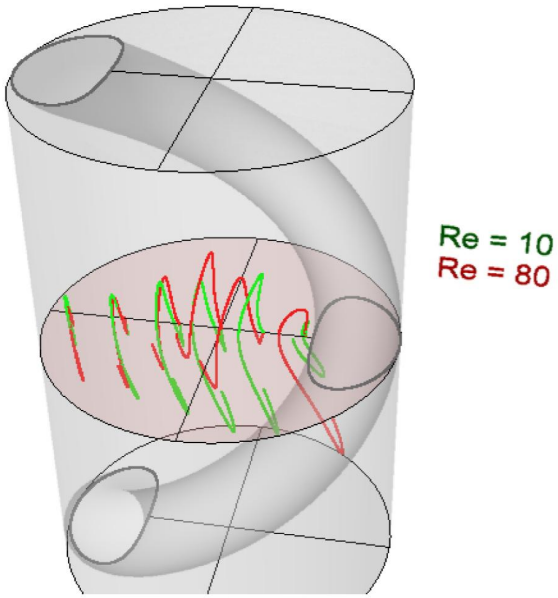

Figura 3.9: Trayectorias en flujo oscilatorio ($Re = 10, 80$).

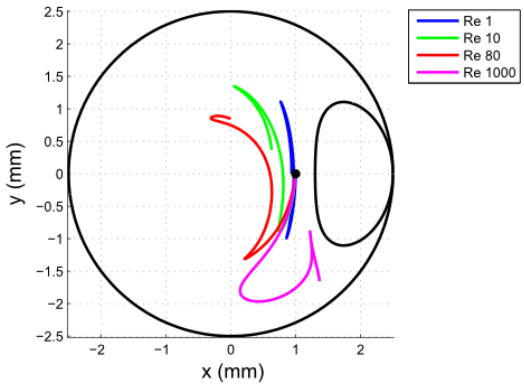

Figura 3.10: Mezclado radial en el entorno de un punto de coordenadas (0,1).

Capítulo 4

Conclusiones

Se ha resuelto satisfactoriamente el estudio del flujo en un tubo con muelle insertado aplicando la condición de simetría.

Gracias al establecimiento de esta condición el problema tridimensional puede reducirse a un problema plano en dos variables.

Esto supone un ahorro en el coste computacional y de almacenamiento de los datos.

La implementación exitosa de las función de generación de malla supone un importante ahorro en tiempo de generación de mallas con distintos refinados, o parámetros geométricos. Esto podría ser una gran ayuda para un futuro análisis de optimización de los parámetros geométricos del tubo, pensando en optimizar alguna variable del flujo como el tiempo de residencia.

Es pensando especialmente en el parámetro del tiempo de residencia, tan crítico en el diseño de reactores OBF, el motivo por el que se ha desarrollado la aplicación de cálculo de trayectorias, la cual hace prescindir de las herramientas de análisis de partículas disponibles en Fluent, que siendo mucho mejores que el código aquí generado requieren de un gran coste computacional ya que hacen necesario generar longitudes de tubo muy largas para obtener los resultados.

Las dificultades que he tenido que salvar para realizar este proyecto no han sido pocas, ya que familiarizarse con el preprocesador Gambit y dominarlo para poder generar el código en la línea de comandos ha sido bastante complicado.

Por otro lado resolver el código de cálculo de trayectorias necesitó salvar pequeños detalles sin los cuales hubiese sido imposible hacer un código con un rendimiento aceptable. Como por ejemplo resolver la parte del código que busca el elemento de la malla al que pertenece un punto. También supuso un inconveniente buscar un criterio de búsqueda que redujera al mínimo el número de elementos que es necesario comprobar hasta encontrar al que pertenece el punto.

Bibliografía

[1] F. Inc., “Gambit modeling guide.”

[2] F. Inc., “Gambit user’s guide.”

[3] Antonio Viedma Robles, *Mecanica de fluidos general*. UPCT, 2009.

[4] Departamento estructuras y construccion UPCT, *El metodo de los elementos finitos en ingenieria*. UPCT, 2009.

ANEXOS

Anexo A

A.1. Codigos

```
function Xrel = abstorel(Xabs)
%Funcion que toma las pathlines de un conjunto de particulas y las expresa
%en sus coordenadas relativas para cada cota.
%
%Xabs: es una matriz de datos con la estructura siguiente:
%
%   x-abs   y-abs   z-abs   id
%   .....   .....   .....   ..
%
%donde x-abs, y-abs, z-abs son las coordendas absolutas de cada particula,
%donde id es el identificador de cada particula.
%
%Xrel: es la transformacion de Xabs a coordendas relativas de cada seccion.
sizeXabs = size(Xabs);
Xrel  = Xabs;
for i = 1:sizeXabs(1)
    Xrel(i,[1 2 3]) = RHC(Xabs(i,[1 2 3]));
end
```

Listado A.1: Pasa de coordenadas absolutas a relativas

```
function a = angle(z)
%Funcion que calcula el angulo polar en sentido antihorario desde el eje +x
%
%Input:
%------
%       z:  vector [x-coordinate, y-coordinate]
%
%Output:
%-------
```

```
%       angulo [0,360] en sentido antihorario medido desde el eje +x

%esta funcion es necesaria cuando se quiere obtener el
%angulo en polares partiendo del dato de coordenadas,
%ya que la funcion atan(y/x) de matlab por si sola no
%devuelve el angulo medido en sentido antihorario desde
%el eje +x en el rango (0, 2*pi)

dim = size(z);
a = zeros(dim(1),1);
for i = 1:dim(1)
    x = z(i,1)+1.0-1.0;
    y = z(i,2)+1.0-1.0;

        if x >= 0 && y >= 0
            a(i) = atan(y/x);
        end

        if x >= 0 && y < 0
            a(i) = 2*pi + atan(y/x);
        end

        if x < 0 && y >= 0
            a(i) = pi + atan(y/x);
        end

        if x < 0 && y < 0
            a(i) = pi + atan(y/x);
        end
end
```

Listado A.2: Calcula el angulo entre 0º y 360º

```
function [a1,ap] = angle180(z,p)

y = z(2)+1.0-1.0;
m = p(2)+1.0-1.0;
a1 = angle(z);
ap = angle(p);

    if y < 0
        a1 = a1 - pi;
        if m >= 0
        ap = ap + pi;
        elseif m < 0
            ap = ap - pi;
        end
```

```
15      end
16  end
```

Listado A.3: Calcula el angulo entre 0º y 180º

```
1   %Inicio de la funcion Cmsh
2   clear all
3   fclose all;
4   %% DEFINICION PARAMETRICA DE LA GEOMETRIA
5   %%
6   %% VALORES GEOMETRICOS DEFINIDOS POR EL USUARIO
7   G = [       % UNIDADES EN (mm)
8       7.5;    % paso del muelle
9       5.0;    % diametro interior del tubo
10      1.2;    % espesor de alambre
11      ];

13  %% VALORES DISCRETIZACION DEFINIDOS POR EL USUARIO
14  E = [       % SIN UNIDADES
15      60;     % elementos por paso
16      1;      % pasos totales
17      ];

19  %% LONGITUD DE LA ZONA DE CONTACTO
20  V = [       % UNIDADES EN GRADOS
21      30.0;   % limite de contacto
22      45.0;   % limite de transicion
23      ];

25  %% NUMERO DE CARAS EN QUE SE VA A DIVIDIR EL DOMINIO
26  F  = 4;     %numero de subdominios
27  A  = [0,4]; %numero de subdominios limitados por j-aristas, j =
        (3,4,5,...)
28              %A = [numero subd con 3 arist, numero subd con 4 aristas,
                   ...]

30  %
        **********************************************************************%
31              % FIN DE ENTRADA DE PARAMETROS %
32  %
        **********************************************************************%

33  %%
34  %
        **********************************************************************%
```

```
35              % INICIO DEL PROGRAMA %%
36  %
        **************************************************************************%
37  p = G(1);
38  D = G(2);
39  e = G(3);
40  n = E(1);
41  N = E(2);
42  t = V(1);
43  v = V(2);
44  c  = 0.5*(D-e);
45  a  = 180*atan(p/(2*pi*c))/pi;
46  dp = p/n;
47  da = 360/n;
48  %% SECCION TRANSVERSAL
49  %  Requiere la definicion de G, E, V, F, A
50  jouname='a1.jou';
51  fopen(jouname,'a+');
52  fo = fopen(jouname,'w');
53  fprintf(fo,'vertex create coordinates %f 0 0\n',c);
54  fprintf(fo,'vertex create coordinates %f 0 0\n',0.5*D);
55  fprintf(fo,'vertex create coordinates %f 0 %f\n',c,0.5*e);
56  fprintf(fo,'face create center2points "vertex.1" "vertex.2" "vertex.3"
        circle\n');
57  fprintf(fo,'face move "face.1" dangle %.16f vector 1 0 0 origin 0 0 0\
       n',a);
58  fprintf(fo,'edge create revolve "vertex.1" height %f angle 720 vector
       0 0 1 origin 0 0 0\n',2*p);
59  fprintf(fo,'volume create rotate "face.1" onedge "edge.2" twist 0\n');
60  fprintf(fo,'volume create width %f depth %f height %f brick\n',2*D,2*D
       ,2*p);
61  fprintf(fo,'volume subtract "volume.1" volumes "volume.2"\n');
62  fprintf(fo,'face create wireframe "edge.17" real\n');
63  fprintf(fo,'volume delete "volume.1" lowertopology\n');
64  fprintf(fo,'face move "face.10" offset 0 0 %f\n',-p);
65  fprintf(fo,'edge delete "edge.2"\n');
66  fprintf(fo,'vertex delete "vertex.3" "vertex.4"\n');
67  fprintf(fo,'vertex create coordinates 0 0 0\n');
68  fprintf(fo,'vertex create coordinates 0 %f 0\n',0.5*D);
69  fprintf(fo,'face create center2points "vertex.15" "vertex.14" "vertex
       .16" circle\n');
70  fprintf(fo,'face subtract "face.11" faces "face.10"\n');
71  fprintf(fo,'coordinate create cylindrical oldsystem "c_sys.1" offset %
       f 0 0 axis1 "x" \\\n',0.5*c);
72  fprintf(fo,'angle1 0 axis2 "y" angle2 0 axis3 "z" angle3 0 rotation\n'
       );
```

```
73  fprintf(fo,'vertex create coordinates %f %f 0\n',0.6*c,t);
74  fprintf(fo,'vertex create coordinates %f %f 0\n',1.5*c,t);
75  fprintf(fo,'vertex create coordinates %f %f 0\n',0.6*c,-t);
76  fprintf(fo,'vertex create coordinates %f %f 0\n',1.5*c,-t);
77  fprintf(fo,'edge create straight "vertex.17" "vertex.18"\n');
78  fprintf(fo,'edge create straight "vertex.19" "vertex.20"\n');
79  fprintf(fo,'edge split "edge.19" tolerance 1e-6 edge "edge.20"
       keeptool connected\n');
80  fprintf(fo,'edge split "edge.18" tolerance 1e-6 edge "edge.20"
       connected\n');
81  fprintf(fo,'edge split "edge.19" tolerance 1e-6 edge "edge.21"
       keeptool connected\n');
82  fprintf(fo,'edge split "edge.24" tolerance 1e-6 edge "edge.21"
       connected\n');
83  fprintf(fo,'edge create straight "vertex.23" "vertex.24"\n');
84  fprintf(fo,'edge create straight "vertex.27" "vertex.28"\n');
85  fprintf(fo,'face create wireframe "edge.26" "edge.28" "edge.24" "edge
       .29" real\n');
86  fprintf(fo,'face delete "face.11" lowertopology\n');
87  fprintf(fo,'vertex delete "vertex.1" "vertex.16"\n');
88  fprintf(fo,'vertex create coordinates %f %f 0\n',0.6*c,v);
89  fprintf(fo,'vertex create coordinates %f %f 0\n',1.5*c,v);
90  fprintf(fo,'vertex create coordinates %f %f 0\n',0.6*c,-v);
91  fprintf(fo,'vertex create coordinates %f %f 0\n',1.5*c,-v);
92  fprintf(fo,'edge create straight "vertex.29" "vertex.30"\n');
93  fprintf(fo,'edge create straight "vertex.31" "vertex.32"\n');
94  fprintf(fo,'edge split "edge.26" tolerance 1e-06 edge "edge.30"
       keeptool connected\n');
95  fprintf(fo,'edge split "edge.24" tolerance 1e-06 edge "edge.30"
       connected\n');
96  fprintf(fo,'edge split "edge.26" tolerance 1e-06 edge "edge.31"
       keeptool connected\n');
97  fprintf(fo,'edge split "edge.34" tolerance 1e-06 edge "edge.31"
       connected\n');
98  fprintf(fo,'edge create straight "vertex.35" "vertex.36"\n');
99  fprintf(fo,'edge create straight "vertex.39" "vertex.40"\n');
100 fprintf(fo,'face split "face.12" edges "edge.38"\n');
101 fprintf(fo,'face split "face.12" edges "edge.39"\n');
102 fclose('all');
103 input('LOAD "a1.jou"')
104 input('1. Malla el dominio');
105 input('2. Renanme Faces: face.1 face.2 face.3 ...');
106 %%
107 %NUMERO DE CARAS LATERALES DE UN DELTA DE PASO
108 %parte de la hipotesis de que la generacion de caras sucesivas seha
109 %realizado partiendo sucesivamente el dominio de partida sin dejar
       ningun
```

```
110 %hueco en la numeracion de la caras
111 I0 = 14; %ultima cara numerada por Gambit al finalizar a1.jou
112 I1 = I0 + (F-3); %El num 3 es porque el minimo de subdominios es 3
113 F_lat = 0;
114 for i=1:length(A)
115     w = 3 + (i-1); %El 3 es porque A empieza en j = 3 aristas
116     F_lat = F_lat + w*A(i);
117 end
118 I2 = I1 + F + F_lat; %I1: donde se quedo, F + F_lat: las que se aaden
119                      %I2: donde se queda al hacer el twist
120 %% PRIMERA REBANADA:
121 %  Requiere la definicion del mallado del dominio
122 jouname='a2.jou';
123 fopen(jouname,'a+');
124 fo = fopen(jouname,'w');
125 fprintf(fo,'default set "GEOMETRY.VOLUME.SWEEP_PATH_ALIGNMENT" numeric
        0\n');
126 fprintf(fo,'volume create rotate \\\n');
127     for i=1:F
128     fprintf(fo,'"face.%d" \\\n',i);
129     end
130 fprintf(fo,'vector 0 0 %.14f origin 0 \\\n',dp);
131 fprintf(fo,'0 0 twist %.14f\n',da);
132 fprintf(fo,'edge smooth replacebad\n');
133 fprintf(fo,'face cmove \\\n');
134     for i = 1:F
135     fprintf(fo,'"face.%d" \\\n',i);
136     end
137 fprintf(fo,'multiple 1 unlinkmesh offset 0 0 %.14f\n',dp);
138 fprintf(fo,'face move \\\n');
139     for i=1:F
140     fprintf(fo,'"face.%d" \\\n',I2+i);
141     end
142 fprintf(fo,'dangle %.14f vector 0 0 1 origin 0 0 0\n',da);
143 fprintf(fo,'face connect\n');
144 fprintf(fo,'volume merge \\\n');
145     for i = 1:F
146     fprintf(fo,'"volume.%d" \\\n',i);
147     end
148 fprintf(fo,'real\n');
149 fprintf(fo,'physics create "INLET" btype "VELOCITY_INLET" face \\\n');
150     for i = 1:(F-1)
151         fprintf(fo,'"face.%d" \\\n',i);
152     end
153         fprintf(fo,'"face.%d"\n',F);
154 fprintf(fo,'physics create "OUTLET" btype "PRESSURE_OUTLET" face \\\n'
        );
```

```
155     for i = 1:F
156         fprintf(fo,'"face.%d" \\\n',I2+i);
157     end
158         fprintf(fo,'"face.%d"\n',I2+F);
159 fprintf(fo,'physics create "INTERIOR" ctype "FLUID" volume "volume.1"\
    n');
160 fclose('all');
161 input('LOAD "a2.jou"');
162 input('1. DEFINE B.C. BTYPE WALL... ');
163 %% GENERAR TODOS LOS VOLUMENES DEL TUBO:
164 %  Requiere la definicion de WALL
165 jouname='a3.jou';
166 fopen(jouname,'a+');
167 fo = fopen(jouname,'w');
168 vols = 1:F;
169 for i = 1:(n*N-1)
170     ndp = i*dp;
171     nda = i*da;

173 fprintf(fo,'volume cmove "volume.1" multiple 1 unlinkmesh copyzone
    offset 0 0 %.14f\n',ndp');
174 fprintf(fo,'volume move "volume.%d" dangle %.14f vector 0 0 1 origin 0
     0 0\n',i+1,nda);
175 fprintf(fo,'window modify volume "volume.%d" invisible\n',i+1);
176 end
177 fprintf(fo,'window modify volume "volume.%d" visible\n',n*N);
178 fclose('all');
179 input('LOAD a3.jou... ');
180 input('EXPORT FACES.TXT TO WORKSPACE:, 1 INLET & 2 OUTLET');
181 %% UNIR TODOS LOS VOLUMENES:
182 %  Requiere la entrada de usurario FACES.TXT
183 jouname='a4.jou';
184 fopen(jouname,'a+');
185 fo = fopen(jouname,'w');
186 read = fopen('faces.txt','r');
187 if read == 5
188 fclose(read);
189 replaceinfile('physics modify INLET btype ' , '' , 'faces');
190 replaceinfile('physics modify OUTLET btype ' , '' , 'faces');
191 replaceinfile('face.' , '' , 'faces');
192 replaceinfile(' \\' , '' , 'faces');
193 replaceinfile('  ' , '' , 'faces');
194 delete faces.txt.bak;
195 A = importdata('faces.txt');
196 B = A.';
197 b = ~isnan(B);
198 l = sum(b,1);
```

```
199 B = B(b).';
200 C = mat2cell(B,1,1)';

202 fioI=B(1:F);
203 B(1:F)=[];
204 fioO=B(length(B)-(F-1):length(B));
205 B(length(B)-(F-1):length(B))=[];
206 A1 = B(1:(length(B)/2)); %outlet
207 A2 = B((1+length(B)/2:length(B))); %inlet
208 fio = [A2;A1];

210 fprintf(fo,'face delete \\\n');
211 for i=1:length(A2)
212     fprintf(fo,'"face.%d" \\\n',A2(i));
213 end
214 fprintf(fo,'lowertopology onlymesh\n');

216 for k = 1:n*N-1
217     fprintf(fo,'face connect \\\n');
218     for j = 1:F
219         p1 = k*F+(j-F);
220         p2 = (k-1)*F-((j-1)-F);
221         fprintf(fo,'"face.%d" "face.%d" \\\n',fio(1,p2),fio(2,p1));
222     end
223     fprintf(fo,'real\n');
224 end

226 fprintf(fo,'physics modify "INLET" btype "VELOCITY_INLET" face \\\n');
227     for i = 1:(F-1)
228         fprintf(fo,'"face.%d" \\\n',i);
229     end
230         fprintf(fo,'"face.%d"\n',F);

232 fprintf(fo,'volume mesh "volume.1" cooper source \\\n');
233     for i=1:F
234         fprintf(fo,'"face.%d" "face.%d" \\\n',i,A1(i));
235     end
236         fprintf(fo,'size 1\n');

238 fio(1,:) = [];
239 for k = 2:n*N-1
240     fprintf(fo,'volume mesh "volume.%d" cooper source \\\n',k);
241     for j = 1:F
242         p1 = (k-1)*F+(j-F);
243         p2 = p1+F;
244         fprintf(fo,'"face.%d" "face.%d" \\\n',fio(p1),fio(p2));
245     end
```

```
246         fprintf(fo,'size 1\n');
247         fprintf(fo,'window modify volume "volume.%d" invisible\n',k);
248 end
249 fprintf(fo,'volume mesh "volume.%d" cooper source \\\n',n*N);
250     for i=1:F
251         fprintf(fo,'"face.%d" "face.%d" \\\n',fio(length(fio)-(i-1)),
              fio0(i));
252     end
253         fprintf(fo,'size 1\n');
254 disp('LOAD a4.jou')
255 else
256 disp('Error: did not find the file "faces.txt", please import it to
    workspace');
257 end
258 disp('REMEMBER IT: the mesh was created in milimeters!!!')

260 fclose('all');
```

Listado A.4: Genera journals para compilar en Gambit

```
1  function vertex = coordelem(element,coord)
2  %Funcion que calcula las coordenadas de los vertices de un elemento
3  %
4  %input:     element: vector fila con los datos coorespondientes de
       listelem
5  %           coord:   listnode de la funcion loadnode
6  %
7  %outpu:     vertex:  matrix with 2 cols and element(2) rows

9  vertex = zeros(element(2),2);

11 for i = 1:element(2)
12     vertex(i,[1 2]) = searchnode(element(i+2),coord);
13 end
```

Listado A.5: Calcula las coordenadas de los vertices de un elemento

```
1  function elements = loadelem(file)
2  %Funcion que carga los datos del fichero ascii con las coordenadas de
       los
3  %nodos de la malla
4  %
5  % INPUT:
6  % ------
7  % 'nombre_del_fichero_que queremos_leer' (escrito entre '...')
8  %
9  % OUTPUT:
```

```
% --------
%
% c1: datos de la primera columna
% c2: datos de la segunda columna
% c3: etc...
%
% FIN

replaceinfile('  Element    Type    Count         Connectivity','',file)
   ;
replaceinfile(' --------- --------- ---------
   -------------------------------','',file);
replaceinfile('quad','',file);
replaceinfile('triangle','',file);
replaceinfile(':','',file);
strbak = sprintf('%s.bak',file);
delete(strbak);

E = importdata(file);
[f c] = size(E);

for j = 1:c
    for i = 1:f
        if ~isnan(E(i,j)) == 0
            E(i,j) = 0;
        end
    end
end
                                        %   d --<-- c
for i = 1:length(E)                     %   |       |
    numnodes = E(i,2);                  %   \/      ^
    E(i,numnodes+3) = E(i,3);           %   |       |
end                                     %   a -->-- b

%Ordenar la numeracion de los elementos
eord = sort(E(:,1));
for i = 1:length(eord)
    for j = 1:length(eord)
       if eord(i) == E(j,1)
           E(i,:) = E(j,:);
       end
    end
end
elements = E;
```

Listado A.6: Carga los datos del fichero ascii con las coordenadas de los nodos de la malla

```
function Fluentnodes = loadFS(secciondata)
%Funcion que carga los datos del fichero ascii con toda la informacion de
%la seccion transversal exportados desde Fluent y separados por un espacio.
%
% INPUT:
% ------
% 'nombre_del_fichero_que queremos_leer' (escrito entre '...')
%
% OUTPUT:
% --------
%
% Cf = [col_1, col_2, col_3 ,...]
%
%       col_1: datos de la primera columna
%       col_2: datos de la segunda columna
%       col_3: etc...
%
% FIN

replaceinfile('x=','x',secciondata);
replaceinfile('y=','y',secciondata);
replaceinfile('z=','z',secciondata);
strbak = sprintf('%s.bak',secciondata);
delete(strbak);

M = importdata(secciondata);
Fluentnodes = M.data(:,1:7);
```

Listado A.7: Carga los datos del fichero ascii con toda la informacion de la seccion transversal exportados desde Fluent y separados por un espacio

```
function Gambitnodes = loadGN(file)
%Funcion que carga los datos de Gambit con las coordenadas de los nodos.
%
% INPUT:
% ------
% 'nombre_del_fichero_que queremos_leer' (escrito entre '...')
%
% OUTPUT:
% --------
%
% Cg = [col_1, col_2, col_3]
%
%       col_1: datos de la primera columna
```

```
%       col_2: datos de la segunda columna
%       col_3: datos de la tercera columna
%
% FIN

replaceinfile('   Node        x            y            z
   Owner','',file);
replaceinfile(' ------- ------------ ------------ ------------
   ------------------','',file);
replaceinfile('vertex.','',file);
replaceinfile('edge.','',file);
replaceinfile('face.','',file);
strbak = sprintf('%s.bak',file);
delete(strbak);

M = importdata(file);
Gambitnodes = M(:,(1:3));
end
```

Listado A.8: Carga los datos de Gambit con las coordenadas de los nodos

```
function [nodes, f2g] = loadnode(Gnodes,Fnodes,scaleF2G,giro)
%Funcion que carga los datos de la Geometria de Gambit con la
   precision de
%FLuent y devuelve la lista f2g de sustitucion de la numeracion de los
%nodos de Fluent por los de Gambit. La escala es para adaptar las
   unidades
%de longitud de Fluent a mm, que es con lo que se esta trabajando en
   Gambit

Cg = loadGN(Gnodes);
Cf = loadFS(Fnodes);

%Paso de metros a milimetros
if nargin >= 3
    fc = size(scaleF2G);
    for i = 1:fc(1)
        Cf(:,scaleF2G(i,1)) = scaleF2G(i,2)*Cf(:,scaleF2G(i,1));
    end
end
Cf(:,[2 3 4]) = Rotz(-giro,Cf(:,[2 3 4]));
Cf(:,[5 6 7]) = Rotz(-giro,Cf(:,[5 6 7]));
%Rename
tol = 1e-4;
fc = size(Cf);
%lista de remplazo de nodos para machacar
   Identifier_Files_Fluent_Nodes
```

```
23  f2g = zeros(fc(1),1);
24  true = 0;
25  for i = 1:fc(1)
26      for j = 1:length(Cg)

28          res1 = abs(Cf(i,2)-Cg(j,2));
29          res2 = abs(Cf(i,3)-Cg(j,3));

31          if res1 <= tol && res2 <= tol
32              true = 1;
33              f2g(i) = Cg(j,1);
34              Cf(i,1)  = Cg(j,1);

36          end
37      end
38  end

40  n  = sort(Cf(:,1));
41  Cn = zeros(length(n),fc(2));
42  for i = 1:length(n)
43      for j = 1:length(n)

45          if n(i) == Cf(j,1)
46              Cn(i,:) = Cf(j,:);
47          end
48      end
49  end

51  if true == 0
52      disp('Error: no se ha encontrado ninguna coincidencia entre Cg y
            Cf')
53      return
54  end

56  nodes = Cn;
57  end
```

Listado A.9: Carga los datos de la geometria de Gambit

```
1  function StepList = meshcicles(ymin,partes,ciclos,REo,ST,prop)
2  %Funcion que hace un mshadapt entre los puntos de control
       especificados en
3  %el vector ti = [t0, t1, t2, ...] y devuelve una lista con:
4  %
5  %StepList(:,1): paso dentro de un ciclo al que pertenece ese instante
6  %StepList(:,2): instantes de tiempo donde se evalua la funcion
7  %StepList(:,3): evaluacion de la funcion
```

```
%generacion del vector ti, optimizado para funciones armonicas
DATA = OSCflow(REo,ST,prop);
f    = DATA.FRECUENCY;
mf   = DATA.MASSFLOW;
function y = FUN(t)
ft = length(t);
y  = zeros(ft,1);

for k = 1:ft
    y(k) = mf*cos(t(k));
end
end



ti = 0:2*pi/partes:2*pi*ciclos;

t = NaN;
for i = 1:length(ti)-1
    tinter = meshadapt(ymin, ti(i), ti(i+1), mf);
    t = [t tinter']; %realmente no es tiempo es angulo
end
t(1) = [];

for i = length(t):-1:2
    if t(i) == t(i-1)
        t(i) = [];
    end
end

t(length(t)) = [];
for i = 1:length(t)
    if t(i) == 2*pi
        Fluent_Total_Saves = i-1;
        break
    end
end

position = zeros(Fluent_Total_Saves*ciclos,1);
j_pos = 1;
for i = 1:ciclos
    for j = 1:Fluent_Total_Saves
        position(j_pos) = j;
        j_pos = j_pos + 1;
    end
end
```

```
58  y = FUN(t);
59  t = t/2/pi/f;

61  StepList = [position t' y];


64  z = zeros(length(t)-1,1);
65  for i = 1:length(t)-1
66  z(i) = t(i+1)-t(i);
67  end
68  end

70  function t = meshadapt(ymin, t0, t1, mf)
71  %Funcion que genera un mallado para la variable temporal atendiendo al
72  %nivel de discretizacion kmin utilizado para el intervalo t0, t1.
73  %
74  %El mallado en t tiene que ser tal que cumpla la condicion:
75  %
76  %                    y_i+1 - y_i = k
77  %
78  %donde kmin es una constante; normalmente definida como:
79  %
80  %                    kmin = (y_max - y_min) / nel
81  %
82  %donde nel es el numero de divisiones o el numero de elementos del
       mallado
83  %y donde y_max e y_min se alcanzan en los extremos [t0, t1] del
       dominio y
84  %la derivada siempre es:
85  %
86  %                    |y'| > 0, para todo x  [t0, t1]
87  %
88  %Las comprobaciones acerca del comportamiento de la funcion "y", debe
89  %realizaras el usuario, ya que el programa no las tiene en cuenta.

91  %Para este caso concreto, la funcion que se usa es:

93  %y = sin(), y-funcion
94  %Y = cos(), Y-derivada

96  %Buscar y sustituir para cada caso

98  function y = FUN(t)
99  ft = length(t);
```

```
100 y  = zeros(ft,1);
101 for k = 1:ft
102     y(k) = mf*cos(t(k));
103 end
104 end

106 function kmin = KMIN(ymin)
107 kmin  = mf/ymin;
108 end

110 kmin  = KMIN(ymin);
111 tol   = 1e-12;
112 apr   = 1000;
113 nel   = ceil(abs(FUN(t1) - FUN(t0))/kmin);
114 dt    = (t1 - t0)/apr;

116 kmin  = abs(FUN(t1) - FUN(t0))/nel;

118 t = zeros(nel+1,1);
119 t(1)     = t0;
120 t(nel+1) = t1;

122 for i = 1:nel-1
123     tr = t(i);
124     while abs(FUN(tr) - FUN(t(i))) - kmin < 0
125         tr = tr + dt;
126     end
127     ti = tr - dt;
128     tj = tr;

130     tm = 0.5*(ti + tj);
131     fm = abs(FUN(tm) - FUN(t(i))) - kmin;
132     fi = abs(FUN(ti) - FUN(t(i))) - kmin;
133     fj = abs(FUN(tj) - FUN(t(i))) - kmin;
134     while abs(fm) > tol
135         if fi*fm < 0
136             tj = tm;
137         end
138         if fj*fm < 0
139             ti = tm;
140         end
141         tm = 0.5*(ti + tj);
142         fm = abs(FUN(tm) - FUN(t(i))) - kmin;
143         fi = abs(FUN(ti) - FUN(t(i))) - kmin;
144         fj = abs(FUN(tj) - FUN(t(i))) - kmin;
145     end
146     t(i+1) = tm;
```

```
147 end
148 end
```

Listado A.10: Hace un mshadapt entre los puntos de control

```
1  function xpdat =  nodinterpole(ndats,N)
2  fcn = size(ndats);
3  fcN = size(N);
4    if fcn(1) == fcN(2)
5        xpdat = zeros(1,fcn(2));
6        for i = 1:fcn(2)
7            idat = ndats(:,i);
8            xpdat(i) = N*idat;
9        end
10   else
11       disp('ERROR en la dimension de los datos')
12       return
13   end
14 end
```

Listado A.11: Interpola entre los nodos de un elemento

```
1  function DATA = OSCflow(REo,ST,prop,Wall_fz_std,Wall_fz_osc)
2  %Programa que calcula todas las magnitudes fluidas para regimen
       oscilatorio
3  %INPUT:
4  %       REo  : Reynolds oscilatorio, (ver paper)
5  %       ST   : Strouhal, (ver paper)
6  %       prop : tipo de fluido
7  %                              1: agua
8  %                              2: glicerina
9  %                              3: aire
10 %
11 %       Wall_fz_std: Sumatorio de fuerzas en z para steady-flow
12 %       Wall_fz_osc: Sumatorio de fuerzas en z para oscillatory-flow
13 %
14 %OUTPUT:
15 %       DATA
16 %
17 %NOTAS:
18 %
19 %   Comprobar la definicion del Womersley porque el factor 2pi puede
       que no
20 %   sea necesario incluirlo, todo debe a como este definido REo y ST
21 %% SETUP
22 Area          = 17.453628*1e-6; %area      seccion transversal (Gambit,
       m)
```

```
23 Peri          = 18.196270*1e-3; %perimetro seccion transversal (Gambit,
       m)
24 Lenght    = 7.5e-3;
25 Dh        = 4*Area/Peri; %hidraulic diameter mm
26 property = [998.2, 0.001003;     %1- propiedades agua
27             1259.9, 0.799;       %2- propiedades glicerina
28             1.225, 1.7894e-5];   %3- propiedades aire
29 %% FUNCTION
30 if nargin < 4
31 NETFORCE_std = 0;
32 else NETFORCE_std = Wall_fz_std;
33 end
34 if nargin < 5
35 NETFORCE_osc = 0;
36 else NETFORCE_osc = Wall_fz_osc;
37 end

39 switch prop
40     case 1
41         fluid = 'water';
42     case 2
43         fluid = 'glyc';
44     case 3
45         fluid = 'air';
46 end

48 %nu  = property(prop,2)/property(prop,1);

50 %INITIALITE
51 %Oscillatory Reynolds: [1 10 80 300 1000]
52 REn    = REo; %condicion de oscilacion despues de flujo nominal
       estacionario
53 Vn     = REn*property(prop,2)/property(prop,1)/Dh;
54 mf     = Vn*Area*property(prop,1);
55 caud   = Vn*Area;

57 %OSCILLATORI
58 Vo     = REo*property(prop,2)/property(prop,1)/Dh;
59 f      = 2*Vo*ST/Dh;
60 T      = Dh/2/Vo/ST;
61 x0     = Vo/2/pi/f;
62 WOMERSLEY = sqrt(2*pi*REo*ST);
63 pl         = 128*property(prop,2)*caud/pi/Dh^4; %(pa/m) Pl tubo liso
       lam
64 PLstd      = NETFORCE_std/Area/Lenght;          %(pa/m) Pl tubo rugoso
       std
65 PLosc      = NETFORCE_osc/Area/Lenght;          %(pa/m) Pl tubo rugoso
```

```
    osc

DATA = struct(...
    'REYNOLDS',REo,'STROUHAL',ST,'WOMERSLEY',WOMERSLEY,...
    'FRECUENCY',f,'TIMECICLE',T,'VELOCITY',Vo,'MASSFLOW',mf,...
    'PLUNGER',x0,'PLsmh',pl,'PLstd',PLstd,'PLosc',PLosc);

filename = sprintf('OSCDAT_RE%.0f_ST%.1f_%s',REo,ST,fluid);
fopen(filename,'a+');
fo = fopen(filename,'w');
fprintf(fo,'REYNOLDS:\t%.1f (-)\n',REo);
fprintf(fo,'STROUHAL:\t%.2f (-)\n',ST);
fprintf(fo,'WOMERSLEY:\t%.6f (-)\n',WOMERSLEY);
fprintf(fo,'FLUID:\t\t%s\n',fluid);
fprintf(fo,'DENSITY:\t%.2f (kg/m3)\n',property(prop,1));
fprintf(fo,'VISCOSITY:\t%.6f (kg/m-s)\n',property(prop,2));
fprintf(fo,'FRECUENCY:\t%.6f (Hz)\n',f);
fprintf(fo,'TIMECICLE:\t%.6f (s)\n',T);
fprintf(fo,'VELOCITY:\t%.12f (m/s)\n',Vo);
fprintf(fo,'MASSFLOW:\t%.12f (mg/s)\n',1000*mf);
fprintf(fo,'PLUNGER:\t%.12f (mm/s)\n',1000*x0);
fprintf(fo,'PRESSURE DROP SMH:\t%.6f (Pa/m)\n',pl);
fprintf(fo,'PRESSURE DROP STD:\t%.6f (Pa/m)\n',PLstd);
fprintf(fo,'PRESSURE DROP OSC:\t%.6f (Pa/m)',PLosc);

fclose all;
end
```

Listado A.12: Calcula todas las magnitudes fluidas para regimen oscilatorio

```
function [XP XR] = PATHLINES2(Step_Size,Total_Steps,REo,ST,prop)
%Programa que calcula la trayectoria, el tiempo y la longitud
    recorrida por
%una particula dentro de una geometria con simetria helicoidal a
    traves del
%campo de velocidades no estacionario en una seccion transversal
    cualquiera
%
%       RESUMEN DE ESCRITURA EN FICHERO ASCII
%
    ------------------------------------------------------------------
%       X     Y     Z     Vx     Vy     Vz     S     T      Id
%       ...   ...   ...   ...    ...    ...    ...   ...    ...
%
%   X :  x-coordinate
%   Y :  y-coordinate
%   Z :  z-coordinate
```

```
14 %   Vx:  x-velocity
15 %   Vy:  y-velocity
16 %   Vz:  z-velocity
17 %   S :  instantaneus distance
18 %   T :  instantaneus time
19 %   Id:  particle identifier

21 % BUGS: No quedan

23     %% SETTINGS %%

25 Cicle_Fluent_Dats             =  1; %Ciclo del que se extrajeron los
       datos
26 Unit_Scales_Length_Fluent     = 'm';
27 Name_File_Gambit_Elements     = 'elements';
28 Name_File_Gambit_Coordinates  = 'ncoord';
29 Name_File_Coordinates_Id      = 'pointsY2';
30 Id_Files_Fluent_No_Standard   = 'n'; %VER FINAL DE PAGINA
31 Identifier_Files_Fluent_Nodes = 'Vfz3_RE1'; %(Vnode1)nombre si
       Id_Fluent = NO
32 Angle_Betwen_Cg_and_Cf        =  180; %0 si no estan giradas
33 Transient                                  = 'n';
34 Step_Coil                     =  7.5; %mm
35 Reference_Plane_Z0            =  0.0;
36 TXT_OUT_PUT                   = 'RHINO'; %formato salida texto (o
       Samuel)
37 Z_RHINO                       =  7.5;
38 Columns_Dats                  =  9; %Numero de columnas exportadas a
       ascii
39 Y_Minimun                     =  15;  %numero de salto minimo en un
       trozo
40 Cicle_Division                =  8;   %trozos en que se parte un ciclo
41 Cicles_Calculated             =  2; %tamao de lista de ciclos

43     %% NOTES %%

45 % 0. VALIDO UNICAMENTE PARA GEOMETRIAS CILINDRICAS CON SIMETRIA
       HELICOIDAL
46 %
47 % 1. EXPORTAR LOS DATOS DE VELOCIDAD EN LOS NODOS CON UNA NUMERACION
48 %    CONSECUTIA Y EMPEZANDO EN UNO, ver el ejemplo Vnode
49 %
50 % 2. PARA CORRER UN ESTACIONARIO ES SUFICIENTE CON NOMBRAR AL ARCHIVO
       DE
51 %    FLUENT CON EL NOMBRE DEL IDENTIFICADOR
52 %
53 % 3. VERIFICAR QUE EL PRIMER PUNTO Y EL ULTIMO ESTAN EN FASE
```

```
54 %    (separados en el tiempo por T = 1/f s)
55 %
56 % 4. UNA SIMULACION SOLO PUEDE EMPEZAR AL TIEMPO DEL INICIO DE CICLO
57 %
58 % 5. LA GEOMETRIA SE DEFINE SOLO CON LAS VARIABLES (X,Y), ver scaleF2G
       (3,2)
59 %
60 % 6. LA GEOMETRIA EN GAMBIT SE REALIZA EN mm EL ESCALADO ES PARA
       ADAPTAR
61 %    LAS UNIDADES DE FLUENT A GAMBIT
62 %
63 % 7. CUANDO SE SUBDIVIDE EL TIME STEP ES PARA INTENTAR QUE LA SOLUCION
64 %    SALTE NECESARIAMENTE DE UN PASO DE TIEMPO A OTRO
65 %
66 % 8. EL ERROR 001 SE DEBE A MULTIPLES CAUSAS, reducir por orden:
67 %       a) Step_Size
68 %       b) Fluent_Time_Step
69 %       c) Boundary_Layer
70 %
71 % 9. USA Ctrl + C PARA DETENER LA SIMULACION EN CASO DE SER NECESARIO

73     %% CODE %%
74 disp('LOADING NODES...')
75 pause(0.1)
76 StepList        = meshcicles(Y_Minimun,Cicle_Division,
       Cicles_Calculated,REo,ST,prop);
77 giro            = Angle_Betwen_Cg_and_Cf;
78 nstep           = Total_Steps;
79 step_size       = Step_Size;
80 pelem           = 1;
81 fcSL            = size(StepList);
82 NameList        = StepList;
83 Cicle_Steps     = StepList(fcSL(1),1);
84 Time_Periodic   = StepList(Cicle_Steps,2);
85 Total_Cicles    = fcSL(1)/Cicle_Steps;
86 iniSL           = (Cicle_Fluent_Dats-1)*Cicle_Steps + 1;
87 finSL           = iniSL + Cicle_Steps - 1;
88 Name_Cicles     = StepList(iniSL:finSL,1);
89 Time_Cicles     = StepList(iniSL:finSL,2);
90 pos             = Name_Cicles;
91 for i = 1:Total_Cicles
92     NameList(pos,1) = Name_Cicles;
93     NameList(pos,2) = Time_Cicles;
94     pos = pos + Cicle_Steps;
95 end

97 % Rescale Units
```

```
98  m  = 1000;
99  dm = 100;
100 cm = 10;
101 mm = 1;
102 switch Unit_Scales_Length_Fluent
103     case {'m'}
104         factor = m;
105     case {'dm'}
106         factor = dm;
107     case {'cm'}
108         factor = cm;
109     case {'mm'}
110         factor = mm;
111     otherwise
112         disp('ERROR Settings: Unknown scale')
113         return
114 end

116 scaleF2G = [2 factor;
117             3 factor;
118             4 0];
119 switch Transient
120         case {'YES', 'Yes', 'yes', 'y'}
121         listelem  = loadelem(Name_File_Gambit_Elements);
122         if Id_Files_Fluent_No_Standard == 'y'
123             strFNOD = printname(1,NameList);
124             if strFNOD == -1
125                 return
126             end
127         else
128         strFNOD   = sprintf('%s',Identifier_Files_Fluent_Nodes);
129         end
130         [listnode f2g]  = loadnode(Name_File_Gambit_Coordinates,
               strFNOD,scaleF2G,giro);
131         if Id_Files_Fluent_No_Standard == 'y'
132             strTj = printname(2,NameList);
133             if strTj == -1
134                 return
135             end
136             strTi = printname(1,NameList);
137             if strTi == -1
138                 return
139             end
140         else
141                 strTj = sprintf('%s',Identifier_Files_Fluent_Nodes);
142                 strTi = sprintf('%s',Identifier_Files_Fluent_Nodes);
143         end
```

```
144                 %no se puede importar directamente desde fluent hay
                        renombrar a
145                 %gambit
146                 Mj    = importdata(strTj);
147                 Mi    = importdata(strTi);
148         Vj    = renameFN(f2g,Mj.data(:,[5 6 7]),factor);
149         Vj    = Rotz(-giro,Vj);
150         Vi    = renameFN(f2g,Mi.data(:,[5 6 7]),factor);
151         Vi    = Rotz(-giro,Vi);
152         case {'NO', 'No', 'no','n'}
153         Fluent_Time_Step   = 1e6;
154         Fluent_Total_Saves = 2;
155         StepList  = zeros(ceil(Cicle_Steps/Fluent_Total_Saves),2); j =
                1;
156         for i = 1:Fluent_Total_Saves
157             StepList(j,:) = [i, Fluent_Time_Step];
158             j = j + 1;
159         end
160         listelem  = loadelem(Name_File_Gambit_Elements);
161         if Id_Files_Fluent_No_Standard == 'y'
162             strFNOD = printname(1,NameList);
163             if strFNOD == -1
164                 return
165             end
166         else
167         strFNOD   = sprintf('%s',Identifier_Files_Fluent_Nodes);
168         end
169         [listnode f2g]  = loadnode(Name_File_Gambit_Coordinates,
               strFNOD,scaleF2G,giro);
170         if Id_Files_Fluent_No_Standard == 'y'
171             strTi = printname(1,NameList);
172             if strTi == -1
173                 return
174             end
175         else
176                 strTi = sprintf('%s',Identifier_Files_Fluent_Nodes);
177         end
178                 Mi    = importdata(strTi);
179                 Vi    = renameFN(f2g,Mi.data(:,[5 6 7]),factor);
180         Vi    = Rotz(-giro,Vi);

182     otherwise
183                 disp('ERROR Settings: Transient')
184 end

186 Particles   = importdata(Name_File_Coordinates_Id);
187 fcpart      = size(Particles);
```

```
188 XP          = zeros((nstep+1)*fcpart(1),Columns_Dats);
189 Xpointer    = zeros(nstep+1,1);
190 pointer     = 1;
191 pointxt     = 1;
192 for m = 1:fcpart(1)
193 strdisp = sprintf('START SIMULATION OF ID PARTICLE %d',Particles(m,1))
    ;
194     disp(strdisp)
195     Time_Count      = 1;
196     CURRENT_TIME    = 0.0;
197     LENGTH_S        = 0.0;
198     xp              = zeros(nstep+1, Columns_Dats);
199     xp(1,1:3)       = [Particles(m,2), Particles(m,3), Particles(m,4)
        ];
200     xp(1,7)         = LENGTH_S;
201     xp(1,8)         = CURRENT_TIME;
202     xp(:,9)         = Particles(m,1);
203     xpw             = RHC(xp(1,1:3));

205     cpuini              = cputime;
206     Xposition           = 1;
207     Xpointer(Xposition) = pointer;
208 for i = 2:nstep+1
209     pointer             = pointer  + 1;
210     Xposition           = Xposition + 1;
211     Xpointer(Xposition) = pointer;
212     pelem   = whoelem(xpw([1 2]),listelem,listnode,pelem);
213     if pelem == 0
214         return
215     end
216     element = listelem(pelem,:);
217     xe      = coordelem(element,listnode); %%INTERPOLAR T
218     N       = weightfun(xe,xpw([1 2]));
219     nodes   = element(3:(2+element(2)));%nodos del elemento pelem

221     switch Transient
222         case {'YES', 'Yes', 'yes', 'y'}
223             Vnodesj = Vj(nodes,:); %carga los datos de Ftime_i
224             Vnodesi = Vi(nodes,:);
225             Vpj     = nodinterpole(Vnodesj,N);
226             Vpi     = nodinterpole(Vnodesi,N);
227             xtime   = (CURRENT_TIME - StepList(Time_Count,2))/(
                    StepList(Time_Count+1,2) - StepList(Time_Count,2));
228             Vpt     = Vpj*xtime + (1 - xtime)*Vpi;
229         case {'NO', 'No', 'no', 'n'}
230             Vnodesi = Vi(nodes,:);
231             Vpt     = nodinterpole(Vnodesi,N);
```

```
232     end
233     Vmod          = sqrt(Vpt(1)^2+Vpt(2)^2+Vpt(3)^2);

235     %COMPROBACION DEL tstep ESCOGIDO
236     tstep          = step_size/Vmod;
237     CURRENT_TIME   = tstep + CURRENT_TIME;
238     %LOAD FOLLOWING FLUENT TIME STEP
239     if CURRENT_TIME >= StepList(Time_Count+1,2)
240         LAST_TIME     = CURRENT_TIME - tstep;
241         CURRENT_TIME  = StepList(Time_Count+1,2);
242         tstep         = CURRENT_TIME - LAST_TIME;
243         Time_Count    = Time_Count + 1;
244         if Id_Files_Fluent_No_Standard == 'y'
245             strTj = printname(Time_Count+1,NameList);
246             if strTj == -1
247                 return
248             end
249             strTi = printname(Time_Count,NameList);
250             if strTi == -1
251                 return
252             end
253         else
254                 strTj = sprintf('%s',Identifier_Files_Fluent_Nodes);
255                 strTi = sprintf('%s',Identifier_Files_Fluent_Nodes);
256         end
257         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%disp('strTj')
258         %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%disp(strTj)
259         Mj     = importdata(strTj);
260         Mi     = importdata(strTi);
261         Vj     = renameFN(f2g,Mj.data(:,[5 6 7]),factor);
262         Vj     = Rotz(-giro,Vj);
263         Vi     = renameFN(f2g,Mi.data(:,[5 6 7]),factor);
264         Vi     = Rotz(-giro,Vi);

266     end

268     Dx               = tstep*Vpt;
269     ang              = 360*(xp(i-1,3)-Reference_Plane_Z0)/Step_Coil;
270     Dx               = Rotz(ang,Dx); %Rotar vectores de velocidad a su
            plano
271     xp(i,1:3)        = Dx + xp((i-1),1:3);
272     xpw              = RHC(xp(i,1:3));
273     LENGTH_S         = LENGTH_S + tstep*Vmod;
274     xp(i,7)          = LENGTH_S;
275     xp(i,8)          = CURRENT_TIME;
276     xp(i-1,[4 5 6]) = Vpt;
```

```
278 end
279 pointer = pointer + 1;
280 timecpu = cputime - cpuini;
281 strdisp = sprintf('SIMULATION TIME: %.8f seconds',CURRENT_TIME);
282 disp(strdisp)
283 strdisp = sprintf('CPU TIME:        %.8f seconds',timecpu);
284 disp(strdisp)
285 XP(Xpointer,:) = xp;

287 %Postprocessing
288 switch TXT_OUT_PUT
289     case {'Samuel'}
290 disp('Writing a file XPtracks.txt')
291 fcXP = size(XP);
292 fid = fopen('XPtracks.txt','a');
293 ff = sprintf('PARTICLES TRACKS: Samuel Espin Tolosa 2012\n');
294 fwrite(fid,ff);
295 ff = sprintf('-----------------------------------------\n');
296 fwrite(fid,ff);
297 ff = sprintf('\tX\t\t\tY\t\t\tZ\t\t\tVx\t\t\tVy\t\t\tVz\t\t\tS\t\t\tT\
    t\tId\n');
298 fwrite(fid,ff);
299 for i = 1:fcXP(1)
300     ff = sprintf('%f\t%f\t%f\t',XP(i,1),XP(i,2),XP(i,3));
301     fwrite(fid,ff);
302     ff = sprintf('%f\t%f\t%f\t',XP(i,4),XP(i,5),XP(i,6));
303     fwrite(fid,ff);
304     ff = sprintf('%f\t%f\t%d\n',XP(i,7),XP(i,8),XP(i,9));
305     fwrite(fid,ff);
306 end

308     case {'RHINO'}
309         strtxt = sprintf('RE%d_XP%d.txt',REo,Particles(m,1));
310         fid = fopen(strtxt,'a');
311 for i = 1:nstep+1
312     if XP(pointxt,3) <= Z_RHINO
313         if XP(pointxt,8) <= Time_Periodic
314         ff = sprintf('%f\t%f\t%f\n',XP(pointxt,1),XP(pointxt,2),XP(
               pointxt,3));
315         fwrite(fid,ff);
316         end
317     end
318     pointxt = pointxt + 1;
319 end
320 end
321 end
322 XR = abstorel(XP);
```

```
323 fclose all;
324 end

326 %% DEFINIR EL NOMBRE QUE TIENEN NUESTROS ARCHIVOS DE DATOS DE FLUENT
327 function strname = printname(Time_Count,NameList)
328 if Time_Count > length(NameList)
329     disp('ERROR StepList: pocos ciclos calculados')
330     strname = -1;
331 else
332 strname = sprintf('Vf_time_%f_step_%d',NameList(Time_Count,2),NameList
    (Time_Count,1));
333 end
334 end
```

Listado A.13: Calcula la trayectoria, tiempo y longitud

```
1  function renamenod = renameFN(f2g,datanod,scale)

3  if nargin == 3
4      datanod = datanod*scale;
5  end
6  fc = size(datanod);
7  stack = zeros(fc(1),fc(2));
8  for i = 1:fc(1)
9      stack(f2g(i),:) = datanod(i,:);

11 end
12 renamenod = stack;
13 end
```

Listado A.14: Renombra la lista de nodos de fluent

```
1  function [s, msg] = replaceinfile(str1, str2, infile, outfile)
2  %REPLACEINFILE replaces characters in ASCII file using PERL
3  %
4  % [s, msg] = replaceinfile(str1, str2, infile)
5  %    replaces str1 with str2 in infile, original file is saved as "
       infile.bak"
6  %
7  % [s, msg] = replaceinfile(str1, str2, infile, outfile)
8  %    writes contents of infile to outfile, str1 replaced with str2
9  %    NOTE! if outputfile is '-nobak' the backup file will be deleted
10 %
11 % [s, msg] = replaceinfile(str1, str2)
12 %    opens gui for the infile, replaces str1 with str2 in infile,
       original file is saved as "infile.bak"
13 %
```

```
14 % in:  str1      string to be replaced
15 %      str2      string to replace with
16 %      infile    file to search in
17 %      outfile   outputfile (optional) if '-nobak'
18 %
19 % out: s         status information, 0 if succesful
20 %      msg       messages from calling PERL

22 % Pekka Kumpulainen 30.08.2000
23 % 16.11.2008 fixed for paths having whitespaces,
24 % 16.11.2008 dos rename replaced by "movefile" to force overwrite
25 % 08.01.2009 '-nobak' option to remove backup file, fixed help a
      little..
26 %
27 % TAMPERE UNIVERSITY OF TECHNOLOGY
28 % Measurement and Information Technology
29 % www.mit.tut.fi

31 infile = sprintf('%s\\%s.txt',pwd,infile);
32 message = nargchk(2,4,nargin);
33 if ~isempty(message)
34     error(message)
35 end

37 %% check inputs
38 if ~(ischar(str1) && ischar(str2))
39     error('Invalid string arguments.')
40 end
41 % in case of single characters, escape special characters
42 % (at least someof them)
43 switch str1
44     case {'\' '.'}
45         str1 = ['\' str1];
46 end

48 %% uigetfile if none given
49 if nargin < 3;
50     [fn, fpath] = uigetfile('*.*','Select file');
51     if ~ischar(fn)
52         return
53     end
54     infile = fullfile(fpath,fn);
55 end

57 %% The PERL stuff
58 perlCmd = sprintf('"%s"',fullfile(matlabroot, 'sys\perl\win32\bin\perl
      '));
```

```
perlstr = sprintf('%s -i.bak -pe"s/%s/%s/g" "%s"', perlCmd, str1, str2
    ,infile);

[s,msg] = dos(perlstr);

%% rename files if outputfile given
if ~isempty(msg)
    error(msg)
else
    if nargin > 3 % rename files
        if strcmp('-nobak',outfile)
            delete(sprintf('%s.bak',infile));
        else
            movefile(infile, outfile);
            movefile(sprintf('%s.bak',infile), infile);
        end
    end
end
```

Listado A.15: Replaces characters in ASCII file using PERL

```
function xr = RHC(xp)
%Funcion que toma las coordenadas x,y,z de una particula respecto de
    un
%plano de referencia z0 = 0 situado a la distancia "offset" del origen
    de
%coordendas y calcula su proyeccion helicoidal respecto de dicho plano
    z0:
%
% in:       xp = [x y z]
%           x,      cartesian coordinate
%           y,      cartesian coordinate
%           z,      cartesian coordinate
%           z0,     proyeccion plane
%
% out:
%           xc = [x', y', z]
% were:     x',     transformed cartesian coordinate
%           y',     transformed cartesian coordinate
%           z,      cartesian coordinate

%%
%%DEFINE
p  = 7.5;  %step coil
z0 = 0.0;
xr = xp;
sizexp = size(xp);
```

```
for i = 1:sizexp(1)
    z          = xp(i,3);
    giro       = 360*(z-z0)/p;
    xr(i,1:3) = Rotz(-giro,xp(i,1:3));
end
```

Listado A.16: Toma las coordenadas x,y,z del origen de coordendas y calcula su proyeccion helicoidal

```
function v2 = Rotz(a,v1)

if a ~= 0

    fc = size(v1);
    a = pi*a/180;
    v2 = zeros(fc(1),3);
    Rij = [cos(a), -sin(a), 0;
           sin(a),  cos(a), 0;
             0   ,    0   , 1];

          for i = 1:fc(1)
             vx = v1(i,1);
             vy = v1(i,2);
             vz = v1(i,3);
             v2(i,:) = (Rij*[vx; vy; vz])';
          end
else

    v2 = v1;
end
end
```

Listado A.17: Matriz de rotacion

```
function x = searchnode(node,coord)
%Funcion que busca las coordenadas de un nodo en la lista de
   coordenadas
%
%input:     node:   numero del nodo
%           coord:  listnode de la funcion loadnode

    if node == coord(node,1)
        x = coord(node,[2 3]);
    end
end
```

Listado A.18: Busca las coordenadas de un nodo en la lista de coordenadas

```
function N = weightfun(xe,xp)
%Funcion que calcula los coeficientes de interpolacion dentro de un
%elemento triangular
%
%input:     xi, coord i-node
%           xj, coord j-node
%           xk, coord k-node
%           xp = [x y] coordinates of point
%
%output:    N, size functions

if length(xe) == 3

    xi = xe(1,:);
    xj = xe(2,:);
    xk = xe(3,:);

    x1 = xi(1);
    y1 = xi(2);

    x2 = xj(1);
    y2 = xj(2);

    x3 = xk(1);
    y3 = xk(2);

    a1 = x2*y3 - x3*y2;     a2 = x3*y1 - x1*y3;         a3 = x1*y2 -
       x2*y1;
    b1 = y2 - y3;           b2 = y3 - y1;               b3 = y1 - y2;
    c1 = x3 - x2;           c2 = x1 - x3;               c3 = x2 - x1;

    D2 = a1 + a2 + a3;

    x = xp(1);
    y = xp(2);

    Ni = (a1 + b1*x + c1*y)/D2;
    Nj = (a2 + b2*x + c2*y)/D2;
    Nk = (a3 + b3*x + c3*y)/D2;

    N = [Ni, Nj, Nk];

end

if length(xe) == 4
```

```
46      Xi = xe(1,:);      xi = Xi(1); yi = Xi(2);
47      Xj = xe(2,:);      xj = Xj(1); yj = Xj(2);
48      Xk = xe(3,:);      xk = Xk(1); yk = Xk(2);
49      Xl = xe(4,:);      xl = Xl(1); yl = Xl(2);

51      x = xp(1);
52      y = xp(2);

54      A1 = xj + xk - xi - xl;
55      B1 = xk + xl - xi - xj;
56      C1 = xi + xk - xj - xl;
57      D1 = 4*x - xi - xj - xk - xl;

59      A2 = yj + yk - yi - yl;
60      B2 = yk + yl - yi - yj;
61      C2 = yi + yk - yj - yl;
62      D2 = 4*y - yi - yj - yk - yl;

64      Ax = A2*C1 - A1*C2;
65      Bx = A2*B1 - A1*B2 + D1*C2 - D2*C1;
66      Cx = D1*B2 - D2*B1;

68      Ay = B2*C1 - B1*C2;
69      By = A1*B2 - A2*B1 + C2*D1 - C1*D2;
70      Cy = D1*A2 - D2*A1;

72      DELTAx = sqrt(Bx^2 - 4*Ax*Cx);
73      DELTAy = sqrt(By^2 - 4*Ay*Cy);

75      EP1 = (-Bx+DELTAx)/2/Ax;
76      EP2 = (-Bx-DELTAx)/2/Ax;

78      NU1 = (-By+DELTAy)/2/Ay;
79      NU2 = (-By-DELTAy)/2/Ay;

81      if EP1 >= -1 && EP1 <= 1
82          if EP2 < -1 || EP2 > 1
83              EP = EP1;
84          end
85      end
86      if EP2 >= -1 && EP2 <= 1
87          if EP1 < -1 || EP1 > 1
88              EP = EP2;
89          end
90      end
91      if EP2 >= -1 && EP2 <= 1 && EP1 >= -1 && EP1 <= 1
92          disp('ERROR EPNU: epsylon tiene 2 soluciones validas')
```

```
93      elseif (EP2 < -1 || EP2 > 1) && (EP1 < -1 || EP1 > 1)
94          disp('ERROR EPNU: epsylon tiene 0 soluciones validas')
95      end

97      if NU1 >= -1 && NU1 <= 1
98          if NU2 < -1 || NU2 > 1
99              NU = NU1;
100         end
101     end
102     if NU2 >= -1 && NU2 <= 1
103         if NU1 < -1 || NU1 > 1
104             NU = NU2;
105         end
106     end
107     if NU2 >= -1 && NU2 <= 1 && NU1 >= -1 && NU1 <= 1
108         disp('ERROR EPNU: nu tiene 2 soluciones validas')
109     elseif (NU2 < -1 || NU2 > 1) && (NU1 < -1 || NU1 > 1)
110         disp('ERROR EPNU: nu tiene 0 soluciones validas')
111     end

113     Ni = 0.25*(1-EP)*(1-NU);
114     Nj = 0.25*(1+EP)*(1-NU);
115     Nk = 0.25*(1+EP)*(1+NU);
116     Nl = 0.25*(1-EP)*(1+NU);

118     N = [Ni, Nj, Nk, Nl];

120 end
121 end
```

Listado A.19: Calcula los coeficientes de interpolacion dentro de un elemento triangular

```
1  function pelem = whoelem(xp,listelem,listnode,pini)
2  %Funcion que encuentra a que elemento pertenece un punto de
       coordenadas xp
3  %con una tolerancia de al menos 1e-5, donde la tolerancia es abs(xp -
       xe),
4  %con xp la coordenada del punto y xe la coordenada del punto mas
       proximo
5  %del borde del elemento
6  %
7  %input:     xp = [x y] :  coordinates of point
8  %           listelem   :  list
9  %           listnode   :  list
10 %           pini       :  (opt) elemento que se obtuvo en el paso
       anterior
11 %
```

```
%output:    pelem      :  elemento en el que se encuentra xp
% disp('listelem')
% disp(listelem)
% disp('listnode')
% disp(listnode)
numel = size(listelem);
pelem = 0;
if nargin == 4
    Lup = pini:-1:1;
    Ldw = pini:+1:numel;
    cup = length(Lup);
    cdw = length(Ldw);
    iup = 1;
    idw = 1;

    for i = 1:numel(1)
        if iup <= cup
            up = Lup(iup);
            numnod1 = listelem(up,2);
            countup   = 0;
            for j1 = 1:numnod1
        xri1  = searchnode(listelem(up,2+j1),listnode);
        xjr11 = searchnode(listelem(up,3+j1),listnode)-xri1;
        xpr1  = xp - xri1;
        [a11 ap1] = angle180(xjr11,xpr1);
        a21 = pi + a11;
        if ap1 < a21 && ap1 > a11
            countup = countup + 1;
        end
            end
            iup = iup + 1;
            if countup == listelem(up,2)
                pelem = listelem(up,1);
                break
            end
        end

        if idw <= cdw
            down = Ldw(idw);
            numnod2 = listelem(down,2);
            countdown = 0;
            for j2 = 1:numnod2
        xri2  = searchnode(listelem(down,2+j2),listnode);
        xjr12 = searchnode(listelem(down,3+j2),listnode)-xri2;
        xpr2  = xp - xri2;
        [a12 ap2] = angle180(xjr12,xpr2);
        a22 = pi + a12;
```

```
59          if ap2 < a22 && ap2 > a12
60              countdown = countdown + 1;
61          end
62              end
63              idw = idw + 1;
64              if countdown == listelem(down,2)
65                  pelem = listelem(down,1);
66                  break
67              end
68          end
69      end
70  end

72  if nargin == 3
73      for i = 1:numel(1)
74          numnod = listelem(i,2);
75          count = 0;
76          for j = 1:numnod
77              xri  = searchnode(listelem(i,2+j),listnode);
78              xjr1 = searchnode(listelem(i,3+j),listnode)-xri;
79              xpr  = xp - xri;
80              [a1 ap] = angle180(xjr1,xpr);
81              a2 = pi + a1;
82              if ap < a2 && ap > a1
83                  count = count + 1;
84              end
85          end
86          if count == listelem(i,2)
87              pelem = listelem(i,1);
88              break
89          end
90      end
91  end

93  if pelem == 0
94      disp('ERROR 001: El punto xp no pertenece al domino')
95      disp('xp = ')
96      disp(xp)
97      return
98  end

100 end
```

Listado A.20: Encuentra un elemento al que pertence un punto del dominio

```
1  %Script para hacer videos del campode velocidades en una seccion
2  %transversal del tubo
```

```
fig1=figure(1);
aviobj = avifile('example3.avi','compression','None');
angle               = 180; %angulo entre Cf Y Cg
Y_Minimun           = 15;
Cicle_Division      = 8;
Cicles_Calculated   = 10;
REo                 = 80;
ST                  = 0.4;
prop                = 1;
Cicle_Fluent_Dats   = 4; %Ciclo del que se extrajeron los datos
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

StepList         = meshcicles(Y_Minimun,Cicle_Division,Cicles_Calculated,REo,ST,prop);
fcSL             = size(StepList);
NameList         = StepList;
Total_Steps      = StepList(fcSL(1),1);
Total_Cicles     = fcSL(1)/Total_Steps;
iniSL            = (Cicle_Fluent_Dats-1)*Total_Steps + 1;
finSL            = iniSL + Total_Steps - 1;
Name_Cicles      = StepList(iniSL:finSL,1);
Time_Cicles      = StepList(iniSL:finSL,2);
pos              = Name_Cicles;
for i = 1:Total_Cicles
    NameList(pos,1) = Name_Cicles;
    NameList(pos,2) = Time_Cicles;
    pos = pos + Total_Steps;
end

winsize = get(fig1,'Position');
%winsize(1:2) = [0 0];
numframes=64;
A=moviein(numframes,fig1,winsize);
set(fig1,'NextPlot','replacechildren')
for i=1:numframes
    strname = sprintf('Vf_time_%f_step_%d',NameList(i,2),NameList(i,1));
    relativefield(strname,angle);
   %plot(X(i),Y(i)); % plot command
   % add axis label, legends, titles, etc. in here
   A(:,i)=getframe(fig1,winsize);
   F = getframe(fig1);
   aviobj = addframe(aviobj,F);
end

movie(fig1,A,1,64,winsize)
```

```
save filename.mat A
load filename.mat


close(fig1);
aviobj = close(aviobj);

%mpgwrite(A,jet,'movie.mpg');
```

Listado A.21: Crea videos

```
function OSCjou(Ciclos,REo,ST,prop)
%CD FM0: fluid mechanics cero case
%scale grid (was done in mm)
%check grid
%define water as fluid boundary condition
%define glycerine as fluid boundary condition
%define piso: 1 skewness correction, 0 neightbor correction, non
    coupling,
%              pressure standard, momentum first order,
%              under relaxations:
%                                            - 1   pressure
%                                            - 0.3 momentum
%residual: 5e-5 for all
%Szct.jou
%number of steps = 1 dejarlo por defecto
%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% SETUP
deltaYmin   = 15; %genera un ciclo con 64 fotogramas
Pasos_ciclo = 8;  %trozos en que se divide un ciclo
res         = 5e-5;

%% FUNCTION
StepList = meshcicles(deltaYmin,Pasos_ciclo,Ciclos,REo,ST,prop);
jouname=sprintf('OSC_Ymin%.0f_RE%.0f_ST%.1f.jou',deltaYmin,REo,ST);
fopen(jouname,'a+');
fo = fopen(jouname,'w');

%Pasos = max(StepList(:,1));
for i = 1:length(StepList)-1
    %%
    %
        ----------------------------------------------------------------------- %
```

```
32      fprintf(fo,'/define/models/unsteady-2nd-order yes\n');
33      %%
34      %
            ------------------------------------------------------------------------%

35      fprintf(fo,'/solve/monitors/residual/convergence-criteria\n');
36      fprintf(fo,'%d\n',res); %continuity
37      fprintf(fo,'%d\n',res); %x-momentum
38      fprintf(fo,'%d\n',res); %y-momentum
39      fprintf(fo,'%d\n',res); %z-momentum
40      %%
41      %
            ------------------------------------------------------------------------%

42      fprintf(fo,'define\n');
43      fprintf(fo,'periodic-conditions\n');
44      fprintf(fo,'massflow-rate-specification\n');
45      fprintf(fo,'%.16f\n',StepList(i+1,3));
46      fprintf(fo,'\n');
47      fprintf(fo,'\n');
48      fprintf(fo,'\n');
49      fprintf(fo,'0\n');
50      fprintf(fo,'0\n');
51      fprintf(fo,'1\n');
52      fprintf(fo,'quit\n');
53      fprintf(fo,'quit\n');
54      %%
55      %
            ------------------------------------------------------------------------%

56      fprintf(fo,'/solve/set/time-step %.16f\n',StepList(i+1,2)-StepList
            (i,2));
57      %%
58      %
            ------------------------------------------------------------------------%

59      fprintf(fo,'/solve/dual-time-iterate 1 1000\n');
60      %%
61      %
            ------------------------------------------------------------------------%

62      fprintf(fo,'wcd\n');
63      fprintf(fo,'Re80St04_y15_t%f_s%d.gz\n',StepList(i+1,2),StepList(i
            +1,1));
64      %%
65      %
```

```
%-------------------------------------------------------------------------%

%EXPORTAR CAMPO DE VELOCIDADES
fprintf(fo,'file\n');
fprintf(fo,'export\n');
fprintf(fo,'ascii\n');
fprintf(fo,'Vf_time_%f_step_%d\n',StepList(i+1,2),StepList(i+1,1))
    ;
fprintf(fo,'z3-3.75\n');
fprintf(fo,'()\n');
fprintf(fo,'no\n');
fprintf(fo,'z-velocity\n');
fprintf(fo,'y-velocity\n');
fprintf(fo,'x-velocity\n');
fprintf(fo,'()\n');
fprintf(fo,'no\n');
fprintf(fo,'quit\n');
fprintf(fo,'quit\n');
%%
%
%-------------------------------------------------------------------------%

%EXPORTAR VELOCIDAD EN LINEA MERIDIANA 1
fprintf(fo,'file\n');
fprintf(fo,'export\n');
fprintf(fo,'ascii\n');
fprintf(fo,'Vm1_time_%f_step_%d\n',StepList(i+1,2),StepList(i+1,1)
    );
fprintf(fo,'lm1\n');
fprintf(fo,'()\n');
fprintf(fo,'no\n');
fprintf(fo,'z-velocity\n');
fprintf(fo,'y-velocity\n');
fprintf(fo,'x-velocity\n');
fprintf(fo,'()\n');
fprintf(fo,'no\n');
fprintf(fo,'quit\n');
fprintf(fo,'quit\n');
%%
%
%-------------------------------------------------------------------------%

%EXPORTAR VELOCIDAD EN LINEA MERIDIANA 2
fprintf(fo,'file\n');
fprintf(fo,'export\n');
fprintf(fo,'ascii\n');
fprintf(fo,'Vm2_time_%f_step_%d\n',StepList(i+1,2),StepList(i+1,1)
```

```
        );
105     fprintf(fo,'lm2\n');
106     fprintf(fo,'()\n');
107     fprintf(fo,'no\n');
108     fprintf(fo,'z-velocity\n');
109     fprintf(fo,'y-velocity\n');
110     fprintf(fo,'x-velocity\n');
111     fprintf(fo,'()\n');
112     fprintf(fo,'no\n');
113     fprintf(fo,'quit\n');
114     fprintf(fo,'quit\n');
115     %%
116     %
        ------------------------------------------------------------------------%

117     %EXPORTAR SUMATORIO DE FUERZAS
118     fprintf(fo,'report\n');
119     fprintf(fo,'forces\n');
120     fprintf(fo,'wall-forces\n');
121     fprintf(fo,'yes\n');
122     fprintf(fo,'0\n');
123     fprintf(fo,'0\n');
124     fprintf(fo,'1\n');
125     fprintf(fo,'yes\n');
126     fprintf(fo,'Wf_time_%f_step_%d\n',StepList(i+1,2),StepList(i+1,1))
        ;
127     fprintf(fo,'quit\n');
128     fprintf(fo,'quit\n');
129 end
130 fclose all;
```

Listado A.22: Genera journals para flujo oscilatorio

```
1  Id =  3;

3  nT =  4;
4  LW =  2;
5  pt1    = '-b';
6  pt10   = '-g';
7  pt80   = '-r';
8  pt1000 = '-m';
9  hold on
10 axis equal
11 grid on

13 x01 = plotp(XR1,Id,pt1,nT*0.029015,LW);
14 plotp(XR10,Id,pt10,nT*0.002902,LW);
```

```
plotp(XR80,Id,pt80,nT*0.228909,LW);
plotp(XR1000,Id,pt1000,nT*0.018313,LW);
legend('Re_o 1, St 0.4', 'Re_o 10, St 0.4','Re_o 80, St 0.4', 'Re_o
    1000, St 0.4','Location','NorthEastOutside');

coil = importdata('xycoil.txt');
plot(coil(:,1),coil(:,2),'-k','LineWidth',2)
circle(0,0,2.5)
axis([-2.52 2.52 -2.52 2.52])
xlabel('X (mm)')
ylabel('Y (mm)')
plot(x01(1),x01(2),'.k','MarkerSize',20);%LineWidth',2,...
                          %       'MarkerEdgeColor','k',...
                          %       'MarkerFaceColor','k',...
                          %       'MarkerSize',25)

clear Id nT LW pt1 pt10 pt80 pt1000 x01 x02 x03 x04 coil
```

Listado A.23: Dibuja graficas de trayectorias en seccion transversal

```
%Postprocessing
hold
grid on
axis equal
%circle(0,0,2.5)
%plot3(XP(:,1),XP(:,2),XP(:,3))
XR = abstorel(XP(:,1:3));
%plot3(xr0(:,1),xr0(:,2),xr0(:,3),'r')
[f c] = size(xp);
fid = fopen('xp.txt','a');
ff = sprintf('PARTICLES TRACKS: Samuel Espin Tolosa 2012\n');
fwrite(fid,ff);
ff = sprintf('-----------------------------------------\n');
fwrite(fid,ff);
ff = sprintf('\tX\t\t\tY\t\t\tZ\t\t\tVx\t\t\tVy\t\t\tVz\t\t\tS\t\t\tT\
    t\tId\n');
fwrite(fid,ff);
for i = 1:f
    ff = sprintf('%f\t%f\t%f\t',xp(i,1),xp(i,2),xp(i,3));
    fwrite(fid,ff);
    ff = sprintf('%f\t%f\t%f\t',xp(i,4),xp(i,5),xp(i,6));
    fwrite(fid,ff);
    ff = sprintf('%f\t%f\t%d\n',xp(i,7),xp(i,8),xp(i,9));
    fwrite(fid,ff);
end
fclose all;
```

```
%SIMBOLS

% b     blue          .     point              -     solid
% g     green         o     circle             :     dotted
% r     red           x     x-mark             -.    dashdot
% c     cyan          +     plus               --    dashed
% m     magenta       *     star             (none)  no line
% y     yellow        s     square
% k     black         d     diamond
% w     white         v     triangle (down)
%                     ^     triangle (up)
%                     <     triangle (left)
%                     >     triangle (right)
%                     p     pentagram
%                     h     hexagram
```

Listado A.24: Postprocesado

```
jouname='Szct.jou';
fopen(jouname,'a+');
fo = fopen(jouname,'w');

p = 7.5e-3;
n = 100; %60 = 1*2*2*3*5 %100 = 1*2*2*5*5
r = 2*5; %cada cuanto quieres partir

N = 1;
z = 0:r*p/n:p*N;

fprintf(fo,'surface\n');
for i = 1:length(z)
    fprintf(fo,'plane\n');
    longitude = 1000*z(i);
    if longitude <  10
        if i < 10
            fprintf(fo,'Z0%d-0%.2f\n',i,longitude);
        end
        if i >= 10 && i < 100
            fprintf(fo,'Z0%d-0%.2f\n',i,longitude);
        end
    end
    if longitude >= 10 && longitude < 100
        if i < 10
            fprintf(fo,'Z0%d-%.2f\n',longitude);
        end
        if i >= 10 && i < 100
            fprintf(fo,'Z%d-%.2f\n',longitude);
```

```
30          end
31      end

33      fprintf(fo,'0\n');
34      fprintf(fo,'0\n');
35      fprintf(fo,'%.8f\n',z(i));
36      fprintf(fo,'0\n');
37      fprintf(fo,'1\n');
38      fprintf(fo,'%.8f\n',z(i));
39      fprintf(fo,'1\n');
40      fprintf(fo,'0\n');
41      fprintf(fo,'%.8f\n',z(i));
42  end
43  fprintf(fo,'quit\n');

45  fclose all;
```

Listado A.25: Genera journals para las secciones transversales de fluent

Anexo B

B.1. Listado de acrónimos

OBC oscillatory baffled colum.

POF Pure Oscillatory Flow.

OBF oscillatory baffled flow.

CR constriction ratio

OFR reactores de flujo oscilatorio

OBR oscillatory baffled reactor

TUI Text User Interface