

# Transferencia de Calor y Masa - 6731

## Simulación de disco de frenos por elementos finitos

### Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Modelo matemático</b>	<b>2</b>
<b>3. Desarrollo numérico</b>	<b>6</b>
3.1. Pre-procesado . . . . .	6
3.1.1. Geometría (Onshape) . . . . .	6
3.1.2. Malla (gmsh) . . . . .	8
3.2. Procesado . . . . .	13
3.2.1. Funciones de forma . . . . .	13
3.2.2. Derivadas de las funciones de forma . . . . .	14
3.2.3. Matriz jacobiana . . . . .	15
3.2.4. Gradiente de las funciones de forma . . . . .	16
3.2.5. Formulación de Petrov-Galerkin . . . . .	17
3.2.6. Integración numérica . . . . .	22
3.2.7. Aplicación de las condiciones de borde . . . . .	27
3.2.8. Ensamble . . . . .	34
3.2.9. Resolución del problema transitorio no lineal . . . . .	36
3.3. Post-procesado . . . . .	41
<b>4. Verificación del código</b>	<b>43</b>
<b>5. Implementación del código</b>	<b>44</b>
5.1. Resultados para distintas velocidades iniciales . . . . .	45
5.1.1. Caso 1: $V_0 = 50 \frac{\text{km}}{\text{h}}$ . . . . .	45
5.1.2. Caso 2: $V_0 = 100 \frac{\text{km}}{\text{h}}$ . . . . .	46
5.1.3. Caso 3: $V_0 = 150 \frac{\text{km}}{\text{h}}$ . . . . .	47
5.1.4. Caso 4: $V_0 = 200 \frac{\text{km}}{\text{h}}$ . . . . .	48
5.1.5. Caso 5: $V_0 = 250 \frac{\text{km}}{\text{h}}$ . . . . .	49
5.2. Análisis de los resultados . . . . .	51
5.3. Consideraciones del modelo . . . . .	51
5.3.1. Influencia del planteo tridimensional . . . . .	51
5.3.2. Influencia de la no linealidad material . . . . .	55
<b>6. Análisis estructural: termoelasticidad</b>	<b>56</b>
<b>7. Conclusión</b>	<b>60</b>

## 1. Introducción

En este trabajo, se busca modelar y resolver numéricamente la transferencia de calor en un disco de frenos durante un proceso de frenado brusco, mediante una formulación Euleriana.

Se detallará a continuación el modelado matemático del problema, la generación de la geometría y la malla, la implementación de un código de elementos finitos para resolver el problema, la visualización y el análisis de los resultados obtenidos.

## 2. Modelo matemático

Se plantea la situación siguiente:

Un vehículo se desplaza a una velocidad inicial  $v_0$ . En el instante inicial ( $t_0 = 0$ ), se pisa el pedal del freno a fondo, hasta que en un tiempo final  $t_f$ , el vehículo se detiene ( $v_f = 0$ ). Suponiendo una evolución lineal de la velocidad del vehículo, se tiene:

$$v(t) = v_0 + a t$$

Siendo la aceleración:

$$a = \frac{v_f - v_0}{t_f - t_0} = \frac{0 - v_0}{t_f - 0} = -\frac{v_0}{t_f}$$

Por lo tanto, se puede expresar la velocidad como:

$$v(t) = v_0 \left(1 - \frac{t}{t_f}\right)$$

Conociendo el radio de las ruedas del vehículo  $R_r$ , se puede plantear:

$$v(t) = \omega(t) R_r \Rightarrow \omega(t) = \frac{v_0}{R_r} + \frac{a}{R_r} t = \omega_0 + \gamma t = \omega_0 \left(1 - \frac{t}{t_f}\right)$$

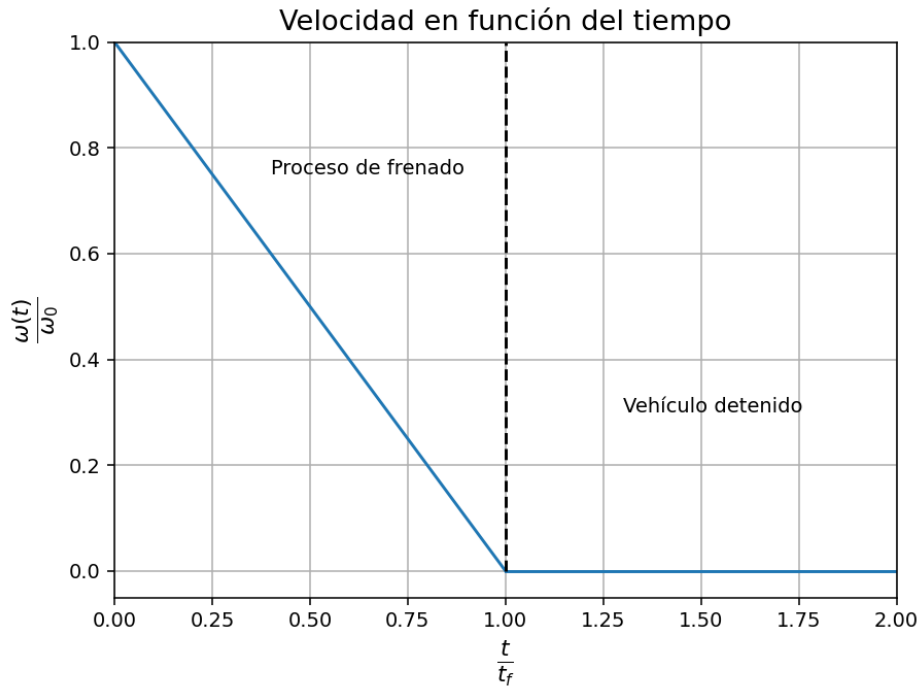


Figura 1: Velocidad angular del vehículo en función del tiempo

El vehículo se desplaza inicialmente con energía cinética:

$$E_c(t) = \frac{1}{2} m v^2(t)$$

Su potencia debida a la variación de energía cinética es entonces:

$$P_c(t) = \dot{E}_c(t) = m v(t) \frac{dv(t)}{dt} = m \omega(t) R_r \gamma R_r = m R_r^2 \omega(t) \gamma$$

El vehículo también pierde energía por el arrastre en su interacción con el aire (DRAG). La potencia disipada por el DRAG es:

$$P_D(t) = -\frac{1}{2}\rho_a C_D A_R v^3(t) = -\frac{1}{2}\rho_a C_D A_R R_r^3 \omega^3(t)$$

Donde  $\rho_a$  es la densidad del aire,  $C_D$  es el coeficiente de DRAG y  $A_R$  es un área de referencia.

La variación de la energía cinética es igual a la suma de las potencias disipadas:

$$P_c(t) = P_f(t) + P_D(t)$$

Siendo  $P_f(t)$  la potencia disipada por fricción en los frenos.

$$P_f(t) = P_c(t) - P_D(t) = m R_r^2 \omega(t) \gamma + \frac{1}{2}\rho_a C_D A_R R_r^3 \omega^3(t)$$

Notar que, como  $\gamma < 0$ ,  $P_c(t) < 0$  y  $|P_c(t)| > |P_D(t)|$ , por lo tanto,  $P_f(t) < 0$ . Es decir, todas las potencias mencionadas son negativas, porque son potencias disipadas.

Suponiendo que cada rueda tiene dos pastillas de frenos, la potencia total disipada es:

$$P_f(t) = 8 \int_{S_q} \underline{f}(t) \cdot \underline{v}_d(t) dS$$

Siendo  $S_q$  la superficie de contacto entre una pastilla y el disco,  $\underline{f}(t)$  la fuerza de fricción por unidad de superficie, y  $\underline{v}_d(t)$  la velocidad del diferencial de área  $dA$ .

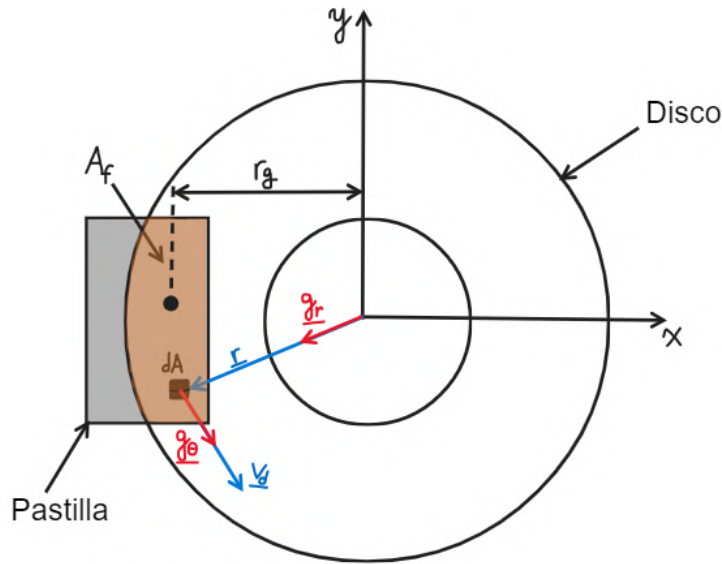


Figura 2: Esquema simplificado del disco y la pastilla de frenos

El vector posición del diferencial de área es:

$$\underline{r} = x \underline{e}_x + y \underline{e}_y = r \cos(\theta) \underline{e}_x + r \sin(\theta) \underline{e}_y$$

Derivando, se obtiene su velocidad:

$$\underline{v}_d = -\omega \underbrace{r \sin(\theta)}_{=y} \underline{e}_x + \omega \underbrace{r \cos(\theta)}_{=x} \underline{e}_y = -\omega y \underline{e}_x + \omega x \underline{e}_y = \omega r \underline{g}_\theta$$

Expresando la velocidad en la base polar, y sabiendo que la fuerza de rozamiento se opone a la velocidad,

$$P_f(t) = 8 \int_{S_q} -f(t) \underline{g}_\theta \cdot \omega(t) r \underline{g}_\theta dS = -8 \int_{S_q} \underbrace{f(t) \omega(t) r}_{=q_T(t)} dS$$

Siendo  $q_T(t)$  el flujo de calor total disipado en un par pastilla-disco. Suponiéndolo uniforme en la superficie de contacto  $S_q$ , y definiendo el área de esta superficie como  $A_f$ ,

$$P_f(t) = -8 q_T(t) A_f$$

Reemplazando la potencia disipada por fricción, se despeja el flujo de calor total disipado entre una pastilla y el disco:

$$q_T(t) = -\frac{P_f(t)}{8A_f} = -\frac{m R_r^2 \omega(t) \gamma + \frac{1}{2} \rho_a C_D A_R R_r^3 \omega^3(t)}{8A_f}$$

De este flujo de calor total, una porción es absorbida por la pastilla y la otra por el disco. Se determina la parte absorbida por el disco mediante la fórmula de [Charron](#):

$$q(t) = \kappa q_T(t), \quad \kappa = \frac{\sqrt{k_d c_{p_d} \rho_d}}{\sqrt{k_d c_{p_d} \rho_d} + \sqrt{k_p c_{p_p} \rho_p}} \quad (1)$$

Donde los subíndices  $d$  o  $p$  implican que las propiedades son del disco o de la pastilla respectivamente.

El flujo de calor absorbido por el disco es entonces:

$$q(t) = -\kappa \frac{m R_r^2 \omega(t) \gamma + \frac{1}{2} \rho_a C_D A_R R_r^3 \omega^3(t)}{8A_f} \quad (2)$$

Además de este flujo de calor generado por la fricción que es absorbido por el disco, existe un flujo de calor que es extraído por convección con el aire exterior, que dependerá de la velocidad absoluta de cada partícula del disco. Esta velocidad está compuesta por la velocidad de traslación del centro de masa del disco ( $\underline{v}_t = -^t V \underline{e}_x$ ) y la de rotación alrededor del centro de masa ( $\underline{v}_d$ ):

$$\underline{v} = \underline{v}_t + \underline{v}_d = -(^t V + ^t \omega ^t y) \underline{e}_x + ^t \omega ^t x \underline{e}_y$$

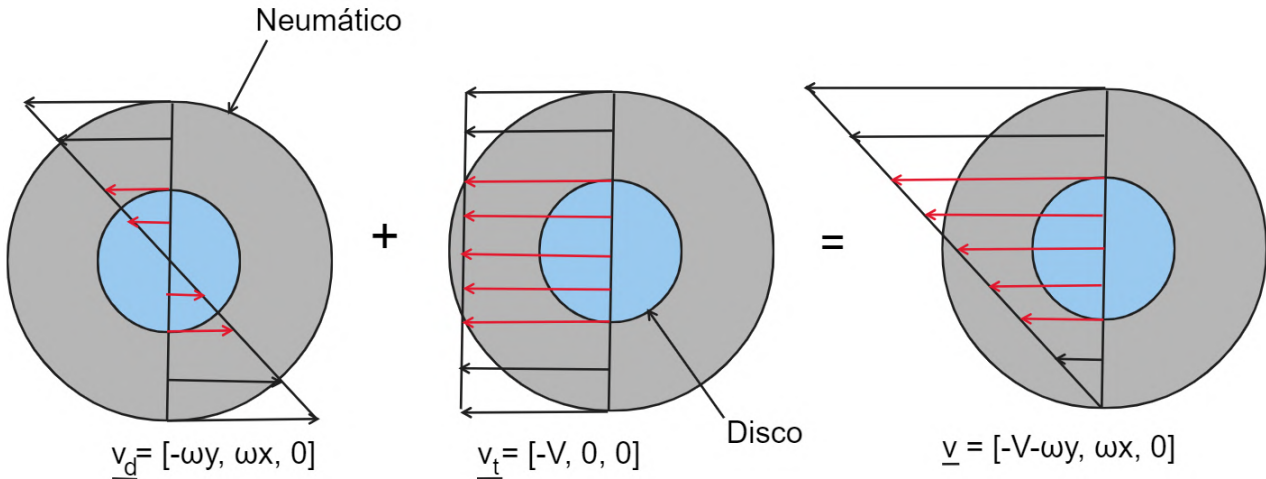


Figura 3: Esquema del campo de velocidades absoluto, por superposición de la rotación y la traslación

Se modela entonces el problema de transferencia de calor en el disco mediante la formulación Euleriana. Es decir, debido a la velocidad del disco, se considera que la posición de cualquier partícula de éste es función del tiempo:

$${}^t \underline{x} = {}^t \underline{x}(t)$$

Por lo tanto, aplicando la regla de la cadena, la derivada material de la temperatura es:

$$\frac{D {}^t T}{Dt} = \frac{\partial {}^t T}{\partial t} + \frac{\partial {}^t x^a}{\partial t} \frac{\partial {}^t T}{\partial x^a} = \frac{\partial {}^t T}{\partial t} + {}^t v^a \frac{\partial {}^t T}{\partial x^a} = \frac{\partial {}^t T}{\partial t} + {}^t \underline{v} \cdot {}^t \nabla {}^t T$$

En otras palabras, se plantea el problema como un caso de **convección-difusión** sin generación de calor volumétrico, y **con propiedades del material del disco que dependen de su temperatura**. Pero para evitar problemas de estabilidad numérica que suelen surgir cuando se resuelven problemas de estas características con grandes velocidades, se usará únicamente la componente local de la velocidad (la de rotación  $\underline{v}_d$ ) en la ecuación diferencial que gobierna el problema.

$${}^t \rho(T) {}^t c_p(T) \frac{\partial {}^t T}{\partial t} + {}^t \rho(T) {}^t c_p(T) {}^t \underline{v}_d \cdot {}^t \nabla {}^t T = {}^t k(T) \nabla^2 {}^t T, \quad {}^t \underline{v}_d = -{}^t \omega ^t y \underline{e}_x + {}^t \omega ^t x \underline{e}_y \quad (3)$$

Donde  ${}^t \underline{v}_d$  es la velocidad de rotación del disco.

Nota: Se abandonaron, por simplicidad, los subíndices  $d$  para las propiedades del disco:  $\rho$ ,  $c_p$ ,  $k$ .

Se tiene además, dos condiciones de borde:

- Condición de borde del tipo Neumann correspondiente al flujo de calor generado por la fricción absorbido por el disco aplicado en la superficie de contacto:

$$-{}^t k(T) \nabla {}^t T \cdot \underline{n} = {}^t q \quad \forall {}^t \underline{x} \in S_q \quad (4)$$

- Condición de borde mixta correspondiente a al flujo de calor extraído por convección con el aire exterior en la superficie del disco expuesta al éste:

$$-{}^t k(T) \nabla {}^t T \cdot \underline{n} = {}^t h_c ({}^t T - T_\infty) \quad \forall {}^t \underline{x} \in S_c \quad (5)$$

Donde  ${}^t h_c$  es el coeficiente de convección en el tiempo  $t$ ,  $T_\infty$  la temperatura del aire exterior, lo suficientemente lejos de la superficie del disco (fuera de la capa límite térmica) y  $S_c$  la superficie del disco en contacto con el aire.

Esta ultima condición de borde se aplica debido a que no se utiliza la velocidad completa en la ecuación diferencial (se utiliza únicamente la componente de rotación  ${}^t v_d$ ). Por lo tanto, para contemplar los efectos de la velocidad de traslación ( ${}^t v_t$ ), se aplica la condición de borde de convección, en la cual se usará únicamente la velocidad de traslación en el cálculo del coeficiente de convección  $h_c$ .

Se escogió una fundición gris como material del disco, y se obtuvieron las leyes de variación de sus propiedades físicas en función de la temperatura de la norma [ASME BPVC 2021 Section II part D \(metric\)](#), página 1115.

**Table TCD**  
**Nominal Coefficients of Thermal Conductivity (TC) and Thermal Diffusivity (TD)**  
**(Cont'd)**

Carbon and Low Alloy Steels (Cont'd)			Ductile Cast Iron		High Chrome Steels		High Chrome Steels		High Alloy Steels		High Alloy Steels	
Material Group F [Note (6)]					Material Group G [Note (7)]		Material Group H [Note (8)]		Material Group I [Note (9)]		Material Group J [Note (10)]	
Temp., °C	TC	TD	TC	TD	TC	TD	TC	TD	TC	TD	TC	TD
20	22.3	6.61	37.50	11.44	24.6	7.12	20.1	5.70	17.3	4.80	14.8	3.90
50	23.1	6.67	38.50	11.59	24.7	6.93	20.2	5.65	17.6	4.84	15.3	3.94
75	23.8	6.71	39.18	11.56	24.7	6.80	20.2	5.59	18.0	4.85	15.8	3.99
100	24.4	6.74	39.73	11.51	24.8	6.69	20.3	5.51	18.4	4.86	16.2	4.04
125	25.0	6.76	40.15	11.44	24.9	6.58	20.4	5.44	18.9	4.87	16.6	4.08
150	25.5	6.76	40.45	11.36	24.9	6.48	20.5	5.37	19.3	4.87	17.0	4.14
175	25.9	6.75	40.64	11.26	25.0	6.37	20.5	5.30	19.8	4.87	17.5	4.19
200	26.3	6.71	40.73	11.13	25.0	6.26	20.6	5.24	20.2	4.88	17.9	4.24
225	26.6	6.66	40.73	10.99	25.1	6.14	20.7	5.18	20.7	4.88	18.3	4.30
250	26.9	6.58	40.64	10.83	25.1	6.02	20.8	5.12	21.1	4.88	18.6	4.35
275	27.2	6.49	40.47	10.65	25.2	5.90	20.9	5.07	21.5	4.87	19.0	4.41
300	27.4	6.39	40.23	10.44	25.2	5.78	21.0	5.02	21.9	4.86	19.4	4.46
325	27.5	6.27	39.93	10.22	25.2	5.65	21.1	4.95	22.2	4.84	19.8	4.52
350	27.7	6.15	...	...	25.3	5.53	21.2	4.89	22.5	4.81	20.1	4.57
375	27.8	6.01	...	...	25.3	5.41	21.3	4.81	22.8	4.77	20.5	4.63
400	27.9	5.87	...	...	25.3	5.29	21.4	4.72	23.0	4.71	20.8	4.69

Figura 4: Propiedades térmicas de la fundición de hierro escogidas para el disco. TC es la conductividad  $k$ , y TD es el coeficiente de difusión  $\alpha$

Con estos datos, se obtuvieron las siguientes curvas aproximadas para la conductividad térmica  $k$  y el producto de la densidad por el calor específico  $\rho c_p$ .

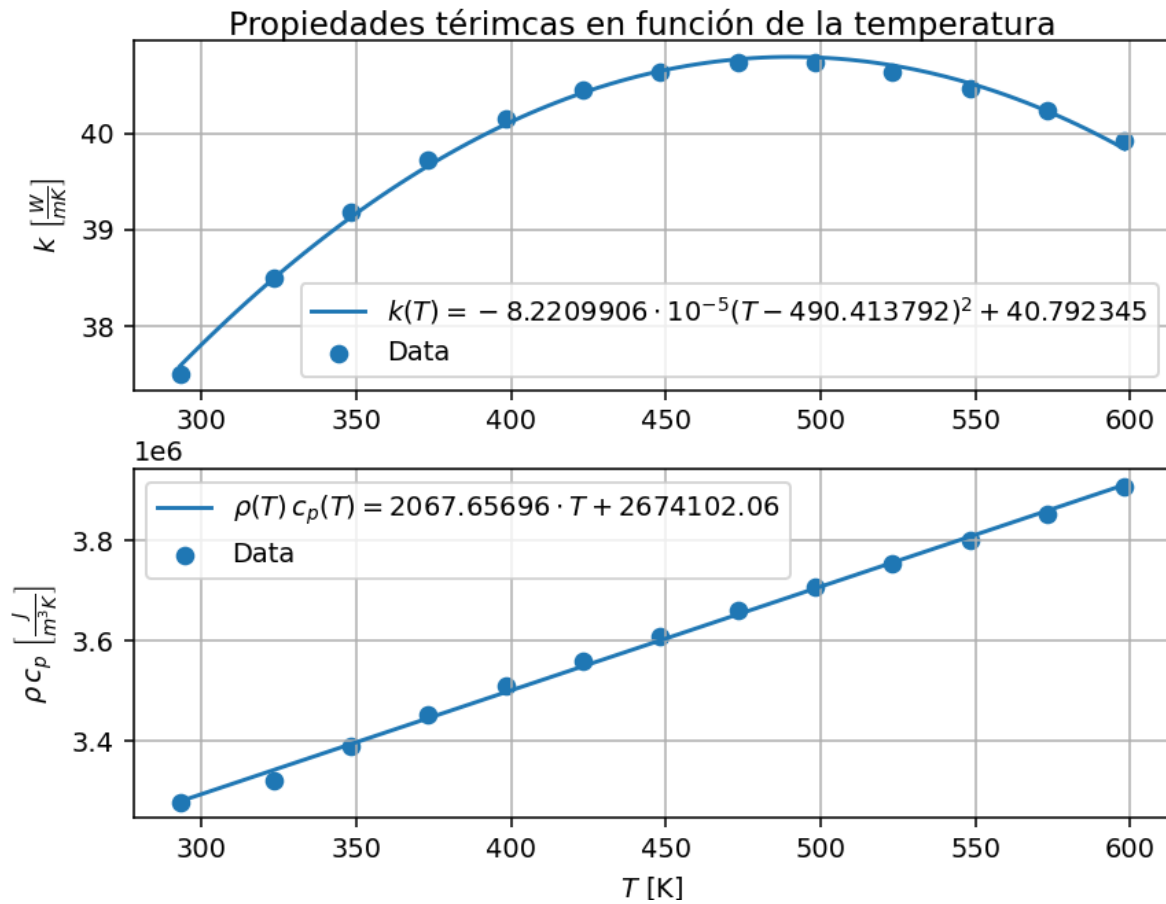


Figura 5: Conductividad térmica en función de la temperatura

### 3. Desarrollo numérico

#### 3.1. Pre-procesado

##### 3.1.1. Geometría (Onshape)

Se generó la geometría del disco con el software [Onshape](#). Se dividió el disco en dos partes, marcando la zona donde apoyaría la pastilla, para poder aplicar posteriormente la condición de borde de Neumann en el código.

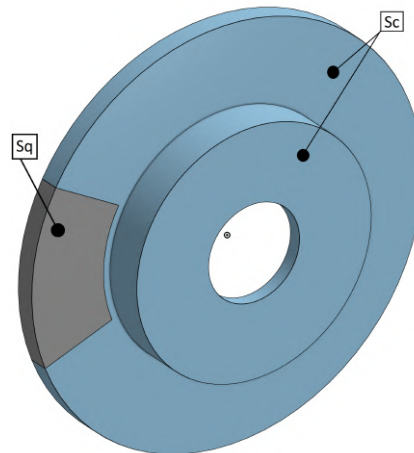
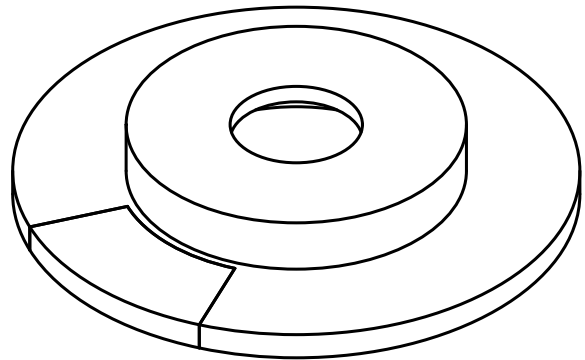
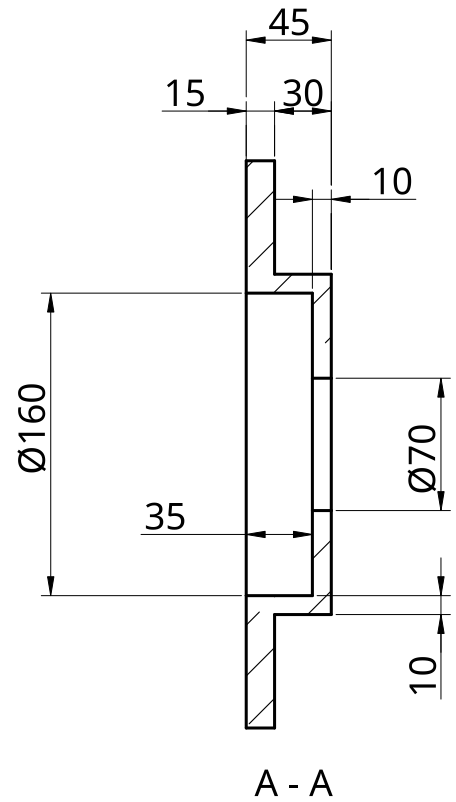
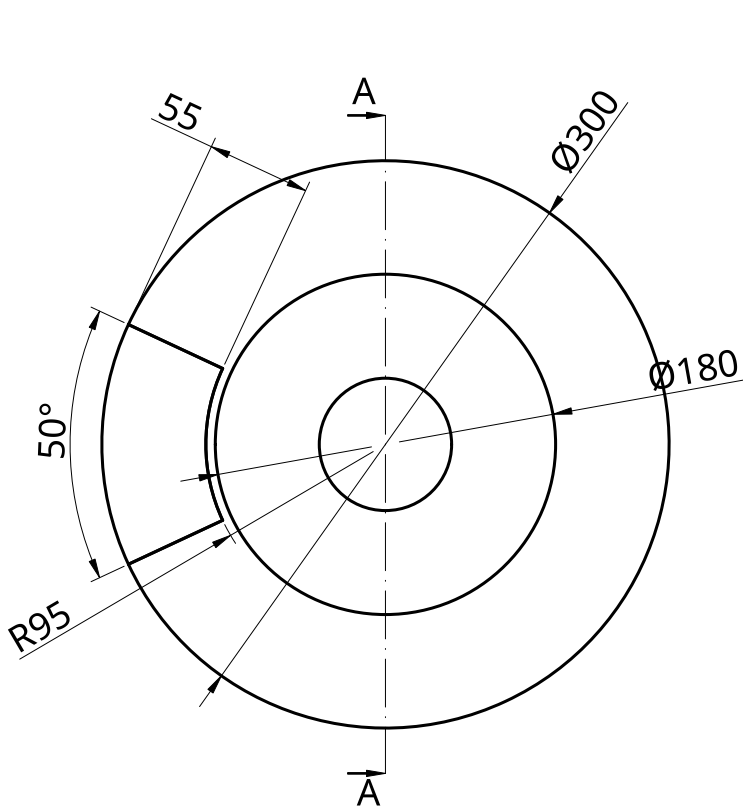




Figura 6: Geometría 3D del disco modelada en Onshape, con la superficie donde se aplicarán las condiciones en las superficies indicadas

Las dimensiones del disco son las que se exponen a continuación:



UNLESS OTHERWISE SPECIFIED, DIMENSIONS ARE IN MILLIMETERS  ANGULAR = ± °  SURFACE FINISH   DO NOT SCALE DRAWING  BREAK ALL SHARP EDGES AND REMOVE BURRS  FIRST ANGLE PROJECTION 		NAME	SIGNATURE	DATE	TITLE  Disco de frenos		
	DRAWN	MARCOS KRUPICZER		2023-07-09			
	CHECKED						
	APPROVED						
						SIZE <b>A4</b>	DWG NO.
	MATERIAL Fundición gris	FINISH		SCALE 1:4	WEIGHT	SHEET 1 of 1	

### 3.1.2. Malla (gmsh)

Con la geometría generada, se exportó un archivo **.STEP** de **Onshape** para luego importarlo al software **gmsh** para producir la malla. Al importar la geometría, se observa una pantalla como la siguiente:

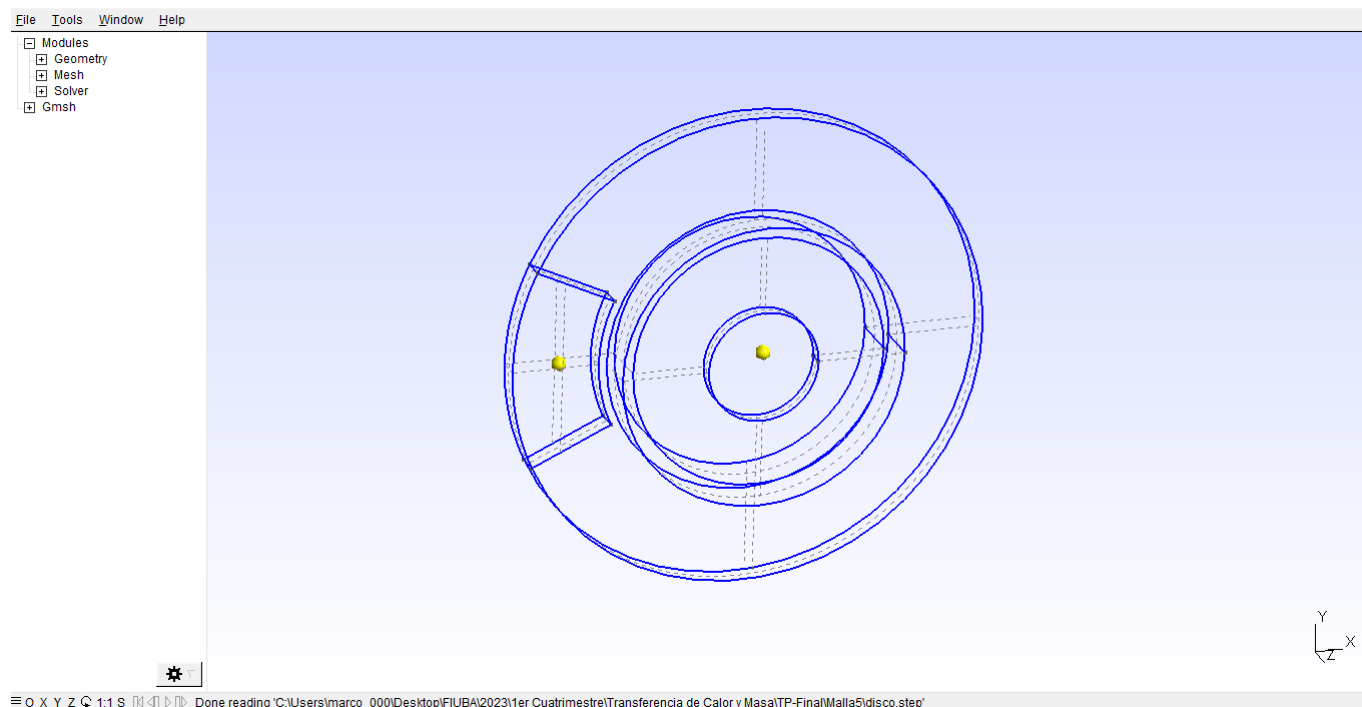


Figura 7: Captura de pantalla del **gmsh** al importar la geometría en formato **.STEP**

Las dos pequeñas esferas amarillas que se ven en la imagen anterior simbolizan los dos volúmenes que conforman la pieza. Lo primero que hay que hacer para obtener una malla adecuada es unir ambos volúmenes. Para ello, se siguen las instrucciones siguientes en la barra de opciones a la izquierda:

**Geometry → Elementary entities → Boolean → Union (Fuse)**

Luego, se clickea sobre uno de los volúmenes (esferas amarillas), se pulsa la letra **e** en el teclado, se clickea sobre el otro volumen, y se pulsa nuevamente la tecla **e**. Para salir de la operación, se pulsa finalmente la tecla **q**. Al terminar este procedimiento, se debería observar una única esfera amarilla, simbolizando un único volumen.



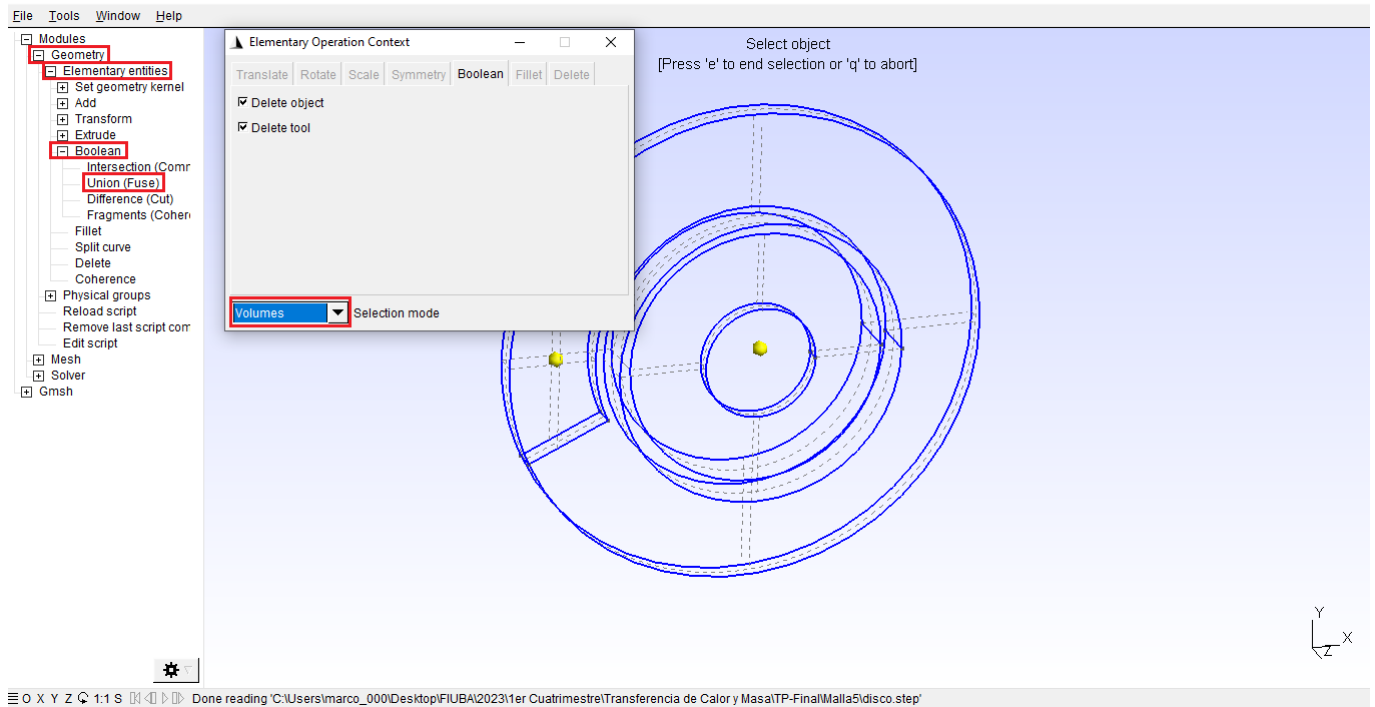


Figura 8: Captura de pantalla del **gmsh** con los pasos a seguir para unir los volúmenes

El segundo paso a realizar es la definición de los **Physical groups**. Éstos son grupos conformados por puntos, líneas, superficies o volúmenes que poseen ciertas características definidas por el usuario. Estos grupos serán de utilidad para aplicar las condiciones de borde en el código. El procedimiento es el siguiente:

**Geometry → Physical groups → Add → Point, Curve, Surface o Volume**

Luego, se nombra el grupo, se cliclean los objetos que se desean incluir (las superficies son las líneas punteadas y los volúmenes son las esferas amarillas), se pulsa la tecla **e** y la tecla **q** para finalizar la operación.

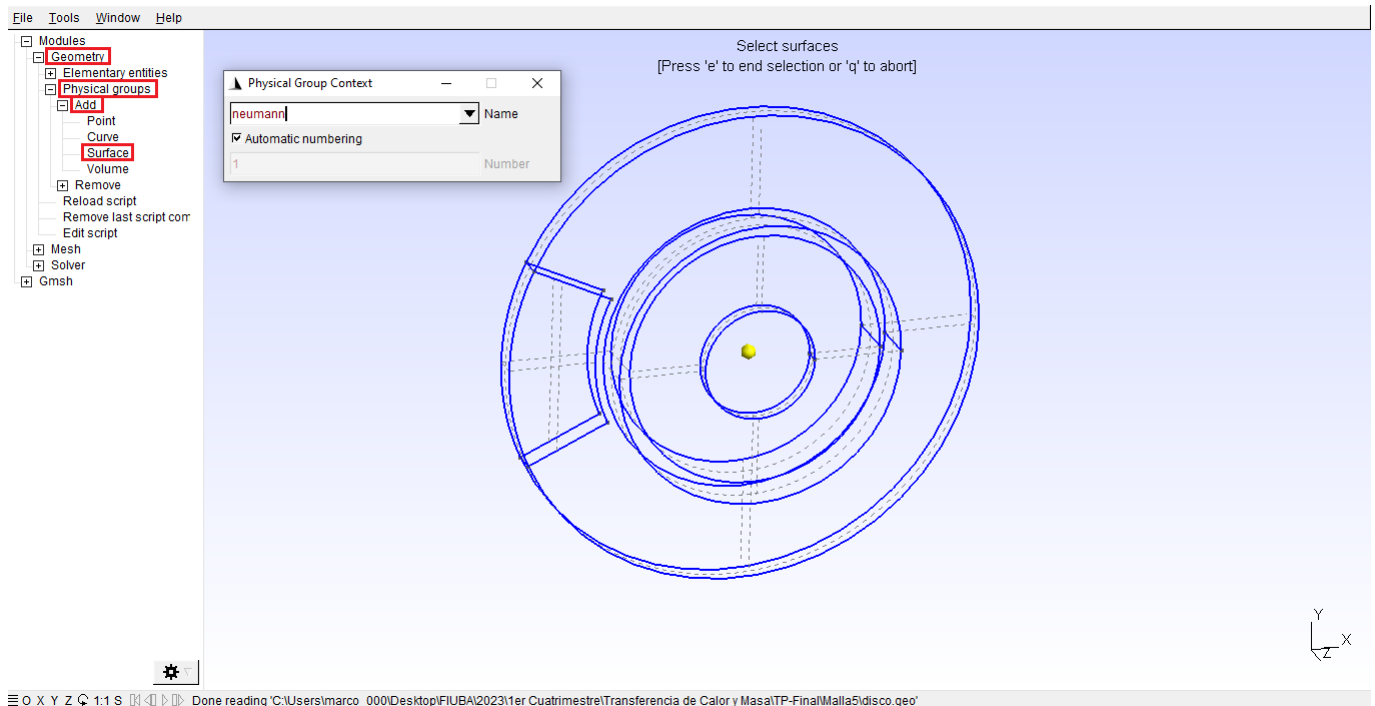


Figura 9: Captura de pantalla del **gmsh** con los pasos a seguir para definir los **Physical groups** (en este caso, las superficies denominadas *neumann*)

En este caso, se define el grupo *neumann* al cual pertenecen las superficies donde se aplicará la condición de borde

de Neumann (el calor generado por fricción). Se definieron también otros grupos como *conveccion* (las superficies expuestas al aire exterior), *aislado* (el resto de las superficies, que se consideran aisladas) y *dominio* (el volumen de todo el disco) que no serán utilizados explícitamente para aplicar condiciones de borde, pero sirven para tener un espacio de trabajo mas organizado.

El siguiente paso consiste en configurar los parámetros de la malla. Haciendo doble-click en cualquier lado de la pantalla donde se visualiza el objeto 3D, y pulsando **all mesh options**, se llega a la ventana mostrada en la Figura siguiente:

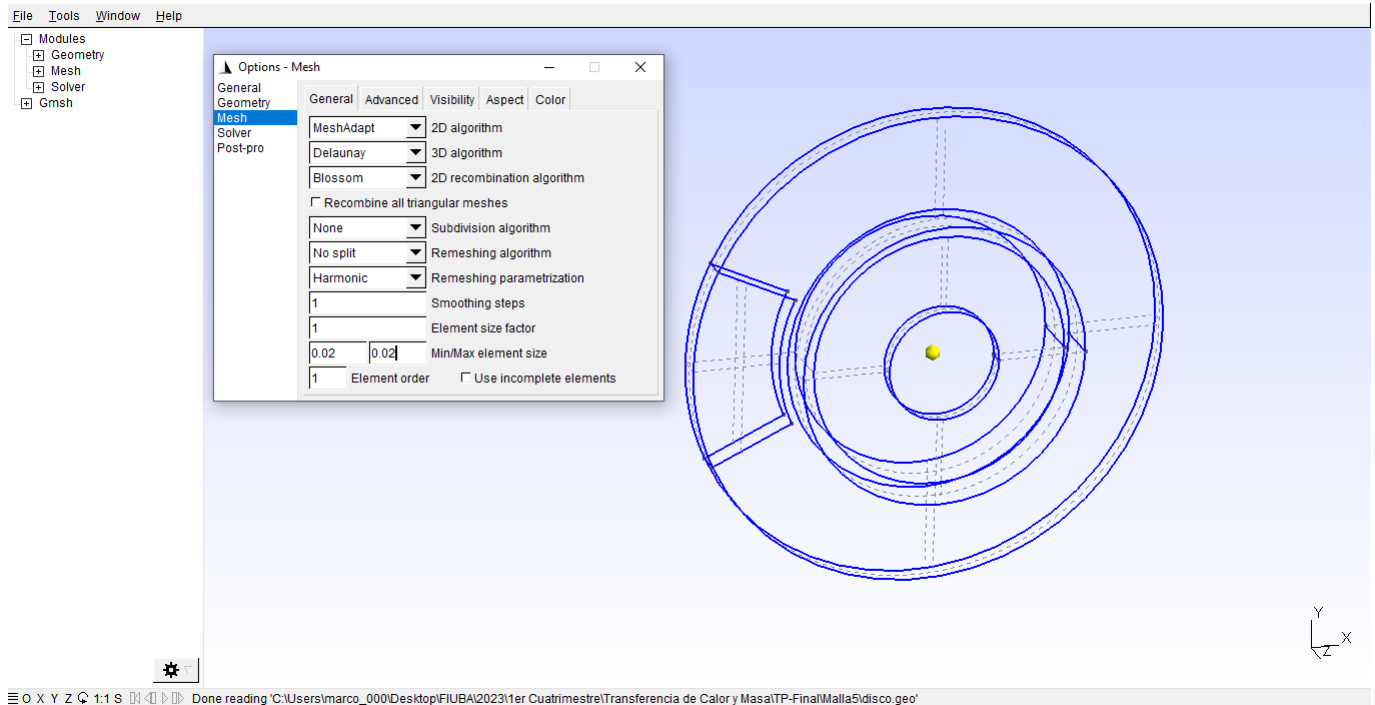


Figura 10: Captura de pantalla del **gmsh** mostrando la ventana de opciones de la malla

Se utilizará una malla compuesta por elementos en dos y tres dimensiones. Se pueden usar elementos tetraédricos (3D) y triangulares (2D) o hexaédricos (3D) y cuadrangulares (2D). Los elementos hexaédricos y cuadrangulares tienen la ventaja de lograr mallas más estructuradas, (por lo tanto, al tener un jacobiano aproximadamente constante, se reduce el error al integrar numéricamente) y además, pueden representar la variación del flujo de calor dentro de cada elemento. En cambio, los tetraedros y los triángulos forman mallas más irregulares, y si se utilizan elementos de orden lineal, no pueden representar la variación del flujo de calor dentro de cada elemento, por lo que se requiere mayor cantidad de elementos para una correcta resolución. Además el uso de elementos tetragonales presenta dificultades adicionales al integrar numéricamente.

Pero, en este caso, al tener una geometría del tipo cilíndrica, los elementos hexaédricos y cuadrangulares no logran copiar adecuadamente la geometría. Se necesitaría un número inmenso de elementos de este tipo para representar adecuadamente esta geometría. En cambio, los tetraedros y los triángulos no tienen este problema, por lo tanto, se opta por una malla no estructurada de este tipo de elementos.

Se selecciona el algoritmo **MeshAdapt** para mallar los triángulos, **Delaunay** para los tetraedros y un tamaño de elemento de 0,02 m, es decir, un tamaño relativo de:  $0,067 D$ , siendo  $D = 300$  mm el diámetro externo.

Una vez definidos los parámetros, se realiza el procedimiento siguiente:

**Mesh** → **1D** → **2D** → **Smooth 2D** (opcional) → **3D** → **Optimize 3D** (opcional) → **Optimize 3D (Netgen)** (opcional)

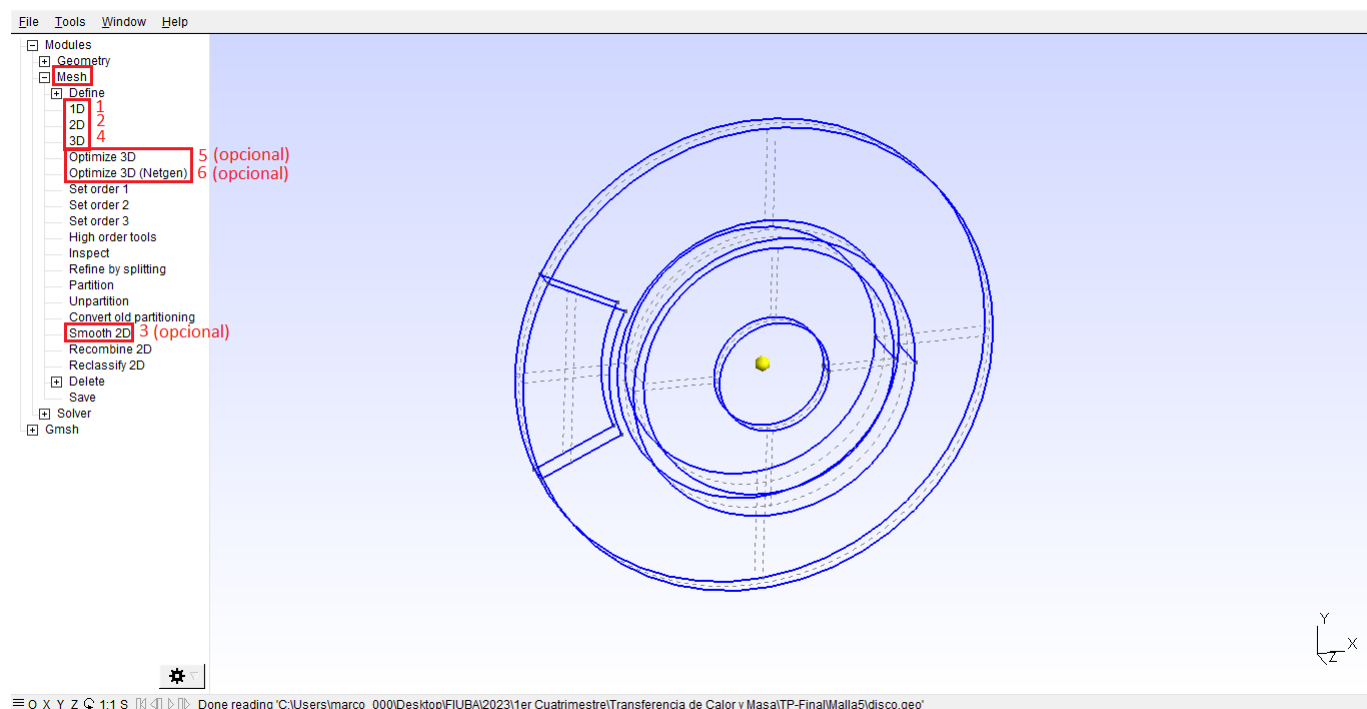


Figura 11: Captura de pantalla del **gmsh** mostrando los pasos a seguir para realizar la malla

Siguiendo estos pasos, se obtiene la malla que se muestra en la Figura 12, donde además se pulsó **Tools** → **Statistics** para ver la cantidad de nodos, elementos triangulares y tetraédricos.

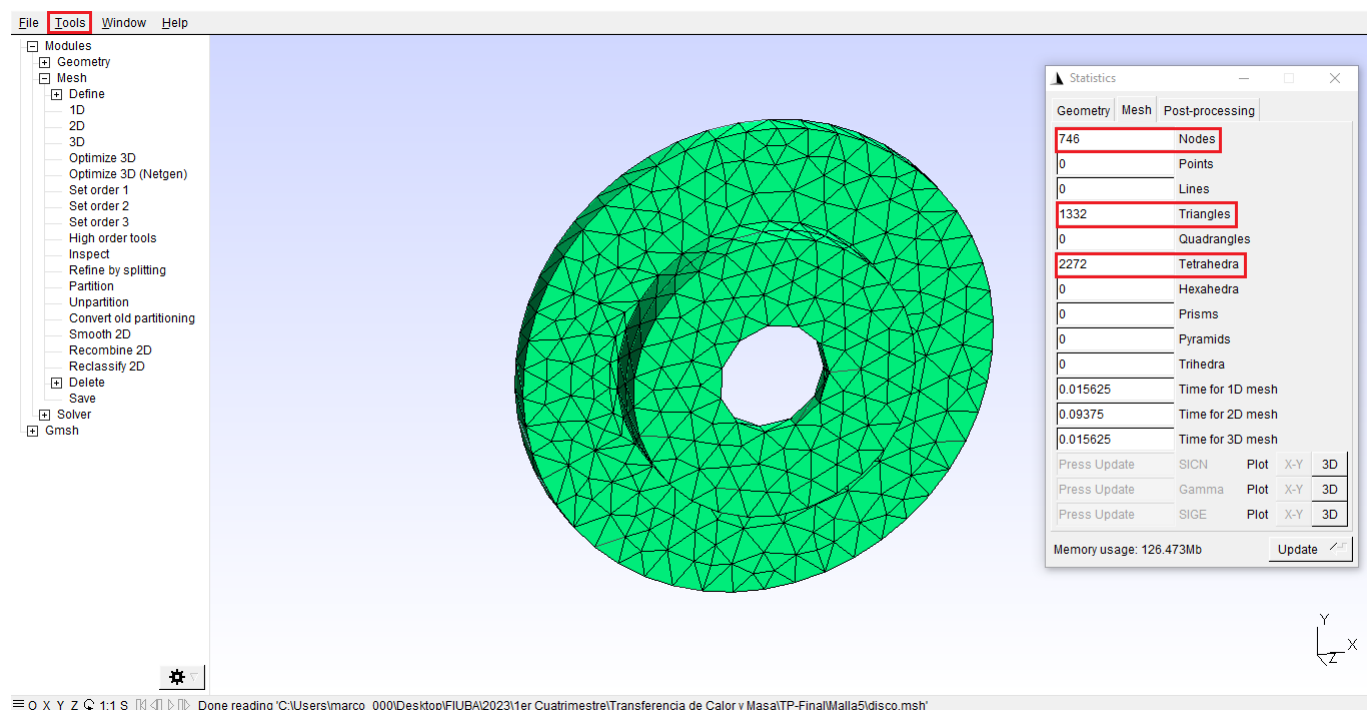


Figura 12: Malla obtenida con sus estadísticas

Finalmente, se exporta la malla en formato **.msh**. Este formato es básicamente un archivo de texto organizado de la siguiente manera:

```

1 $MeshFormat
2 2.2 0 8
3 $EndMeshFormat
4 $PhysicalNames
5 {Cantidad de Physical groups (n)}
6 {numero y nombre del Physical group 1}

```

```

7 {numero y nombre del Physical group 2}
8     {...}
9 {numero y nombre del Physical group n}
10 $EndPhysicalNames
11 $Nodes
12 {Cantidad de nodos (Nn)}
13 1 {x1} {y1} {z1}
14 2 {x2} {y2} {z2}
15     {...}
16 {Nn} {xNn} {yNn} {zNn}
17 $EndNodes
18 $Elements
19 {Cantidad de elementos (Ne)}
20 {e} {Tipo de e} {Num de ctes} {Num de Pg} {Num de superficie/volumen} {Nodos de e}
21 {Hay Ne filas con la informacion anterior en cada columna}
22 $EndElements

```

Todo lo que está entre {llaves} es reemplazado por su correspondiente significado, donde,

- e: número de elemento
- Tipo de e: tipo de elemento (2: triangulo de 3 nodos, 4: tetraedro de 4 nodos)
- Num de ctes: número de constantes (siempre son 2, no es importante)
- Num de Pg: número del Physical group
- Num de superficie/volumen: número de la superficie/volumen a la cual pertenece el elemento (no es importante)
- Nodos de e: números de los nodos que pertenecen al elemento (tres columnas para los elementos triangulares y 4 para los tetraedros)

La información importante que hay que extraer de este archivo es:

- Los números y nombres de los **Physical groups**

La idea es definir un diccionario que asocie cada numero de **Physical group** a su respectivo número.

$$\text{physicalGroups} = \{1 : \text{neumann}, 2 : \text{conveccion}, 3 : \text{aislado}, 4 : \text{dominio}\}$$

- Las coordenadas de los nodos

Se busca obtener una matriz que contenga el numero y las coordenadas de cada nodo, donde la fila  $n$  corresponde al nodo  $n$ :

$$[X] = \begin{bmatrix} 1 & x_1 & y_1 & z_1 \\ 2 & x_2 & y_2 & z_2 \\ \vdots & \vdots & \vdots & \vdots \\ n & x_n & y_n & z_n \\ \vdots & \vdots & \vdots & \vdots \\ N_n & x_{N_n} & y_{N_n} & z_{N_n} \end{bmatrix}$$

- La conectividad entre los nodos y los elementos 2D y 3D, junto con los **Physical groups** de los elementos triangulares

Se buscan obtener matrices como las siguientes, donde la fila  $e$  corresponde al elemento  $e$ :

$$[MC2D] = \begin{bmatrix} 1 & PG_1 & \text{nodo}_1^{(1)} & \text{nodo}_2^{(1)} & \text{nodo}_3^{(1)} \\ 2 & PG_2 & \text{nodo}_1^{(2)} & \text{nodo}_2^{(2)} & \text{nodo}_3^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ e & PG_e & \text{nodo}_1^{(e)} & \text{nodo}_2^{(e)} & \text{nodo}_3^{(e)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ N_{e2D} & PG_{e2D} & \text{nodo}_1^{(N_{e2D})} & \text{nodo}_2^{(N_{e2D})} & \text{nodo}_3^{(N_{e2D})} \end{bmatrix}$$

$$[MC3D] = \begin{bmatrix} \text{nodo}_1^{(1)} & \text{nodo}_2^{(1)} & \text{nodo}_3^{(1)} & \text{nodo}_4^{(1)} \\ \text{nodo}_1^{(2)} & \text{nodo}_2^{(2)} & \text{nodo}_3^{(2)} & \text{nodo}_4^{(2)} \\ \vdots & \vdots & \vdots & \vdots \\ \text{nodo}_1^{(e)} & \text{nodo}_2^{(e)} & \text{nodo}_3^{(e)} & \text{nodo}_4^{(e)} \\ \vdots & \vdots & \vdots & \vdots \\ \text{nodo}_1^{(N_{e3D})} & \text{nodo}_2^{(N_{e3D})} & \text{nodo}_3^{(N_{e3D})} & \text{nodo}_4^{(N_{e3D})} \end{bmatrix}$$

Todo esto se logra con el siguiente código:

```

1 #1. PREPROCESSING
2 #En esta seccion, se realiza el tratamiento de los datos de la malla
3 #El objetivo es extraer las coordenadas de los nodos, la conectividad entre
4 #nodos y elementos y los Physical groups
5 #Se abre el archivo de la malla exportado del gmsh en modo lectura
6 with open('Malla5//disco.msh','r') as m:
7     #Se saltan las primeras 4 lineas
8     for i in range(4):
9         m.readline()
10    #Se obtiene la cantidad de Physical Groups
11    nPhysicalGroups = int(m.readline())
12    #Se crea un diccionario vacio de los Physical Groups
13    physicalGroups = {}
14    #Se recorren los Physical Groups
15    for i in range(nPhysicalGroups):
16        #Se lee y se separa cada linea
17        pg = m.readline().split()
18        #Se convierte el numero identificador del Physical Group (str) a int
19        idPhysGroup = int(pg[1])
20        #Se asocia cada numero identificador a su respectivo nombre
21        physicalGroups[idPhysGroup] = pg[2].replace('"', '')
22    #Se saltan 2 lineas
23    m.readline()
24    m.readline()
25    #Numero de nodos
26    nNodos = int(m.readline())
27    #Se crea ahora una matriz de coordenadas globales de los nodos de la forma
28    # [X] = [[1,x1,y1,z1],
29    #         [2,x2,y2,z2],
30    #         [3,x3,y3,z3],
31    #         [...],
32    #         [n,xn,yn,zn]]
33    X = np.loadtxt(m,max_rows=nNodos)
34    #Se saltan 2 lineas
35    m.readline()
36    m.readline()
37    #Numero de elementos totales
38    nElementos = int(m.readline())
39    #Numero de elementos triangulares (sale de statistics en gmsh)
40    nElementos2D = 1332 #triangulos (de 3 nodos)
41    #Numero de elementos tetraedricos (sale de statistics en gmsh)
42    nElementos3D = 2272 #tetraedros (de 4 nodos)
43    #Se almacena la informacion de los elementos en un array (conectividad)
44    MC2D = np.loadtxt(m,dtype=int,max_rows=nElementos2D)
45    MC3D = np.loadtxt(m,dtype=int,max_rows=nElementos3D)
46    #Estos arrays contienen la informacion siguiente:
47    #Columna 0: numero de elemento
48    #Columna 1: numero del tipo de elemento (2: triangulo de 3 nodos,
49    #4: tetraedro de 4 nodos)
50    #Columna 2: numero de ctes (siempre son 2, no es importante)
51    #Columna 3: numero de physical group
52    #Columna 4: numero de la superficie/volumen a la cual pertenece el
53    #elemento (no es importante)
54    #Columna 5+: numeros de los nodos que pertenecen al elemento
55
56    #Se crea una matriz que contenga unicamente los nodos de cada elemento 3D
57    #Es decir, se filtra la matriz MC3D, de manera que queden unicamente las
58    #ultimas 4 columnas
59    MC3D = np.delete(MC3D,[0,1,2,3,4],axis=1)

```

## 3.2. Procesado

En esta sección se detalla, paso a paso, todo el procedimiento realizado para programar y resolver la ecuación diferencial mediante el método de elementos finitos.

### 3.2.1. Funciones de forma

Lo primero que hay que definir son las funciones de forma ( $\phi$ ) utilizadas para interpolar los resultados dentro de cada elemento. Se debe cumplir siempre que:

$$\begin{cases} \phi_i = 1 & \text{en el nodo } i \\ \phi_i = 0 & \text{en los otros nodos} \end{cases}$$

Siguiendo las recomendaciones de la bibliografía (Bathe, [Finite Element Procedures](#)), se definen las siguientes funciones de forma para elementos tetraédricos lineales (de 4 nodos):

$$\begin{cases} \phi_1 = 1 - \eta - \xi - \tau \\ \phi_2 = \eta \\ \phi_3 = \xi \\ \phi_4 = \tau \end{cases}$$

Siendo  $\{\eta, \xi, \tau\}$  las coordenadas locales de cada elemento, que varían de 0 a 1, y los nodos 1, 2, 3, 4 locales de cada elemento definidos como se muestra en la Figura 13.

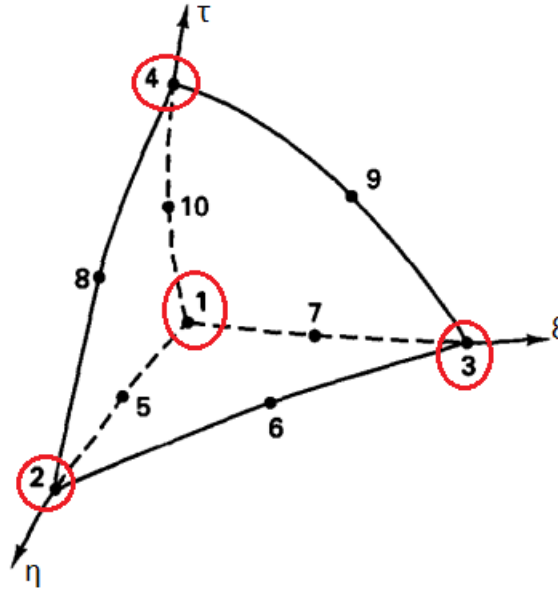


Figura 13: Esquema de un elemento tetraédrico con los nodos de interés remarcados. El resto no son tenidos en cuenta en este análisis ya que se utilizarán elementos lineales

Con estas funciones de forma, se define el siguiente vector:

$$\underline{\Phi} = [\phi_1 \quad \phi_2 \quad \phi_3 \quad \phi_4]$$

En el código, se define entonces una función que devuelva el vector con las funciones de forma, para los valores de  $\{\eta, \xi, \tau\}$  introducidos.

```

1 #2. PROCESSING
2 #En esta seccion, se aplica el metodo de elementos finitos para la resolucion del problema
3 #Funciones de forma
4 #Toma como argumentos las coordenadas convectivas de los elementos tetragonales
5 def funcion_F(eta,xi,tau):
6     #Funciones de forma para elementos tetragonales
7     phi1 = 1 - eta - xi - tau
8     phi2 = eta
9     phi3 = xi
10    phi4 = tau
11    #Vector de funciones de forma
12    Phi = np.array([phi1,phi2,phi3,phi4])
13    return Phi

```

### 3.2.2. Derivadas de las funciones de forma

Una vez definidas las funciones de forma, se deben hallar sus derivadas respecto de las coordenadas locales.

$$\begin{array}{cccc} \frac{\partial \phi_1}{\partial \eta} = -1 & \frac{\partial \phi_2}{\partial \eta} = 1 & \frac{\partial \phi_3}{\partial \eta} = 0 & \frac{\partial \phi_4}{\partial \eta} = 0 \\ \frac{\partial \phi_1}{\partial \xi} = -1 & \frac{\partial \phi_2}{\partial \xi} = 0 & \frac{\partial \phi_3}{\partial \xi} = 1 & \frac{\partial \phi_4}{\partial \xi} = 0 \\ \frac{\partial \phi_1}{\partial \tau} = -1 & \frac{\partial \phi_2}{\partial \tau} = 0 & \frac{\partial \phi_3}{\partial \tau} = 0 & \frac{\partial \phi_4}{\partial \tau} = 1 \end{array}$$

Se arma la matriz siguiente:

$$\underbrace{[\partial\Phi]}_{3 \times 4} = \begin{bmatrix} \frac{\partial\phi_1}{\partial\eta} & \frac{\partial\phi_2}{\partial\eta} & \frac{\partial\phi_3}{\partial\eta} & \frac{\partial\phi_4}{\partial\eta} \\ \frac{\partial\phi_1}{\partial\xi} & \frac{\partial\phi_2}{\partial\xi} & \frac{\partial\phi_3}{\partial\xi} & \frac{\partial\phi_4}{\partial\xi} \\ \frac{\partial\phi_1}{\partial\tau} & \frac{\partial\phi_2}{\partial\tau} & \frac{\partial\phi_3}{\partial\tau} & \frac{\partial\phi_4}{\partial\tau} \end{bmatrix} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ -1 & 0 & 0 & 1 \end{bmatrix}$$

```

1 #Derivadas de las funciones de forma
2 #Toma como argumentos las coordenadas convectivas de los elementos tetragonales
3 def funcion_∂Φ(η,ξ,τ):
4     #Derivadas para los elementos tetragonales de 4 nodos
5     ϕ1_η, ϕ1_ξ, ϕ1_τ = -1, -1, -1
6     ϕ2_η, ϕ2_ξ, ϕ2_τ = 1, 0, 0
7     ϕ3_η, ϕ3_ξ, ϕ3_τ = 0, 1, 0
8     ϕ4_η, ϕ4_ξ, ϕ4_τ = 0, 0, 1
9     #Matriz gradiente de las funciones de forma
10    ∂Φ = np.array([[ϕ1_η, ϕ2_η, ϕ3_η, ϕ4_η],
11                  [ϕ1_ξ, ϕ2_ξ, ϕ3_ξ, ϕ4_ξ],
12                  [ϕ1_τ, ϕ2_τ, ϕ3_τ, ϕ4_τ]])
13    return ∂Φ

```

### 3.2.3. Matriz jacobiana

Las coordenadas globales se interpolan en cada elemento mediante las funciones de forma definidas previamente:

$$\begin{cases} x^{(e)} = \phi_1 x_1^{(e)} + \phi_2 x_2^{(e)} + \phi_3 x_3^{(e)} + \phi_4 x_4^{(e)} \\ y^{(e)} = \phi_1 y_1^{(e)} + \phi_2 y_2^{(e)} + \phi_3 y_3^{(e)} + \phi_4 y_4^{(e)} \\ z^{(e)} = \phi_1 z_1^{(e)} + \phi_2 z_2^{(e)} + \phi_3 z_3^{(e)} + \phi_4 z_4^{(e)} \end{cases}$$

Las temperaturas que se obtendrán en los nodos, se interpolan dentro de cada elemento de la misma manera:

$$T^{(e)} = \phi_1 T_1^{(e)} + \phi_2 T_2^{(e)} + \phi_3 T_3^{(e)} + \phi_4 T_4^{(e)} \quad (6)$$

Se sabe por la ley de Fourier, que el flujo de calor es:

$$\underline{q} = -k \underline{\nabla} T$$

El operador gradiente actúa sobre las coordenadas globales  $(\{x, y, z\})$ , por lo que, para calcularlo, se plantea la regla de la cadena derivando respecto de las coordenadas locales:

$$\begin{cases} \frac{\partial T}{\partial x} = \frac{\partial T}{\partial \eta} \frac{\partial \eta}{\partial x} + \frac{\partial T}{\partial \xi} \frac{\partial \xi}{\partial x} + \frac{\partial T}{\partial \tau} \frac{\partial \tau}{\partial x} \\ \frac{\partial T}{\partial y} = \frac{\partial T}{\partial \eta} \frac{\partial \eta}{\partial y} + \frac{\partial T}{\partial \xi} \frac{\partial \xi}{\partial y} + \frac{\partial T}{\partial \tau} \frac{\partial \tau}{\partial y} \\ \frac{\partial T}{\partial z} = \frac{\partial T}{\partial \eta} \frac{\partial \eta}{\partial z} + \frac{\partial T}{\partial \xi} \frac{\partial \xi}{\partial z} + \frac{\partial T}{\partial \tau} \frac{\partial \tau}{\partial z} \end{cases}$$

Las derivadas de  $T$  respecto de las coordenadas locales  $\{\eta, \xi, \tau\}$  se obtienen derivando las funciones de forma en la ecuación (6). Las derivadas de las coordenadas locales  $\{\eta, \xi, \tau\}$  respecto de las globales  $\{x, y, z\}$  se obtendrán de la matriz jacobiana inversa.

Se calcula entonces primero la matriz jacobiana:

$$\underbrace{[J^{(e)}]}_{3 \times 3} = \begin{bmatrix} \frac{\partial x^{(e)}}{\partial \eta} & \frac{\partial y^{(e)}}{\partial \eta} & \frac{\partial z^{(e)}}{\partial \eta} \\ \frac{\partial x^{(e)}}{\partial \xi} & \frac{\partial y^{(e)}}{\partial \xi} & \frac{\partial z^{(e)}}{\partial \xi} \\ \frac{\partial x^{(e)}}{\partial \tau} & \frac{\partial y^{(e)}}{\partial \tau} & \frac{\partial z^{(e)}}{\partial \tau} \end{bmatrix}$$

Donde cada derivada es:

$$\frac{\partial Z^\alpha}{\partial \theta^i} = \sum_{j=1}^4 \frac{\partial \phi_j}{\partial \theta^i} Z_j^\alpha \quad , \quad \alpha = 1, 2, 3 \quad , \quad i = 1, 2, 3$$

Siendo  $\{Z^1, Z^2, Z^3\} = \{x, y, z\}$  y  $\{\theta^1, \theta^2, \theta^3\} = \{\eta, \xi, \tau\}$ .

Utilizando la matriz de derivadas de las funciones de forma y la matriz de coordenadas globales de los nodos, se calcula la matriz jacobiana.

```

1 #Matriz jacobiana elemental de la transformacion de coordenadas cartesianas (globales) a las
  convectivas (locales)
2 #Toma como argumentos:
3 #X: matriz de coordenadas de los nodos
4 #MC3De: fila (e) de la matriz de conectividades 3D (hexaedros) -> elemento (e)
5 #∂Φ: matriz de derivadas de las funciones de forma
6 def funcion_J(X,MC3De,∂Φ):
7     #Nodos x elemento tetragonal
8     nodosXelemento3D = 4
9     #Coordenadas globales de los nodos del elemento e
10    Xe = np.zeros((nodosXelemento3D,4))
11    for i in range(nodosXelemento3D):
12        Xe[i] = X[int(MC3De[i]-1)]
13    #Xe = [[1,x1,y1,z1],
14           #       [2,x2,y2,z2],
15           #       [3,x3,y3,z3],
16           #       [4,x4,y4,z4]] del elemento (e)
17    #Derivada de (x,y,z) respecto de (η,ξ,τ)
18    #Inicializacion de las derivadas de x
19    x_η, x_ξ, x_τ = 0, 0, 0
20    #Inicializacion de las derivadas de y
21    y_η, y_ξ, y_τ = 0, 0, 0
22    #Inicializacion de las derivadas de z
23    z_η, z_ξ, z_τ = 0, 0, 0
24    #Sumatoria
25    for i in range(nodosXelemento3D):
26        #Derivadas de x
27        x_η += ∂Φ[0][i]*Xe[i][1]
28        x_ξ += ∂Φ[1][i]*Xe[i][1]
29        x_τ += ∂Φ[2][i]*Xe[i][1]
30        #Derivadas de y
31        y_η += ∂Φ[0][i]*Xe[i][2]
32        y_ξ += ∂Φ[1][i]*Xe[i][2]
33        y_τ += ∂Φ[2][i]*Xe[i][2]
34        #Derivadas de z
35        z_η += ∂Φ[0][i]*Xe[i][3]
36        z_ξ += ∂Φ[1][i]*Xe[i][3]
37        z_τ += ∂Φ[2][i]*Xe[i][3]
38    #Matriz jacobiana
39    matrizJ = np.array([[x_η,y_η,z_η],
40                        [x_ξ,y_ξ,z_ξ],
41                        [x_τ,y_τ,z_τ]])
42    return matrizJ

```

### 3.2.4. Gradiente de las funciones de forma

Tal como se explicó previamente, se necesitan las derivadas de las funciones de forma respecto de las coordenadas globales para poder calcular el flujo de calor. Se define entonces la matriz gradiente de las funciones de forma de cada elemento como:

$$\underbrace{[B^{(e)}]}_{3 \times 4} = \underbrace{[J^{(e)}]^{-1}}_{3 \times 3} \cdot \underbrace{[\partial\Phi]}_{3 \times 4} = [\nabla\Phi^{(e)}]$$

Esto no es más que derivar aplicando la regla de la cadena:  $[J^{(e)}]^{-1}$  contiene las derivadas de las coordenadas locales  $\{\eta, \xi, \tau\}$  respecto de las globales  $\{x, y, z\}$  y  $[\partial\Phi]$  contiene las derivadas de las funciones de forma respecto de las coordenadas locales.

Definiendo esta matriz, el flujo de calor en cada elemento resulta:

$$\underbrace{q^{(e)}}_{3 \times 1} = -k \underbrace{[B^{(e)}]}_{3 \times 4} \cdot \underbrace{T^{(e)}}_{4 \times 1} = \begin{bmatrix} q_x^{(e)} \\ q_y^{(e)} \\ q_z^{(e)} \end{bmatrix}$$

Donde  $T^{(e)}$  es el vector columna de temperaturas de los nodos del elemento (e).

```

1 #Matriz B (gradiente de las funciones de forma)
2 #Toma como argumentos:
3 #matrizJ: matriz jacobiana
4 #∂Φ: matriz gradiente de las funciones de forma
5 def funcion_B(matrizJ,∂Φ):
6     #Inversa de la matriz jacobiana
7     matrizJ_inv = np.linalg.inv(matrizJ)
8     #Matriz B
9     B = np.matmul(matrizJ_inv,∂Φ)
10    return B

```



### 3.2.5. Formulación de Petrov-Galerkin

Para los problemas de convección-difusión con grandes velocidades, como este caso, conviene usar la formulación de Petrov-Galerkin que mejora la estabilidad de la solución considerablemente.

#### ■ Vector de pesos de Petrov

Se comienza definiendo el número adimensional de Peclet de cada elemento, que mide la influencia de los efectos convectivos respecto de los difusivos en cada dirección:

$$\text{Pe}_i^{(e)} = \frac{v_i^{(e)} l_i^{(e)}}{2\alpha^{(e)}} \quad , \quad i = x, y, z$$

Siendo la difusividad térmica del elemento:  $\alpha^{(e)} = \frac{k^{(e)}}{\rho^{(e)} c_p^{(e)}}$ ,  $v_i^{(e)}$  su velocidad en la dirección  $i$  y  $l_i^{(e)}$  su longitud característica en la dirección  $i$ , tal como se muestra en la Figura 14.

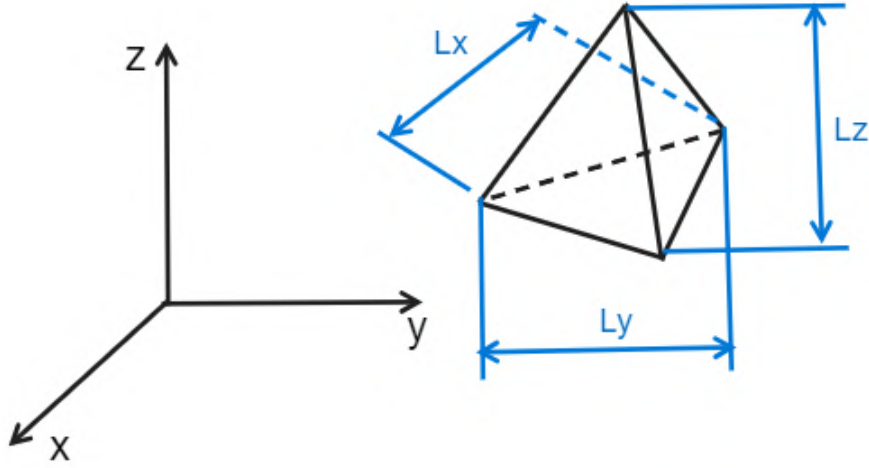


Figura 14: Longitudes características en cada dirección

Para calcular las longitudes características de cada elemento, se toma la máxima diferencia de las coordenadas en cada dirección:

$$\begin{cases} l_x^{(e)} = \max \left\{ |x_i^{(e)} - x_j^{(e)}| \right\} & i, j = 1, 2, 3, 4 \\ l_y^{(e)} = \max \left\{ |y_i^{(e)} - y_j^{(e)}| \right\} & i, j = 1, 2, 3, 4 \\ l_z^{(e)} = \max \left\{ |z_i^{(e)} - z_j^{(e)}| \right\} & i, j = 1, 2, 3, 4 \end{cases}$$

Las velocidades son:

$$\begin{cases} v_x^{(e)}(\eta, \xi, \tau) = -\omega y^{(e)}(\eta, \xi, \tau) \\ v_y^{(e)}(\eta, \xi, \tau) = \omega x^{(e)}(\eta, \xi, \tau) \\ v_z^{(e)} = 0 \end{cases}$$

Luego, se calcula un coeficiente auxiliar:

$$a_i^{(e)} = \coth |\text{Pe}_i^{(e)}| - \frac{1}{|\text{Pe}_i^{(e)}|}$$

Con este coeficiente, se calcula el tiempo intrínseco:

$$\tau_{in}^{(e)} = \sum_{j=1}^{N_{\text{dim}}} \frac{a_j^{(e)} l_j^{(e)}}{2 \|\mathbf{v}\|^{(e)}}$$

En este caso,  $N_{\text{dim}} = 3$  (dimensiones del problema). Luego, se calcula el vector de pesos de Petrov:

$$\underbrace{W^{(e)}}_{1 \times 4} = \tau_{in}^{(e)} \underbrace{\mathbf{v}^{(e)}}_{1 \times 3} \cdot \underbrace{[B]^{(e)}}_{3 \times 4}$$

```

1 #Pesos de Petrov
2 #Toma como argumentos:
3 #η,ξ,τ: coordenadas convectivas locales
4 #X: matriz de coordenadas de los nodos
5 #MC3De: fila (e) de la matriz de conectividades (tetrahedros)
6 #ω: velocidad angular
7 #α: difusividad termica
8 def funcion_W(η,ξ,τ,X,MC3De,ω,α):
9     if ω == 0:
10         W = np.zeros(4)
11     else:
12         nodosXelemento3D = 4
13         #Coordenadas gloables de los nodos del elemento e
14         Xe = np.zeros((nodosXelemento3D,4))
15         for i in range(nodosXelemento3D):
16             Xe[i] = X[int(MC3De[i]-1)]
17         #Xe = [[nodo1(e), x1(e), y1(e), z1(e)],
18             #     [nodo2(e), x2(e), y2(e), z2(e)],
19             #     [nodo3(e), x3(e), y3(e), z3(e)],
20             #     [nodo4(e), x4(e), y4(e), z4(e)]]
21         #Se obtienen el vector de las funciones de forma y sus derivadas
22         Φ, ∂Φ = funcion_Φ(η,ξ,τ), funcion_∂Φ(η,ξ,τ)
23         #Se calcula Φ.Xe en una variable auxiliar (aux)
24         aux = np.matmul(Φ,Xe)
25         #Coordenadas x,y del elemento (e) interpoladas
26         x, y = aux[1], aux[2]
27         #Vector velocidad
28         v = np.array([-ω*y,ω*x,0])
29         #Modulos de las componentes de la velocidad
30         vx, vy = abs(v[0]), abs(v[1])
31         #Velocidad absoluta
32         V = np.sqrt(vx**2 + vy**2)
33         #Coordenadas xy de los nodos del elemento
34         x1, y1 = Xe[0][1], Xe[0][2]
35         x2, y2 = Xe[1][1], Xe[1][2]
36         x3, y3 = Xe[2][1], Xe[2][2]
37         x4, y4 = Xe[3][1], Xe[3][2]
38         #Posibles longitudes características del elemento (e)
39         #Direccion x
40         Lx12, Lx13, Lx14 = abs(x1 - x2), abs(x1 - x3), abs(x1 - x4)
41         Lx23, Lx24 = abs(x2-x3), abs(x2-x4)
42         Lx34 = abs(x3-x4)
43         #Direccion y
44         Ly12, Ly13, Ly14 = abs(y1 - y2), abs(y1 - y3), abs(y1 - y4)
45         Ly23, Ly24 = abs(y2-y3), abs(y2-y4)
46         Ly34 = abs(y3-y4)
47         #Se toman las mayores longitudes como las características del elemento
48         Lx = max([Lx12,Lx13,Lx14,Lx23,Lx24,Lx34])
49         Ly = max([Ly12,Ly13,Ly14,Ly23,Ly24,Ly34])
50         #Numero de Peclet en cada direccion
51         Pe_x, Pe_y = vx*Lx/(2*α), vy*Ly/(2*α)
52         #Coeficientes auxiliares
53         a_x = 1/np.tanh(abs(Pe_x)) - 1/abs(Pe_x)
54         a_y = 1/np.tanh(abs(Pe_y)) - 1/abs(Pe_y)
55         #Tiempo intrinseco
56         τin = a_x*Lx/(2*V) + a_y*Ly/(2*V)
57         #Matriz jacobiana
58         matrizJ = funcion_J(X,MC3De,∂Φ)
59         #Matriz B, de dimensiones (3x4)
60         B = funcion_B(matrizJ, ∂Φ)
61         #Se calcula el vector de pesos de Petrov, dimensiones (1x4)
62         W = τin*np.matmul(v,B)
63     return W

```

### ■ Matriz de masa elemental

La matriz de masa (también llamada capacitiva) es la que se debe introducir en la formulación debido a los efectos transitorios. Se divide la matriz de masa en dos términos: la contribución de Galerkin y la de Petrov.

$$\begin{aligned}
 \underline{\underline{M}}_G^{(e)} &= \int_V \rho c_p \underbrace{\underline{\Phi}^T}_{4 \times 1} \cdot \underbrace{\underline{\Phi}}_{1 \times 4} dV \\
 \underline{\underline{M}}_P^{(e)} &= \int_V \rho c_p \underbrace{\underline{W}^{(e)T}}_{4 \times 1} \cdot \underbrace{\underline{\Phi}}_{1 \times 4} dV
 \end{aligned}$$

La matriz de masa total de cada elemento es entonces:

$$\underline{\underline{M}}^{(e)} = \underline{\underline{M}}_G^{(e)} + \underline{\underline{M}}_P^{(e)}$$

#### ■ Matriz convectiva elemental

Es la matriz que se debe introducir en la formulación debido a los efectos convectivos que surgen de plantear la formulación Euleriana. Nuevamente, se la divide en la de Galerkin y la de Petrov:

$$\begin{aligned}\underline{\underline{N}}_G^{(e)} &= \int_V \rho c_p \underbrace{\underline{\Phi}^T}_{4 \times 1} \cdot \underbrace{\underline{v}^{(e)}}_{1 \times 3} \cdot \underbrace{[B^{(e)}]}_{3 \times 4} dV \\ \underline{\underline{N}}_P^{(e)} &= \int_V \rho c_p \underbrace{\underline{W}^{(e)T}}_{4 \times 1} \cdot \underbrace{\underline{v}^{(e)}}_{1 \times 3} \cdot \underbrace{[B^{(e)}]}_{3 \times 4} dV\end{aligned}$$

La matriz convectiva total es:

$$\underline{\underline{N}}^{(e)} = \underline{\underline{N}}_G^{(e)} + \underline{\underline{N}}_P^{(e)}$$

#### ■ Matriz de rigidez elemental

La matriz de rigidez es la que se debe introducir en la formulación debido a los efectos de la conducción. En principio, esta matriz también se divide en sus dos contribuciones:

$$\begin{aligned}\underline{\underline{K}}_G^{(e)} &= \int_V k \underbrace{[B^{(e)}]^T}_{4 \times 3} \cdot \underbrace{[B^{(e)}]}_{3 \times 4} dV \\ \underline{\underline{K}}_P^{(e)} &= - \int_V \underline{W}^{(e)T} \cdot \underline{\nabla} \cdot (k[B^{(e)}]) dV\end{aligned}$$

Pero al usar elementos de orden lineal (las funciones de forma son lineales), la matriz de rigidez de Petrov se anula, por lo tanto, la matriz de rigidez total de cada elemento es:

$$\underline{\underline{K}}^{(e)} = \underline{\underline{K}}_G^{(e)}$$

#### ■ Matriz de rigidez de convección elemental

Esta matriz se debe considerar por la condición de borde de convección. Se calcula como:

$$\underline{\underline{K}}_c^{(e)} = \begin{cases} \int_{S_c} h_c^{(e)} \underbrace{\underline{\Phi}^T}_{4 \times 1} \cdot \underbrace{\underline{\Phi}}_{1 \times 4} dS^{(e)} & \text{si } e \in S_c \\ \underline{0} & \text{si } e \notin S_c \end{cases}$$

#### ■ Vector de calor de convección elemental

Este el vector de calor de cada elemento debido a la condición de borde de convección.

$$\underline{Q}_c^{(e)} = \begin{cases} \int_{S_c} h_c^{(e)} T_\infty \underline{\Phi}^T dS^{(e)} & \text{si } e \in S_c \\ \underline{0} & \text{si } e \notin S_c \end{cases}$$

Para calcular este vector y la matriz anterior, se necesita el coeficiente de convección. Éste se obtendrá a partir de la correlación de flujo forzado a lo largo de una placa plana. De la bibliografía, se obtienen las propiedades del aire en función de la temperatura.

Gas	$T$ K	$k$ W/m K	$\rho$ kg/m <sup>3</sup>	$c_p$ J/kg K	$\mu \times 10^6$ kg/m s	$\nu \times 10^6$ m <sup>2</sup> /s	Pr
Aire (PE 82 K)	150	0.0158	2.355	1017	10.64	4.52	0.69
	200	0.0197	1.767	1009	13.59	7.69	0.69
	250	0.0235	1.413	1009	16.14	11.42	0.69
	260	0.0242	1.360	1009	16.63	12.23	0.69
	270	0.0249	1.311	1009	17.12	13.06	0.69
	280	0.0255	1.265	1008	17.60	13.91	0.69
	290	0.0261	1.220	1007	18.02	14.77	0.69
	300	0.0267	1.177	1005	18.43	15.66	0.69
	310	0.0274	1.141	1005	18.87	16.54	0.69
	320	0.0281	1.106	1006	19.29	17.44	0.69
	330	0.0287	1.073	1006	19.71	18.37	0.69
	340	0.0294	1.042	1007	20.13	19.32	0.69
	350	0.0300	1.012	1007	20.54	20.30	0.69
	360	0.0306	0.983	1007	20.94	21.30	0.69
	370	0.0313	0.956	1008	21.34	22.32	0.69
	380	0.0319	0.931	1008	21.75	23.36	0.69
	390	0.0325	0.906	1009	22.12	24.42	0.69
	400	0.0331	0.883	1009	22.52	25.50	0.69
	500	0.0389	0.706	1017	26.33	37.30	0.69
	600	0.0447	0.589	1038	29.74	50.50	0.69
	700	0.0503	0.507	1065	33.03	65.15	0.70

Figura 15: Propiedades del aire a distintas temperaturas. Se remarcen las que serán utilizadas: la conductividad, la viscosidad cinemática y el número de Prandtl

Interpolando los datos, se construyen las siguientes funciones de propiedades físicas del aire en función de la temperatura:

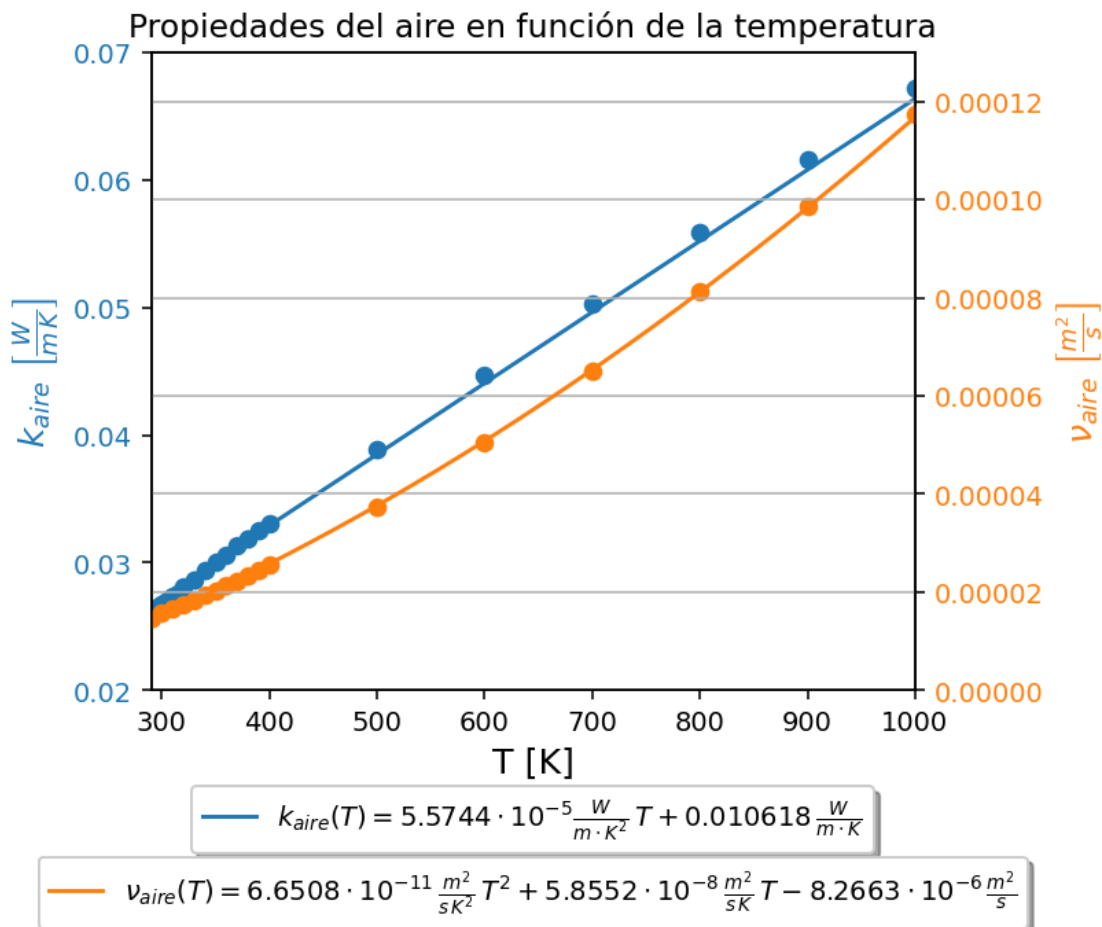


Figura 16: Conductividad térmica y viscosidad cinemática del aire en función de su temperatura

Se definen las funciones en el código:

```
1 #Propiedades termicas del aire en funcion de la temperatura
2 def funcion_k_aire(T):
3     return 5.5744e-5*T + 0.010618
4
5 def funcion_nu_aire(T):
6     return 6.6508e-11*T**2 + 5.8552e-8*T - 8.2663e-6
```

Para obtener el coeficiente de convección de cada elemento, se comienza calculando la temperatura de referencia del elemento:

$$T_R^{(e)} = \frac{T_m^{(e)} + T_\infty}{2}$$

Siendo  $T_m^{(e)}$  la temperatura media del elemento, que se obtiene como el promedio de las temperaturas de sus nodos sobre la superficie  $S_c$ :

$$T_m^{(e)} = \frac{T_1^{(e)} + T_2^{(e)} + T_3^{(e)}}{3}$$

Suponiendo que los nodos 1, 2 y 3 pertenecen a la superficie  $S_c$ .

Luego, se obtienen las propiedades del aire para esta temperatura de referencia mediante las expresiones interpoladas de la Figura 16.

$$k_{\text{aire}} = k_{\text{aire}}(T_R) \quad , \quad \nu_{\text{aire}} = \nu_{\text{aire}}(T_R)$$

Se toma la longitud característica del elemento como la raíz cuadrada de su área:

$$L^{(e)} = \sqrt{A^{(e)}}$$

El área se calcula mediante la fórmula de Herón:

$$A^{(e)} = \sqrt{s^{(e)} (s^{(e)} - a^{(e)}) (s^{(e)} - b^{(e)}) (s^{(e)} - c^{(e)})} \quad \text{con} \quad s^{(e)} = \frac{a^{(e)} + b^{(e)} + c^{(e)}}{2}$$

Siendo  $a^{(e)}, b^{(e)}, c^{(e)}$  los lados de la cara triangular del elemento sobre la superficie  $S_c$ , que se obtienen a partir de las coordenadas interpoladas  $x, y$  del elemento.

Luego, se calcula el número de Reynolds a partir de la velocidad de traslación del vehículo:

$$Re^{(e)} = \frac{V L^{(e)}}{\nu_{\text{aire}}}$$

Notar que probablemente, se tenga al principio un régimen turbulento, y medida que se reduce la velocidad, haya una transición hacia el régimen laminar.

Se calcula el número de Nusselt, evaluando si se está en régimen laminar o turbulento y sabiendo que  $Pr = 0,69$ :

$$Nu^{(e)} = \begin{cases} 0,664 Re^{1/2} Pr^{1/3} & \text{si } Re < 5 \cdot 10^5 \\ 0,664 Re^{1/2} Pr^{1/3} + 0,036 Re^{0,8} Pr^{0,43} \left[ 1 - \left( \frac{5 \cdot 10^5}{Re} \right)^{0,8} \right] & \text{si } Re \geq 5 \cdot 10^5 \end{cases}$$

Finalmente, se calcula el coeficiente de convección de cada elemento como:

$$h_c^{(e)} = \frac{k_{\text{aire}} Nu^{(e)}}{L^{(e)}}$$

```
1 #Coeficiente de conveccion
2 #Se calcula el coeficiente de conveccion hc local para cada elemento
3 #utilizando la correlacion de flujo forzado externo sobre placa plana
4 #Toma como argumentos:
5 #V: velocidad de traslacion [m/s]
6 #Te: temperatura del elemento (e) [K]
7 #T_aire: temperatura del aire ambiente [K]
8 #Le: longitud caracteristica del elemento (e) [m]
9 def funcion_hc(V,Te,T_aire,Le):
10     #Temperatura de referencia
11     TR = (Te+T_aire)/2
12     #T es la temperatura del elemento considerado
13     #Conductividad termica del aire
14     ka = funcion_k_aire(TR)
15     #Numero de Prandlt del aire
16     Pr = 0.69
17     #Viscosidad cinematica del aire
18     nu_a = funcion_nu_aire(TR)
```

```

19 #Numero de Reynolds
20 Re = V*Le/νa
21 #Numero de Nusselt
22 #Se supone flujo forzado externo a lo largo de placa plana
23 #Caso laminar:
24 if Re<5e5:
25     Nu = 0.664*Re**(1/2)*Pr**(1/3)
26 #Caso turbulento:
27 else:
28     Nu=0.664*Re**(1/2)*Pr**(1/3)+0.036*Re**0.8*Pr**0.43*(1-(5e5/Re)**0.8)
29 #Coeficiente de conveccion
30 hc = ka*Nu/Le
31 return hc

```

### ■ Vector de Neumann calor elemental

Es el vector de calor en cada elemento que surge de la aplicación de la condición de borde de Neumann en la superficie de contacto entre disco y pastilla. Éste se calcula como:

$$\underbrace{Q_q^{(e)}}_{4 \times 1} = \begin{cases} \int_{S_q} q \Phi^T dS^{(e)} & \text{si } e \in S_q \\ 0 & \text{si } e \notin S_q \end{cases}$$

Siendo  $q$  el flujo de calor cuya expresión se dedujo en la sección 2 de este informe. Es decir, este vector de calor solo será distinto de cero para los elementos que pertenezcan a la superficie denominada *neumann* ( $S_q$ ).

### 3.2.6. Integración numérica

Para obtener todas las matrices y vectores expuestos previamente, se deben realizar integrales sobre el volumen del dominio y la superficie donde se aplica la condición de borde. Estas integrales se calcularán numéricamente mediante el método de Gauss-Legendre.

Para una función de una única variable  $F(x)$ , su integral entre  $-1$  y  $1$  se calcula de la siguiente manera:

$$\int_{-1}^1 F(x) dx = \sum_{i=1}^n w_i F(r_i)$$

Donde  $w_i$  son los pesos de Gauss,  $r_i$  son las posiciones de Gauss, y  $n$  es la cantidad de puntos de Gauss considerados.

Se pueden obtener resultados exactos al integrar polinomios, siempre y cuando se elijan suficientes puntos de Gauss, teniendo en cuenta el orden del polinomio a integrar:

$$n \geq \frac{\text{orden} + 1}{2}$$

Como en este caso las funciones de forma son lineales, casi todas las expresiones a integrar en las matrices son cuadráticas (orden=2), por lo tanto,

$$n \geq \frac{2+1}{2} = 1,5 \Rightarrow n = 2$$

Es decir, se tomarán 2 puntos de Gauss en cada dirección.

Conociendo la cantidad de puntos de Gauss, se pueden obtener los pesos y las posiciones de tablas como la Tabla 1.

$n$	$w_i$	$r_i$
1	2	0
2	1	$\pm 0,5773502692$
3	0,888888889 0,555555556	0 $\pm 0,7745966692$
4	0,6521451549 0,3478548451	$\pm 0,3399810436$ $\pm 0,8611363116$
5	0,568888889 0,4786286705 0,2369268851	0 $\pm 0,5384693101$ $\pm 0,9061798459$
6	0,1713244924 0,3607615730 0,4679139346	$\pm 0,9324695142$ $\pm 0,6612093865$ $\pm 0,2386191861$

Tabla 1: Pesos y puntos de Gauss. Se remarca la fila correspondiente a 2 puntos de Gauss

En el código, se arma una función que entregue los pesos y las posiciones de Gauss, dado un número de puntos de Gauss:

```

1 #Integración por Gauss-Legendre (n: número de puntos de Gauss)
2 #Toma como argumento el número de puntos de Gauss
3 def integralGauss(nGauss):
4     #Iniciamos los vectores de pesos y posiciones de Gauss en cero
5     w = np.zeros(int(nGauss)) #pesos
6     r = np.zeros(int(nGauss)) #posiciones
7     #Introducimos los valores de la tabla para cada n
8     if nGauss==1:
9         w[0] = 2
10        r[0] = 0
11    elif nGauss==2:
12        r[0] = np.sqrt(1/3)
13        r[1] = -np.sqrt(1/3)
14        w[0] = 1
15        w[1] = 1
16    elif nGauss==3:
17        r[0] = 0
18        r[1] = np.sqrt(3/5)
19        r[2] = -np.sqrt(3/5)
20        w[0] = 8/9
21        w[1] = 5/9
22        w[2] = 5/9
23    elif nGauss==4:
24        r[0] = np.sqrt((3-2*np.sqrt(6/5))/7)
25        r[1] = -np.sqrt((3-2*np.sqrt(6/5))/7)
26        r[2] = np.sqrt((3+2*np.sqrt(6/5))/7)
27        r[3] = -np.sqrt((3+2*np.sqrt(6/5))/7)
28        w[0] = (18+np.sqrt(30))/36
29        w[1] = (18+np.sqrt(30))/36
30        w[2] = (18-np.sqrt(30))/36
31        w[3] = (18-np.sqrt(30))/36
32    return r, w

```

Para integrales dobles, se tiene:

$$\int_{-1}^1 \int_{-1}^1 F(x, y) dV = \sum_{i=1}^n \sum_{j=1}^n w_i w_j F(r_i, r_j)$$

Y análogamente, para integrales triples,

$$\int_{-1}^1 \int_{-1}^1 \int_{-1}^1 F(x, y, z) dV = \sum_{i=1}^n \sum_{j=1}^n \sum_{k=1}^n w_i w_j w_k F(r_i, r_j, r_k)$$

El problema es que las matrices y vectores que se desean obtener contienen integrales sobre los volúmenes y superficies de cada elemento. Se deben transformar esas integrales de manera que los límites de integración sean  $-1$  y  $1$  en cada dirección. Para ello, se plantean tres transformaciones sucesivas.

#### ■ Transformación I: coordenadas globales a locales

En primer lugar, se transforman las coordenadas globales  $\{x, y, z\}$  a las locales  $\{\eta, \xi, \tau\}$  mediante las funciones de forma  $\phi_i$ .

$$\begin{cases} x^{(e)} = \phi_1(\eta, \xi, \tau) x_1^{(e)} + \phi_2(\eta, \xi, \tau) x_2^{(e)} + \phi_3(\eta, \xi, \tau) x_3^{(e)} + \phi_4(\eta, \xi, \tau) x_4^{(e)} \\ y^{(e)} = \phi_1(\eta, \xi, \tau) y_1^{(e)} + \phi_2(\eta, \xi, \tau) y_2^{(e)} + \phi_3(\eta, \xi, \tau) y_3^{(e)} + \phi_4(\eta, \xi, \tau) y_4^{(e)} \\ z^{(e)} = \phi_1(\eta, \xi, \tau) z_1^{(e)} + \phi_2(\eta, \xi, \tau) z_2^{(e)} + \phi_3(\eta, \xi, \tau) z_3^{(e)} + \phi_4(\eta, \xi, \tau) z_4^{(e)} \end{cases}$$

Siendo la matriz jacobiana la ya calculada en la sección 3.2.3. Entonces, la integral de volumen se transforma como:

$$\int_V F(x, y, z) dV = \int_0^1 \int_0^{1-\eta} \int_0^{1-\eta-\xi} G(\eta, \xi, \tau) |J|^{(e)} d\tau d\xi d\eta$$

Con:

$$G(\eta, \xi, \tau) = F(x(\eta, \xi, \tau), y(\eta, \xi, \tau), z(\eta, \xi, \tau))$$

#### ■ Transformación II: tetraedro a cubo unitario

Luego, se realiza el mapeo de la transformación del tetraedro en un cubo de lado unitario, definiendo nuevas coordenadas  $\{l, m, n\}$ :

$$\begin{cases} \eta = l m n \\ \xi = l m (1 - n) \\ \tau = l (1 - m) \end{cases}$$

La matriz jacobiana de esta transformación de coordenadas es:

$$[J_{lmn}] = \begin{bmatrix} \frac{\partial \eta}{\partial l} & \frac{\partial \xi}{\partial l} & \frac{\partial \tau}{\partial l} \\ \frac{\partial \eta}{\partial m} & \frac{\partial \xi}{\partial m} & \frac{\partial \tau}{\partial m} \\ \frac{\partial \eta}{\partial n} & \frac{\partial \xi}{\partial n} & \frac{\partial \tau}{\partial n} \end{bmatrix} = \begin{bmatrix} m n & m(1-n) & 1-m \\ l n & l(1-n) & -l \\ l m & -l m & 0 \end{bmatrix}$$

Calculando el determinante, se obtiene:

$$|J_{lmn}| = l^2 m$$

La integral de volumen pasa a ser entonces:

$$\int_V F(x, y, z) dV = \int_0^1 \int_0^1 \int_0^1 H(l, m, n) |J|^{(e)} |J_{lmn}| dl dm dn$$

Con:

$$H(l, m, n) = G(\eta(l, m, n), \xi(l, m, n), \tau(l, m, n))$$

■ Transformación III: cubo unitario a cubo unitario centrado

Finalmente, se transforma el cubo anterior, con uno de sus vértices en el origen de coordenadas  $\{l, m, n\}$  a uno centrado en un nuevo origen de coordenadas  $\{u, v, w\}$ :

$$\begin{cases} l = \frac{1+u}{2} \\ m = \frac{1+v}{2} \\ n = \frac{1+w}{2} \end{cases}$$

La matriz jacobiana correspondiente es:

$$[J_{uvw}] = \begin{bmatrix} \frac{\partial l}{\partial u} & \frac{\partial m}{\partial u} & \frac{\partial n}{\partial u} \\ \frac{\partial l}{\partial v} & \frac{\partial m}{\partial v} & \frac{\partial n}{\partial v} \\ \frac{\partial l}{\partial w} & \frac{\partial m}{\partial w} & \frac{\partial n}{\partial w} \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & \frac{1}{2} \end{bmatrix}$$

Cuyo jacobiano es:

$$|J_{uvw}| = \frac{1}{8}$$

La integral de volumen resulta ser:

$$\int_V F(x, y, z) dV = \int_{-1}^1 \int_{-1}^1 \int_{-1}^1 I(u, v, w) |J|^{(e)} |J_{lmn}| |J_{uvw}| du dv dw$$

Con:

$$I(u, v, w) = H(l(u, v, w), m(u, v, w), n(u, v, w))$$

Finalmente, se llegó a una integral con límites  $-1$  y  $1$ , por lo que se la puede calcular con los pesos y las posiciones de Gauss. Se resumen las transformaciones efectuadas en la Figura 17.

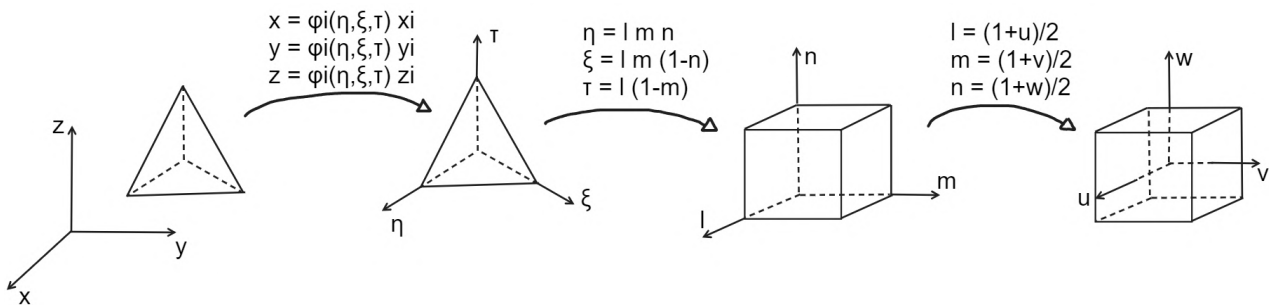


Figura 17: Transformaciones realizadas para poder integrar numéricamente con el método de Gauss-Legendre

Se procede a programar las integraciones numéricas de los volúmenes elementales para cada matriz de las estudiadas anteriormente.



```

1 #Matriz de masa local
2 #Toma como argumentos:
3 #X: matriz de coordenadas de los nodos
4 #MC3De: fila (e) de la matriz de conectividades 3D (tetraedros) -> elemento (e)
5 # $\rho_{cp}$ : densidad por el calor específico del material del disco
6 #nGauss: numero de puntos de Gauss
7 def funcion_Mlocal(X,MC3De,k, $\rho_{cp}$ ,nGauss):
8     #Cantidad de nodos x elemento tetragonal
9     nodosXelemento3D = 4
10    #Se obtienen los pesos y puntos de Gauss
11    r,w = integralGauss(nGauss)
12    #Se inicializan las matrices de masa locales
13    MGe = np.zeros((nodosXelemento3D,nodosXelemento3D))
14    MPe = np.zeros((nodosXelemento3D,nodosXelemento3D))
15    #Se recorren los puntos de Gauss
16    for a in range(nGauss):
17        for b in range(nGauss):
18            for c in range(nGauss):
19                #3er mapeo (transformacion III)
20                l = (1+r[a])/2 # u = r[a]
21                m = (1+r[b])/2 # v = r[b]
22                n = (1+r[c])/2 # w = r[c]
23                #Jacobiano de la transformacion l,m,n -> u,v,w
24                Juvw = 1/8
25                #2do mapeo (transforma el tetraedro en cubo)
26                 $\eta$  = l*m*n
27                 $\xi$  = l*m*(1-n)
28                 $\tau$  = l*(1-m)
29                #Jacobiano de la transformacion  $\eta,\xi,\tau$  -> l,m,n
30                Jlmn = abs(1**2*m)
31                #Se obtienen el vector de las funciones de forma
32                 $\Phi$  = np.array([funcion_ $\Phi$ ( $\eta,\xi,\tau$ )])
33                #Se calcula la traspuesta
34                 $\Phi^T$  =  $\Phi$ .transpose()
35                #Se obtiene la matriz de derivadas de las funciones de forma
36                 $\partial\Phi$  = funcion_ $\partial\Phi$ ( $\eta,\xi,\tau$ )
37                #Se obtiene la matriz jacobiana
38                matrizJ = funcion_J(X,MC3De, $\partial\Phi$ )
39                #Se calcula el jacobiano de la transformacion x,y,z ->  $\eta,\xi,\tau$ 
40                J = abs(np.linalg.det(matrizJ))
41                #Se calcula la matriz de masa local de Galerkin
42                MGe += w[a]*w[b]*w[c]* $\rho_{cp}$ *J*Jlmn*Juvw*np.matmul( $\Phi^T$ , $\Phi$ )
43                #Se obtiene el vector de pesos de Petrov
44                W = np.array([funcion_W( $\eta,\xi,\tau$ , X, MC3De,  $\omega$ , k)])
45                #Se calcula la traspuesta
46                WT = W.transpose()
47                #Se calcula la matriz de masa local de Petrov
48                MPe += w[a]*w[b]*w[c]* $\rho_{cp}$ *J*Jlmn*Juvw*np.matmul(WT, $\Phi$ )
49    #Matriz de masa local total
50    Me = MGe + MPe
51    return Me
52
53 #Matriz convectiva local
54 #Toma como argumentos:
55 #X: matriz de coordenadas de los nodos
56 #MC3De: fila (e) de la matriz de conectividades 3D (tetrahedros) -> elemento (e)
57 #k: conductividad termica
58 # $\rho_{cp}$ : densidad por el calor específico del disco
59 # $\omega$ : velocidad angular
60 #nGauss: numero de puntos de Gauss
61 def funcion_Nlocal(X,MC3De,k, $\rho_{cp}$ , $\omega$ ,nGauss):
62     #Cantidad de nodos x elemento tetragonal
63     nodosXelemento3D = 4
64     if  $\omega$  == 0:
65         Ne = np.zeros((nodosXelemento3D,nodosXelemento3D))
66     else:
67         #Coordenadas globales de los nodos del elemento e
68         Xe = np.zeros((4,4))
69         for i in range(4):
70             Xe[i] = X[int(MC3De[i]-1)]
71         #Se obtienen los pesos y puntos de Gauss
72         r,w = integralGauss(nGauss)
73         #Se inicializan las matrices convectivas locales
74         NGe = np.zeros((nodosXelemento3D,nodosXelemento3D))
75         NPe = np.zeros((nodosXelemento3D,nodosXelemento3D))
76         #Se recorren los puntos de Gauss
77         for a in range(nGauss):
78             for b in range(nGauss):
79                 for c in range(nGauss):

```

```

80         #3er mapeo (transformacion III)
81         l = (1+r[a])/2 # u = r[a]
82         m = (1+r[b])/2 # v = r[b]
83         n = (1+r[c])/2 # w = r[c]
84         #Jacobiano de la transformacion l,m,n -> u,v,w
85         Juvw = 1/8
86         #2do mapeo (transforma el tetraedro en cubo)
87         η = l*m*n
88         ξ = l*m*(1-n)
89         τ = l*(1-m)
90         #Jacobiano de la transformacion η,ξ,τ → l,m,n
91         Jlmn = abs(1**2*m)
92         #Se obtienen el vector de las funciones de forma
93         Φ = funcion_Φ(η,ξ,τ)
94         #Se calcula Φ.Xe en una variable auxiliar (aux)
95         aux = np.matmul(Φ,Xe)
96         #Coordenadas x,y del elemento (e) interpoladas
97         x, y = aux[1], aux[2]
98         #Vector velocidad
99         v = np.array([-ω*y,ω*x,0])
100         #Se calcula la traspuesta
101         ΦT = np.array([Φ]).transpose()
102         #Se obtiene la matriz de derivadas de las funciones de forma
103         ∂Φ = funcion_∂Φ(η,ξ,τ)
104         #Se obtiene la matriz jacobiana
105         matrizJ = funcion_J(X,MC3De,∂Φ)
106         #Se calcula el jacobiano
107         J = abs(np.linalg.det(matrizJ))
108         #Se obtiene la matriz B
109         B = funcion_B(matrizJ,∂Φ)
110         #Se calcula la matriz convectiva de Galerkin local
111         NGe+=ρcp*w[a]*w[b]*w[c]*J*Jlmn*Juvw*np.matmul(ΦT,np.array([np.matmul(v,B)]))
112         #Se obtiene el vector de pesos de Petrov
113         W = funcion_W(η,ξ,τ, X, MC3De, ω, k)
114         WT = np.array([W]).transpose()
115         #Se calcula la matriz convectiva de Petrov local
116         NPe+=ρcp*w[a]*w[b]*w[c]*J*Jlmn*Juvw*np.matmul(WT,np.array([np.matmul(v,B)]))
117         #Matriz convectiva local total
118         Ne = NGe + NPe
119         return Ne
120
121 #Matriz de rigidez local de conduccion
122 #Toma como argumentos:
123 #X: matriz de coordenadas de los nodos
124 #MC3De: fila (e) de la matriz de conectividades 3D (tetrahedros) -> elemento (e)
125 #k: conductividad termica del material del disco
126 #nGauss: numero de puntos de Gauss
127 def funcion_Klocal(X,MC3De,k,nGauss):
128     #Se obtienen los pesos y puntos de Gauss
129     r,w = integralGauss(nGauss)
130     #Cantidad de nodos x elemento tetragonal
131     nodosXelemento3D = 4
132     #Se inicializa la matriz de rigidez local
133     Ke = np.zeros((nodosXelemento3D,nodosXelemento3D))
134     #Se recorren los puntos de Gauss
135     for a in range(nGauss):
136         for b in range(nGauss):
137             for c in range(nGauss):
138                 #3er mapeo (transformacion III)
139                 l = (1+r[a])/2 # u = r[a]
140                 m = (1+r[b])/2 # v = r[b]
141                 n = (1+r[c])/2 # w = r[c]
142                 #Jacobiano de la transformacion l,m,n -> u,v,w
143                 Juvw = 1/8
144                 #2do mapeo (transforma el tetraedro en cubo)
145                 η = l*m*n
146                 ξ = l*m*(1-n)
147                 τ = l*(1-m)
148                 #Jacobiano de la transformacion η,ξ,τ -> l,m,n
149                 Jlmn = abs(1**2*m)
150                 #Se obtiene la matriz de derivadas de las funciones de forma
151                 ∂Φ = funcion_∂Φ(η,ξ,τ)
152                 #Se obtiene la matriz jacobiana
153                 matrizJ = funcion_J(X,MC3De,∂Φ)
154                 #Se calcula el jacobiano
155                 J = abs(np.linalg.det(matrizJ))
156                 #Se obtiene la matriz B
157                 B = funcion_B(matrizJ,∂Φ)
158                 #Se calcula la traspuesta de B

```

```

159     BT = np.transpose(B)
160     #Se calcula la matriz de rigidez local por conduccion
161     Ke += w[a]*w[b]*w[c]*k*J*Jlmn*Juvw*np.matmul(BT,B)
162     return Ke

```

Para las integrales de superficie, se plantean las mismas transformaciones en dos dimensiones, pero se tiene un inconveniente adicional, tal como se verá en la sección siguiente.

Para todos los casos que se analizarán a continuación, el jacobiano de la transformación II en dos dimensiones es:  $|J_{lm}| = l$ . Y la transformación III es:

$$\begin{cases} l = \frac{1+u}{2} \\ m = \frac{1+v}{2} \end{cases}$$

Con jacobiano:  $|J_{uv}| = \frac{1}{4}$

### 3.2.7. Aplicación de las condiciones de borde

Para aplicar las condiciones de borde de convección y de Neumann, lo primero que se debe hacer es identificar los elementos tridimensionales que pertenezcan a las superficies  $S_c$  y  $S_q$  respectivamente. Para ello, se recorren los elementos bidimensionales (triangulares) y se identifican los que pertenecen a  $S_c$  o  $S_q$  mediante el número del **Physical group** (que se encuentra en la matriz de conectividades  $[MC2D]$ ) asociado al nombre del grupo: *convection* o *neumann* respectivamente. Una vez identificado el elemento triangular perteneciente al grupo correspondiente (*convection* o *neumann*), se recorren los elementos tridimensionales (tetraedros), y se comparan los nodos del tetraedro con los del triángulo seleccionado. Si 3 de los nodos del tetraedro son coincidentes con los 3 nodos del triángulo, quiere decir que el elemento tetraédrico pertenece a la superficie en cuestión, ya sea  $S_c$  o  $S_q$ .

Se sabe que las superficies de *convección* y de *neumann* están sobre planos de  $z = \text{cte}$ , por lo tanto, se elimina la columna 3 de la matriz jacobiana de la transformación de las coordenadas globales a las locales (Transformación I), porque las derivadas de  $z$  son nulas. El problema es que la definición de las funciones de forma, y todos los cálculos que siguieron a partir de ese punto deben ser coherentes con la numeración de los nodos de cada elemento, tal como se muestra en la Figura 13, pero al ser un mallado no estructurado, no hay forma de predecir cuáles son los 3 nodos del tetraedro que pertenecen a la superficie en cuestión. Por lo tanto, se preparan 4 distintos casos posibles de coincidencia de los nodos del tetraedro con los 3 nodos del triángulo:

- Nodos 1,2, y 3 del tetraedro coincidentes con los nodos del triángulo

En ese caso, la cara del elemento que pertenece a la superficie  $S_q$  es coincidente con el plano  $\tau = 0$ . Es decir, se integra sobre las variables  $\eta, \xi$ . En este caso, se dice que se integra sobre el plano 123. Se esquematiza esta situación en la Figura 18.

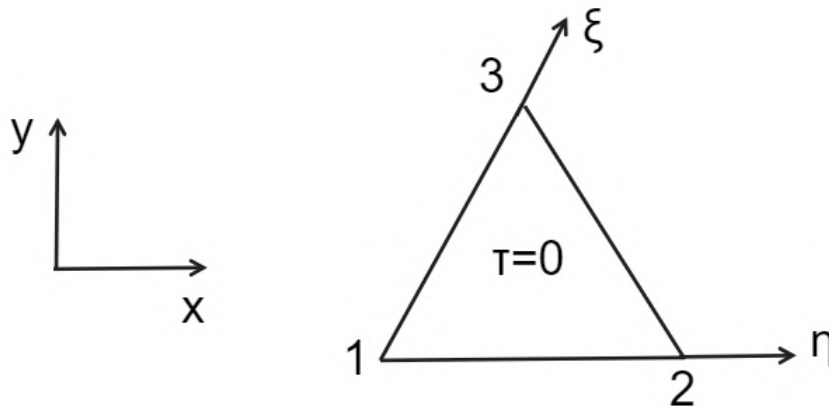


Figura 18: Esquema del caso del plano 123

En este caso, al ser  $\tau = 0$ , se elimina la fila 3 de la matriz jacobiana de la transformación I. Por lo tanto, el jacobiano resultante es:

$$|J_3|^{(e)} = \begin{vmatrix} \frac{\partial x^{(e)}}{\partial \eta} & \frac{\partial y^{(e)}}{\partial \eta} \\ \frac{\partial x^{(e)}}{\partial \xi} & \frac{\partial y^{(e)}}{\partial \xi} \end{vmatrix}$$

Luego, la transformación II es:

$$\begin{cases} \eta = l m \\ \xi = l(1 - m) \end{cases}$$

La transformación III es la que se explicó en la sección anterior (será la misma en todos estos casos).

- Nodos 1,2, y 4 del tetraedro coincidentes con los nodos del triángulo

En ese caso, la cara del elemento que pertenece a la superficie  $S_q$  es coincidente con el plano  $\xi = 0$ . Es decir, se integra sobre las variables  $\eta, \tau$ . En este caso, se dice que se integra sobre el plano 124. Se esquematiza esta situación en la Figura 19.

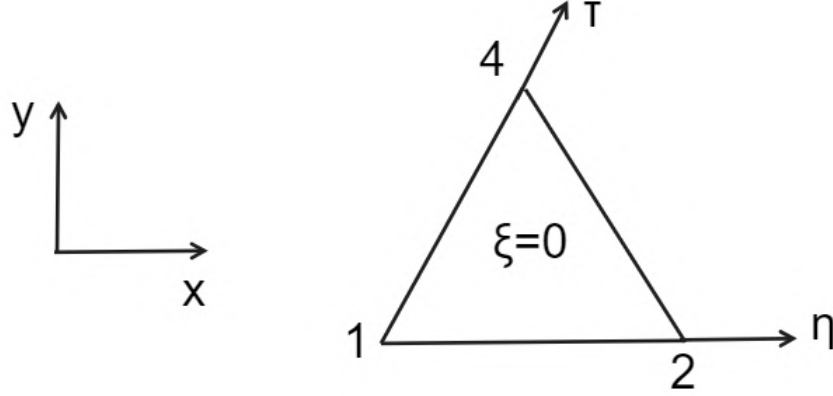


Figura 19: Esquema del caso del plano 124

En este caso, al ser  $\xi = 0$ , se elimina la fila 2 de la matriz jacobiana de la transformación I. Por lo tanto, el jacobiano resultante es:

$$|J_2|^{(e)} = \begin{vmatrix} \frac{\partial x^{(e)}}{\partial \eta} & \frac{\partial y^{(e)}}{\partial \eta} \\ \frac{\partial x^{(e)}}{\partial \tau} & \frac{\partial y^{(e)}}{\partial \tau} \end{vmatrix}$$

Luego, la transformación II es:

$$\begin{cases} \eta = l m \\ \tau = l(1 - m) \end{cases}$$

- Nodos 1,3, y 4 del tetraedro coincidentes con los nodos del triángulo

En este caso, la cara del elemento que pertenece a la superficie  $S_q$  es coincidente con el plano  $\eta = 0$ . Es decir, se integra sobre las variables  $\xi, \tau$ . Se dice que se integra sobre el plano 134. Se esquematiza esta situación en la Figura 20.

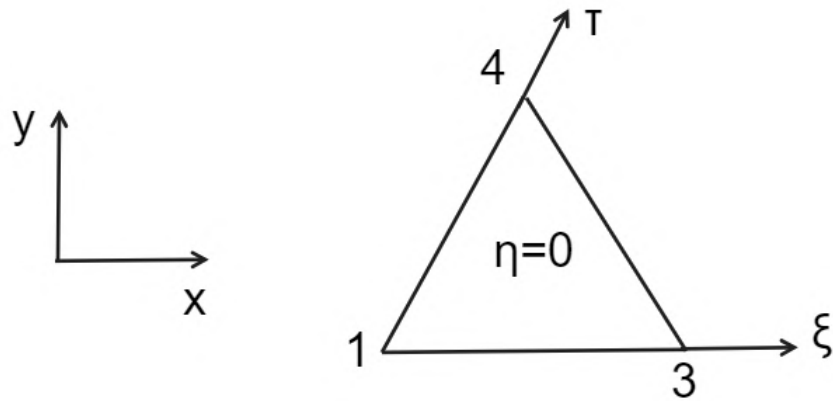


Figura 20: Esquema del caso del plano 134

En este caso, al ser  $\eta = 0$ , se elimina la fila 1 de la matriz jacobiana de la transformación I. Por lo tanto, el jacobiano resultante es:

$$|J_1|^{(e)} = \begin{vmatrix} \frac{\partial x^{(e)}}{\partial \xi} & \frac{\partial y^{(e)}}{\partial \xi} \\ \frac{\partial x^{(e)}}{\partial \tau} & \frac{\partial y^{(e)}}{\partial \tau} \end{vmatrix}$$

Luego, la transformación II es:

$$\begin{cases} \xi = l m \\ \tau = l(1 - m) \end{cases}$$

- Nodos 2,3 y 4 del tetraedro coincidentes con los nodos del triángulo

En este último caso, la cara del tetraedro que está sobre la superficie  $S_q$  es el plano de ecuación:

$$\eta + \xi + \tau = 1$$

Se podría parametrizar el plano y trabajar con dos variables, siendo la tercera dependiente de las otras, pero una solución más simple y práctica consiste en definir nuevas coordenadas  $\{\theta^1, \theta^2\}$ , tales como se muestran en la Figura 21.

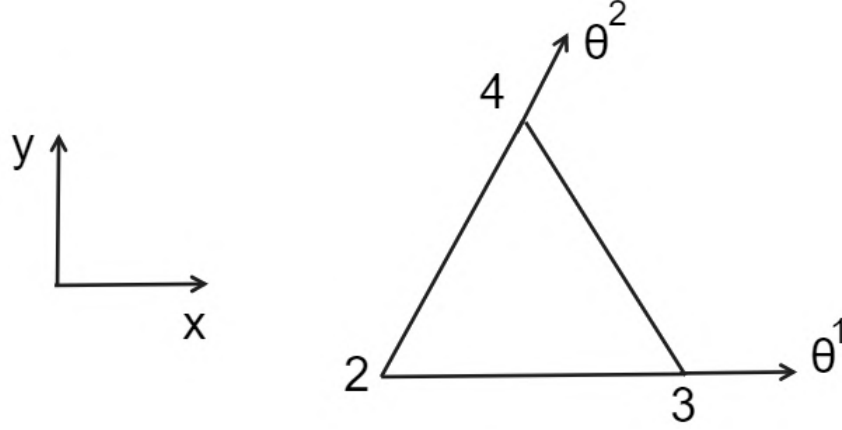


Figura 21: Esquema del caso del plano 234

En base a estas nuevas coordenadas, se definen nuevas funciones de forma:

$$\begin{cases} \psi_1 = 0 \\ \psi_2 = 1 - \theta^1 - \theta^2 \\ \psi_3 = \theta^1 \\ \psi_4 = \theta^2 \end{cases}$$

Las coordenadas globales interpoladas son entonces:

$$\begin{cases} x^{(e)} = \psi_2 x_2^{(e)} + \psi_3 x_3^{(e)} + \psi_4 x_4^{(e)} = (1 - \theta^1 - \theta^2) x_2^{(e)} + \theta^1 x_3^{(e)} + \theta^2 x_4^{(e)} \\ y^{(e)} = \psi_2 y_2^{(e)} + \psi_3 y_3^{(e)} + \psi_4 y_4^{(e)} = (1 - \theta^1 - \theta^2) y_2^{(e)} + \theta^1 y_3^{(e)} + \theta^2 y_4^{(e)} \end{cases}$$

Derivando, se halla la matriz jacobiana de la transformación I para este caso:

$$[J]^{(e)} = \begin{bmatrix} \frac{\partial x^{(e)}}{\partial \theta^1} & \frac{\partial x^{(e)}}{\partial \theta^2} \\ \frac{\partial y^{(e)}}{\partial \theta^1} & \frac{\partial y^{(e)}}{\partial \theta^2} \end{bmatrix} = \begin{bmatrix} x_3^{(e)} - x_2^{(e)} & y_3^{(e)} - y_2^{(e)} \\ x_4^{(e)} - x_2^{(e)} & y_4^{(e)} - y_2^{(e)} \end{bmatrix}$$

Luego, se plantea la transformación II:

$$\begin{cases} \theta^1 = l m \\ \theta^2 = l(1 - m) \end{cases}$$

El resto es idéntico que en los casos anteriores.

Entonces, en el código, se define en primer lugar una función que identifica los elementos tetraédricos pertenecientes a las superficies  $S_c$  y  $S_q$  y cuál es el plano que pertenece a la superficie.

```

1 def CB_neumann(nElementos2D, MC2D, nElementos3D, MC3D, X, physicalGroups):
2     CB = np.zeros(nElementos3D)
3     #Se recorren los elementos 2D
4     for i in range(nElementos2D):
5         #Si el elemento 2D esta sobre la superficie neumann:
6         if physicalGroups[MC2D[i][3]] == 'neumann':
7             #Se recorren los elementos 3D
8             for e in range(nElementos3D):
9                 #Coordenadas z de los nodos del elemento e
10                z1, z2, z3, z4 = X[int(MC3D[e][0]-1)][3], X[int(MC3D[e][1]-1)][3], X[int(MC3D[e][2]-1)
11                ][3], X[int(MC3D[e][3]-1)][3]
12                #Si los nodos 2,3,4 del tetraedro coinciden con los nodos del triangulo, y las
13                coordenadas z son las mismas para esos nodos:

```

```

12         if len(intersection(MC3D[e][[1,2,3]], MC2D[i][[5,6,7]]))==3 and z2==z3==z4:
13             #El plano 234 coincide con la superficie de neumann
14             CB[e] = 234 #( $\tau = 1-\eta-\xi$ )
15             #Si los nodos 1,3,4 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
16             if len(intersection(MC3D[e][[0,2,3]], MC2D[i][[5,6,7]]))==3 and z1==z3==z4:
17                 #El plano 134 coincide con la superficie de neumann
18                 CB[e] = 134 #( $\eta=0$ )
19                 #Si los nodos 1,2,4 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
20                 if len(intersection(MC3D[e][[0,1,3]], MC2D[i][[5,6,7]]))==3 and z1==z2==z4:
21                     #El plano 124 coincide con la superficie de neumann
22                     CB[e] = 124 #( $\xi=0$ )
23                     #Si los nodos 1,2,3 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
24                     if len(intersection(MC3D[e][[0,1,2]], MC2D[i][[5,6,7]]))==3 and z1==z2==z3:
25                         #El plano 123 coincide con la superficie de neumann
26                         CB[e] = 123 #( $r=0$ )
27         return CB
28
29 def CB_conveccion(nElementos2D,MC2D,nElementos3D,MC3D,X,physicalGroups):
30     CB = np.zeros(nElementos3D)
31     #Se recorren los elementos 2D
32     for i in range(nElementos2D):
33         #Si el elemento 2D esta sobre la superficie neumann:
34         if physicalGroups[MC2D[i][3]]=='conveccion':
35             #Se recorren los elementos 3D
36             for e in range(nElementos3D):
37                 #Coordenadas z de los nodos del elemento e
38                 z1,z2,z3,z4 = X[int(MC3D[e][0]-1)][3],X[int(MC3D[e][1]-1)][3],X[int(MC3D[e][2]-1)
][3],X[int(MC3D[e][3]-1)][3]
39                 #Si los nodos 2,3,4 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
40                 if len(intersection(MC3D[e][[1,2,3]], MC2D[i][[5,6,7]]))==3 and z2==z3==z4:
41                     #El plano 234 coincide con la superficie de neumann
42                     CB[e] = 234 #( $\tau = 1-\eta-\xi$ )
43                     #Si los nodos 1,3,4 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
44                     if len(intersection(MC3D[e][[0,2,3]], MC2D[i][[5,6,7]]))==3 and z1==z3==z4:
45                         #El plano 134 coincide con la superficie de neumann
46                         CB[e] = 134 #( $\eta=0$ )
47                         #Si los nodos 1,2,4 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
48                         if len(intersection(MC3D[e][[0,1,3]], MC2D[i][[5,6,7]]))==3 and z1==z2==z4:
49                             #El plano 124 coincide con la superficie de neumann
50                             CB[e] = 124 #( $\xi=0$ )
51                             #Si los nodos 1,2,3 del tetraedro coinciden con los nodos del triangulo, y las
coordenadas z son las mismas para esos nodos:
52                             if len(intersection(MC3D[e][[0,1,2]], MC2D[i][[5,6,7]]))==3 and z1==z2==z3:
53                                 #El plano 123 coincide con la superficie de neumann
54                                 CB[e] = 123 #( $r=0$ )
55         return CB

```

Luego, se define una función que calcula el vector de calor por la condición de Neumann de cada elemento introducido:

```

1 #Vector de calor local debido a la friccion entre pastilla y disco
2 #Toma como argumentos:
3 #X: matriz de coordenadas de los nodos de la malla inicial
4 #MC3De: fila (e) de la matriz de conectividades 3D (tetraedros) -> elemento (e)
5 #plano: plano sobre el cual se integra (123, 124, 134 o 234)
6 # $\omega$ : velocidad angular
7 # $\gamma$ : aceleracion angular
8 #k: conductividad termica del disco
9 # $\rho c_p$ : densidad por el calor especifico del disco
10 #CDA: coeficiente de drag por el area de referencia
11 #m: masa del vehiculo
12 #nGauss: numero de puntos de Gauss
13 def funcion_Qlocal(X,MC3De,plano, $\omega,\gamma,k,\rho c_p,CDA,m,nGauss$ ):
14     #Cantidad de nodos x elemento tetragonal
15     nodosXelemento3D = 4
16     if  $\omega == 0$ :
17         Qke = np.zeros((nodosXelemento3D,1))
18     else:
19         #Radio de la rueda
20         R = 0.3 #m
21         #Radio al baricentro del area de contacto
22         rg = 0.12064293 #m
23         #Area de contacto
24         Ac = 0.00587958 #m2
25         #Coordenadas globales de los nodos del elemento e

```

```

26 Xe = np.zeros((nodosXelemento3D,4))
27 for i in range(4):
28     Xe[i] = X[int(MC3De[i]-1)]
29 #Xe = [[nodo1(e), x1(e), y1(e), z1(e)],
30 #      [nodo2(e), x2(e), y2(e), z2(e)],
31 #      [nodo3(e), x3(e), y3(e), z3(e)],
32 #      [nodo4(e), x4(e), y4(e), z4(e)]]
33 #Se obtienen los pesos y puntos de Gauss
34 r,w = integralGauss(nGauss)
35 #Se inicializa el vector de calor local por conduccion
36 Qe = np.zeros((nodosXelemento3D,1))
37 #Densidad del aire
38  $\rho_a = 1.2$  #kg/m3
39 #Flujo de calor total
40 q = -(0.1*m*R**2* $\omega$ * $\gamma$ +0.5* $\rho_a$ *CDA*R**3* $\omega$ **3)/(8*Ac)
41 #Conductividad termica del la pastilla
42 kp = 2.5 #W/mK
43 #Densidad de la pastilla
44  $\rho_p = 1900$  #kg/m3
45 #Calor especifico de la pastilla
46 cpp = 950 #J/kgK
47 #Relacion disco/pastilla
48  $\kappa = \text{np.sqrt}(k*\rho_c p)/(\text{np.sqrt}(k*\rho_c p)+\text{np.sqrt}(kp*\rho_p*cpp))$ 
49 #Flujo de calor absorbido por el disco
50 qd =  $\kappa*q$ 
51 #Se recorren los puntos de Gauss
52 for a in range(nGauss):
53     for b in range(nGauss):
54         #Si la superficie de neumann del tetraedro es el plano 234:
55         if plano==234:
56             #Transformacion III
57             l = (1+r[a])/2 # u = r[a]
58             m = (1+r[b])/2 # v = r[b]
59             #Jacobiano
60             Juv = 1/4
61             #Transformacion de triangulo a cuadrado
62              $\theta^1 = l*m$ 
63              $\theta^2 = l*(1-m)$ 
64             #Jacobiano
65             Jlm = abs(l)
66             #Se obtienen el vector de las funciones de forma
67              $\Phi = \text{np.array}([0,1-\theta^1-\theta^2,\theta^1,\theta^2])$ 
68             #Se calcula la traspuesta
69              $\Phi^T = \text{np.reshape}(\Phi,(nodosXelemento3D,1))$ 
70             #Coordenadas de los nodos {2,3,4}
71             x2,y2 = Xe[1][1], Xe[1][2]
72             x3,y3 = Xe[2][1], Xe[2][2]
73             x4,y4 = Xe[3][1], Xe[3][2]
74             #Se obtiene la matriz jacobiana
75             matrizJ = np.array([[x3-x2,y3-y2],
76                                 [x4-x2,y4-y2]])
77             #Se calcula el jacobiano
78             J = abs(np.linalg.det(matrizJ))
79         #Si la superficie de neumann del tetraedro es el plano 134:
80         elif plano==134:
81             #Transformacion III
82             l = (1+r[a])/2 # u = r[a]
83             m = (1+r[b])/2 # v = r[b]
84             #Jacobiano
85             Juv = 1/4
86             #Transformacion de triangulo a cuadrado
87              $\xi = l*m$ 
88              $\tau = l*(1-m)$ 
89             #Jacobiano
90             Jlm = abs(l)
91             #Se obtienen el vector de las funciones de forma
92              $\Phi = \text{funcion\_}\Phi(0,\xi,\tau)$ 
93             #Se calcula la traspuesta
94              $\Phi^T = \text{np.reshape}(\Phi,(nodosXelemento3D,1))$ 
95             #Se obtiene la matriz de las derivadas de las funciones de forma
96              $\partial\Phi = \text{funcion\_}\partial\Phi(0,\xi,\tau)$ 
97             #Se obtiene la matriz jacobiana
98             matrizJ = funcion_J(X, MC3De,  $\partial\Phi$ )
99             #Se elimina la 1era fila (plano 134:  $\eta=cte$ )
100            matrizJ = np.delete(matrizJ,0,axis=0)
101            #Se elimina la 3era columna (z=cte)
102            matrizJ = np.delete(matrizJ,2,axis=1)
103            #Se calcula el jacobiano

```

```

104     J = abs(np.linalg.det(matrizJ))
105     #Si la superficie de neumann del tetraedro es el plano 124:
106     elif plano==124:
107         #Transformacion III
108         l = (1+r[a])/2 # u = r[a]
109         m = (1+r[b])/2 # v = r[b]
110         #Jacobiano
111         Juv = 1/4
112         #Transformacion de triangulo a cuadrado
113         η = l*m
114         τ = l*(1-m)
115         #Jacobiano
116         Jlm = abs(l)
117         #Se obtienen el vector de las funciones de forma
118         Φ = funcion_Φ(η,0,τ)
119         #Se calcula la traspuesta
120         ΦT = np.reshape(Φ,(nodosXelemento3D,1))
121         #Se obtiene la matriz de las derivadas de las funciones de forma
122         ∂Φ = funcion_∂Φ(η,0,τ)
123         #Se obtiene la matriz jacobiana
124         matrizJ = funcion_J(X, MC3De, ∂Φ)
125         #Se elimina la 2da fila (plano 124: ξ=cte)
126         matrizJ = np.delete(matrizJ,1,axis=0)
127         #Se elimina la 3era columna (z=cte)
128         matrizJ = np.delete(matrizJ,2,axis=1)
129         #Se calcula el jacobiano
130         J = abs(np.linalg.det(matrizJ))
131     #Si la superficie de neumann del tetraedro es el plano 123:
132     elif plano==123:
133         #Transformacion III
134         l = (1+r[a])/2 # u = r[a]
135         m = (1+r[b])/2 # v = r[b]
136         #Jacobiano
137         Juv = 1/4
138         #Transformacion de triangulo a cuadrado
139         η = l*m
140         ξ = l*(1-m)
141         #Jacobiano
142         Jlm = abs(l)
143         #Se obtienen el vector de las funciones de forma
144         Φ = funcion_Φ(η,ξ,0)
145         #Se calcula la traspuesta
146         ΦT = np.reshape(Φ,(nodosXelemento3D,1))
147         #Se obtiene la matriz de las derivadas de las funciones de forma
148         ∂Φ = funcion_∂Φ(η,ξ,0)
149         #Se obtiene la matriz jacobiana
150         matrizJ = funcion_J(X, MC3De, ∂Φ)
151         #Se elimina la 3era fila (plano 123: τ=cte)
152         matrizJ = np.delete(matrizJ,2,axis=0)
153         #Se elimina la 3era columna (z=cte)
154         matrizJ = np.delete(matrizJ,2,axis=1)
155         #Se calcula el jacobiano
156         J = abs(np.linalg.det(matrizJ))
157     #Se calcula el vector de calor
158     Qe += w[a]*w[b]*qd*J*Jlm*Juv*ΦT
159     return Qe

```

De manera similar, se define una función que calcula la matriz de rigidez y el vector de calor de convección para los elementos pertenecientes a la superficie de convección  $S_c$ :

```

1  #Matriz de rigidez local y vector de calor local por conveccion
2  #Toma como argumentos:
3  #X: matriz de coordenadas de los nodos
4  #MC3De: fila (e) de la matriz de conectividades 3D (tetrahedros) -> elemento (e)
5  #hc: coeficiente de conveccion
6  #T_aire: temperatura del aire ambiente
7  #plano: el plano del elemento tetragonal coincidente con la superficie de integracion
8  #nGauss: numero de puntos de Gauss
9  #Devuelve la matriz de rigidez local debida a la conveccion y el vector de calor local debido a la
   conveccion
10 def conveccion(X,MC3De,hc,T_aire,plano,nGauss):
11     #Cantidad de nodos x elemento tetragonal
12     nodosXelemento3D = 4
13     #Coordenadas globales de los nodos del elemento e
14     Xe = np.zeros((nodosXelemento3D,4))
15     for i in range(4):
16         Xe[i] = X[int(MC3De[i]-1)]
17     #Xe = [[nodo1(e), x1(e), y1(e), z1(e)],
18     #      [nodo2(e), x2(e), y2(e), z2(e)],

```



```

19 # [nodo3(e), x3(e), y3(e), z3(e)],
20 # [nodo4(e), x4(e), y4(e), z4(e)]]
21 #Se obtienen los pesos y puntos de Gauss
22 r,w = integralGauss(nGauss)
23 #Se inicializa la matriz de rigidez local
24 Kce = np.zeros((nodosXelemento3D,nodosXelemento3D))
25 #Se inicializa el vector de calor local por conveccion
26 Qce = np.zeros((nodosXelemento3D,1))
27 #Se recorren los puntos de Gauss
28 for a in range(nGauss):
29     for b in range(nGauss):
30         #Si la superficie de conveccion del tetraedro es el plano 234:
31         if plano==234:
32             #Transformacion de cuadrado a cuadrado centrado
33             l = (1+r[a])/2 # u = r[a]
34             m = (1+r[b])/2 # v = r[b]
35             #Jacobiano
36             Juv = 1/4
37             #Transformacion de triangulo cuadrado
38              $\theta^1 = l*m$ 
39              $\theta^2 = l*(1-m)$ 
40             #Jacobiano
41             Jlm = abs(1)
42             #Se obtienen el vector de las funciones de forma
43              $\Phi = \text{np.array}([[0, 1-\theta^1-\theta^2, \theta^1, \theta^2]])$ 
44             #Se calcula la traspuesta
45              $\Phi^T = \text{np.reshape}(\Phi, (\text{nodosXelemento3D}, 1))$ 
46             #Coordenadas de los nodos {2,3,4}
47             x2,y2 = Xe[1][1], Xe[1][2]
48             x3,y3 = Xe[2][1], Xe[2][2]
49             x4,y4 = Xe[3][1], Xe[3][2]
50             #Se obtiene la matriz jacobiana
51             matrizJ = np.array([[x3-x2,y3-y2],
52                                 [x4-x2,y4-y2]])
53             #Se calcula el determinante
54             J = abs(np.linalg.det(matrizJ))
55
56         #Si la superficie de conveccion del tetraedro es el plano 134:
57         elif plano==134:
58             #Transformacion de cuadrado a cuadrado centrado
59             l = (1+r[a])/2 # u = r[a]
60             m = (1+r[b])/2 # v = r[b]
61             #Jacobiano
62             Juv = 1/4
63             #Transformacion de triangulo a cuadrado
64              $\xi = l*m$ 
65              $\tau = l*(1-m)$ 
66             #Jacobiano
67             Jlm = abs(1)
68             #Se obtienen el vector de las funciones de forma
69              $\Phi = \text{np.array}([\text{funcion\_}\Phi(0,\xi,\tau)])$ 
70             #Se calcula la traspuesta
71              $\Phi^T = \text{np.reshape}(\Phi, (\text{nodosXelemento3D}, 1))$ 
72             #Se obtiene la matriz de las funciones de forma
73              $\partial\Phi = \text{funcion\_}\partial\Phi(0,\xi,\tau)$ 
74             #Se obtiene la matriz jacobiana
75             matrizJ = funcion_J(X, MC3De,  $\partial\Phi$ )
76             #Al estar en el plano 134, se borra la fila 1
77             matrizJ = np.delete(matrizJ,0,axis=0)
78             #Al estar en el plano xy (z=cte), se borra la columna 3
79             matrizJ = np.delete(matrizJ,2,axis=1)
80             #Se calcula el jacobiano
81             J = abs(np.linalg.det(matrizJ))
82
83         #Si la superficie de conveccion del tetraedro es el plano 124:
84         elif plano==124:
85             #Transformacion de cuadrado a cuadrado centrado
86             l = (1+r[a])/2 # u = r[a]
87             m = (1+r[b])/2 # v = r[b]
88             #Jacobiano
89             Juv = 1/4
90             #Transformacion de triangulo a cuadrado
91              $\eta = l*m$ 
92              $\tau = l*(1-m)$ 
93             #Jacobiano
94             Jlm = abs(1)
95             #Se obtienen el vector de las funciones de forma
96              $\Phi = \text{np.array}([\text{funcion\_}\Phi(\eta,0,\tau)])$ 

```

```

97     #Se calcula la traspuesta
98      $\Phi^T$  = np.reshape( $\Phi$ , (nodosXelemento3D, 1))
99     #Se obtiene la matriz de las funciones de forma
100      $\partial\Phi$  = funcion_ $\partial\Phi$ ( $\eta$ , 0,  $\tau$ )
101     #Se obtiene la matriz jacobiana
102     matrizJ = funcion_J(X, MC3De,  $\partial\Phi$ )
103     #Al estar en el plano 124, se borra fila 2
104     matrizJ = np.delete(matrizJ, 1, axis=0)
105     #Al estar en el plano xy (z=cte), se borra la columna 3
106     matrizJ = np.delete(matrizJ, 2, axis=1)
107     #Se calcula el jacobiano
108     J = abs(np.linalg.det(matrizJ))
109
110     #Si la superficie de conveccion del tetraedro es el plano 123:
111     elif plano==123:
112         #Transformacion de cuadrado cuadrado centrado
113         l = (1+r[a])/2 # u = r[a]
114         m = (1+r[b])/2 # v = r[b]
115         #Jacobiano
116         Juv = 1/4
117         #Transformacion de triangulo a cuadrado
118          $\eta$  = l*m
119          $\xi$  = l*(1-m)
120         #Jacobiano
121         Jlm = abs(1)
122         #Se obtienen el vector de las funciones de forma
123          $\Phi$  = np.array([funcion_ $\Phi$ ( $\eta$ ,  $\xi$ , 0)])
124         #Se calcula la traspuesta
125          $\Phi^T$  = np.reshape( $\Phi$ , (nodosXelemento3D, 1))
126         #Se obtiene la matriz de las funciones de forma
127          $\partial\Phi$  = funcion_ $\partial\Phi$ ( $\eta$ ,  $\xi$ , 0)
128         #Se obtiene la matriz jacobiana
129         matrizJ = funcion_J(X, MC3De,  $\partial\Phi$ )
130         #Al estar en el plano 123, se borra la fila 3
131         matrizJ = np.delete(matrizJ, 2, axis=0)
132         #Al estar en el plano xy (z=cte), se borra la columna 3
133         matrizJ = np.delete(matrizJ, 2, axis=1)
134         #Se calcula el jacobiano
135         J = abs(np.linalg.det(matrizJ))
136
137     #Se calcula la matriz de rigidez local por conveccion
138     Kce += w[a]*w[b]*hc*J*Jlm*Juv*np.matmul( $\Phi^T$ ,  $\Phi$ )
139     #Se calcula el vector de calor local por conveccion
140     Qce += w[a]*w[b]*hc*T_aire*J*Jlm*Juv* $\Phi^T$ 
141
142     return Kce, Qce

```

### 3.2.8. Ensamble

Habiendo calculado todas las matrices y vectores elementales, se busca ahora ensamblarlos para obtener matrices y vectores globales, es decir, que representen toda la malla. Esto se consigue planteando la equivalencia entre los nodos globales de la malla con los locales de cada elemento, a través de la matriz de conectividades de los elementos tetraédricos ( $[MC3D]$ ). Por ejemplo, para la matriz de rigidez de un elemento (e):

$$\underline{\underline{K}}^{(e)} = \begin{array}{c|cccc|c} & \begin{array}{c} 1 \text{ local} \\ MC3D_{e1} \text{ global} \end{array} & \begin{array}{c} 2 \text{ local} \\ MC3D_{e2} \text{ global} \end{array} & \begin{array}{c} 3 \text{ local} \\ MC3D_{e3} \text{ global} \end{array} & \begin{array}{c} 4 \text{ local} \\ MC3D_{e4} \text{ global} \end{array} & \\ \hline & K_{11}^{(e)} & K_{12}^{(e)} & K_{13}^{(e)} & K_{14}^{(e)} & \begin{array}{c} 1 \text{ local} \\ MC3D_{e1} \text{ global} \end{array} \\ \hline & K_{21}^{(e)} & K_{22}^{(e)} & K_{23}^{(e)} & K_{24}^{(e)} & \begin{array}{c} 2 \text{ local} \\ MC3D_{e2} \text{ global} \end{array} \\ \hline & K_{31}^{(e)} & K_{32}^{(e)} & K_{33}^{(e)} & K_{34}^{(e)} & \begin{array}{c} 3 \text{ local} \\ MC3D_{e3} \text{ global} \end{array} \\ \hline & K_{41}^{(e)} & K_{42}^{(e)} & K_{43}^{(e)} & K_{44}^{(e)} & \begin{array}{c} 4 \text{ local} \\ MC3D_{e4} \text{ global} \end{array} \\ \hline \end{array}$$

Se puede deducir entonces que la componente  $ij$  de cada matriz de rigidez local se corresponde con la componente  $MC3D_{ei}MC3D_{ej}$  de la matriz de rigidez global. Es decir,

$$K_{MC3D_{ei}MC3D_{ej}} = K_{ij}^{(e)}$$

Además, a medida que se recorren los elementos, es importante sumar las componentes de cada matriz elemental, sin sobrescribir los resultados. Por lo tanto, sumando el resultado anterior, queda:

$$K_{MC3D_{ei}MC_{ej}} = K_{ij}^{(e)} + K_{MC3D_{ei}MC_{ej}}|_{\text{anterior}}$$

Se puede hacer el mismo análisis para todas las matrices y vectores. De esta forma, se programa una función ensamble que devuelve las matrices y vectores globales:

```

1 #Ensamble
2 #Toma como argumentos:
3 #X_orig: la matriz de coordenadas iniciales de los nodos
4 #X: la matriz de coordenadas de los nodos rotadas
5 #T: vector de temperaturas propuesto
6 #elem_neumann: lista de elementos 3D pertenecientes a la superficie Sq
7 #MC3D: matriz de conectividades de los elementos tetragonales de 4 nodos
8 #nElementos3D: cantidad de elementos tetragonales de 4 nodos
9 #CDA: coeficiente de Drag por Area del vehiculo [m^2]
10 #ω: velocidad angular
11 #γ: aceleracion angular
12 #m: masa del vehiculo
13 #nGauss: numero de puntos de Gauss para la integracion numerica
14 def ensamble(X_orig,X,T,elem_neumann,MC3D,nElementos3D,CDA,ω,γ,m,nGauss):
15     #Se inicializan las matrices y vectores globales
16     M = np.zeros((nNodos,nNodos))
17     K = np.zeros((nNodos,nNodos))
18     N = np.zeros((nNodos,nNodos))
19     Q = np.zeros((nNodos,1))
20     #Nodos x elemento 3D
21     nodosXelemento3D = 4
22     #Se inicializa un vector de temperaturas de los elementos3D(tetragonales)
23     Telem = np.zeros(nElementos3D)
24     #Se recorren los elementos 3D
25     for e in range(nElementos3D):
26         #Se recorren los nodos del elemento (e)
27         for j in range(nodosXelemento3D):
28             #Se calcula el promedio de las temperaturas nodales y se la almacena en la posicion (e)
29             #del vector de temperaturas elementales
30             Telem[e] += 1/nodosXelemento3D*T[int(MC3D[e][j]-1)][0]
31
32     #Se obtiene la conductividad termica del elemento del disco
33     k = funcion_k_fundicion(Telem[e])
34     #Se obtiene el producto densidad calor especifico del elemento del disco
35     ρcp = funcion_ρcp_fundicion(Telem[e])
36     #Difusividad termica en el disco
37     α = k/(ρcp)
38     #Condicion de borde de conveccion
39     if elem_conveccion[e]!=0:
40         #Plano coincidente con la superficie Sc
41         plano = elem_conveccion[e]
42         if plano==123:
43             #Coordenadas de los nodos del elemento
44             x1,y1 = X[int(MC3D[e][0]-1)][1],X[int(MC3D[e][0]-1)][2] #1
45             x2,y2 = X[int(MC3D[e][1]-1)][1],X[int(MC3D[e][1]-1)][2] #2
46             x3,y3 = X[int(MC3D[e][2]-1)][1],X[int(MC3D[e][2]-1)][2] #3
47         elif plano==124:
48             #Coordenadas de los nodos del elemento
49             x1,y1 = X[int(MC3D[e][0]-1)][1],X[int(MC3D[e][0]-1)][2] #1
50             x2,y2 = X[int(MC3D[e][1]-1)][1],X[int(MC3D[e][1]-1)][2] #2
51             x3,y3 = X[int(MC3D[e][3]-1)][1],X[int(MC3D[e][3]-1)][2] #4
52         elif plano==134:
53             #Coordenadas de los nodos del elemento
54             x1,y1 = X[int(MC3D[e][0]-1)][1],X[int(MC3D[e][0]-1)][2] #1
55             x2,y2 = X[int(MC3D[e][2]-1)][1],X[int(MC3D[e][2]-1)][2] #3
56             x3,y3 = X[int(MC3D[e][3]-1)][1],X[int(MC3D[e][3]-1)][2] #4
57         elif plano==234:
58             #Coordenadas de los nodos del elemento
59             x1,y1 = X[int(MC3D[e][1]-1)][1],X[int(MC3D[e][1]-1)][2] #2
60             x2,y2 = X[int(MC3D[e][2]-1)][1],X[int(MC3D[e][2]-1)][2] #3
61             x3,y3 = X[int(MC3D[e][3]-1)][1],X[int(MC3D[e][3]-1)][2] #4
62     #Lados del triangulo
63     a1 = np.sqrt((x2-x1)**2 + (y2-y1)**2)
64     a2 = np.sqrt((x3-x2)**2 + (y3-y2)**2)
65     a3 = np.sqrt((x3-x1)**2 + (y3-y1)**2)
66     #Semi-perimetro
67     s = (a1 + a2 + a3)/2
68     #Area del triangulo
69     Area = np.sqrt(s*(s-a1)*(s-a2)*(s-a3))
70     #Longitud caracteristica
71     Le = np.sqrt(Area)

```

```

71 #Se obtiene el coeficiente de conveccion en la superficie del elemento
72 hc = funcion_hc(V,Telem[e],T_aire,Le,plano)
73 #Se obtiene la matriz de rigidez de conveccion local
74 #y el vector de calor por conveccion local
75 Kce, Qce = conveccion(X,MC3D[e],hc,T_aire,plano,nGauss)
76 elif elem_conveccion[e]==0:
77     Kce = np.zeros((nodosXelemento3D,nodosXelemento3D))
78     Qce = np.zeros((nodosXelemento3D,1))
79
80 #Condicion de borde de neumann
81 if elem_neumann[e]!=0:
82     plano = elem_neumann[e]
83     Qqe = funcion_Qlocal(X_orig,MC3D[e],plano,omega,gamma,k,rho_cp,CDA,m,nGauss)
84 elif elem_neumann[e]==0:
85     Qqe = np.zeros((nodosXelemento3D,1))
86
87 #Se obtiene la matriz de rigidez por conduccion local
88 Ke = funcion_Klocal(X,MC3D[e],k,nGauss)
89 #Se obtiene la matriz convectiva local
90 Ne = funcion_Nlocal(X,MC3D[e],alpha,rho_cp,omega,nGauss)
91 #Se obtiene la matriz de masa local
92 Me = funcion_Mlocal(X,MC3D[e],alpha,rho_cp,nGauss)
93 for i in range(nodosXelemento3D):
94     #Vector de calor por conduccion global
95     Qq[MC3D[e][i]-1][0] = Qq[MC3D[e][i]-1][0] + Qqe[i][0]
96     #Vector de calor por conveccion global
97     Qc[MC3D[e][i]-1][0] = Qc[MC3D[e][i]-1][0] + Qce[i][0]
98     for j in range(nodosXelemento3D):
99         #Matriz de masa global
100         M[MC3D[e][i]-1][MC3D[e][j]-1] = M[MC3D[e][i]-1][MC3D[e][j]-1] + Me[i][j]
101         #Matriz convectiva global
102         N[MC3D[e][i]-1][MC3D[e][j]-1] = N[MC3D[e][i]-1][MC3D[e][j]-1] + Ne[i][j]
103         #Matriz de rigidez por conduccion global
104         Kk[MC3D[e][i]-1][MC3D[e][j]-1] = Kk[MC3D[e][i]-1][MC3D[e][j]-1] + Ke[i][j]
105         #Matriz de rigidez por conveccion global
106         Kc[MC3D[e][i]-1][MC3D[e][j]-1] = Kc[MC3D[e][i]-1][MC3D[e][j]-1] + Kce[i][j]
107 #Matriz de rigidez total (conduccion + conveccion)
108 K = Kk + Kc
109 #Vector de calor total (neumann + conveccion)
110 Q = Qq + Qc
111 return M,N,K,Q

```

### 3.2.9. Resolución del problema transitorio no lineal

Una vez obtenidas las matrices y el vector de calor globales, se debe resolver el sistema siguiente:

$$\underline{\underline{M}}(T) \cdot \dot{\underline{T}} + \left( \underline{\underline{N}}(T) + \underline{\underline{K}}(T) + \underline{\underline{K}}_c(T) \right) \cdot \underline{T} = \underline{\underline{Q}}_q(T) + \underline{\underline{Q}}_c(T) \quad (7)$$

La matriz de masa y la convectiva dependen de la temperatura debido a que la densidad y el calor específico son funciones de la temperatura:

```

1 def funcion_rho_cp_fundicion(T):
2     return 2.06765696e+03*T + 2.67410206e+06

```

La matriz de rigidez depende de la temperatura debido a que la conductividad térmica es función de ella:

```

1 def funcion_k_fundicion(T):
2     return -8.22099060e-05*(T - 4.90413792e+02)**2 + 4.07923450e+01

```

El vector de calor de Neumann también depende de la temperatura ya que utiliza  $k$ ,  $\rho$  y  $c_p$ . Además, la matriz de rigidez y el vector de calor de convección dependen también de la temperatura debido a la dependencia del coeficiente de convección:  $h_c = h_c(T)$

Se define la matriz auxiliar siguiente:

$$\underline{\underline{E}}(T) = \underline{\underline{N}}(T) + \underline{\underline{K}}(T) + \underline{\underline{K}}_c(T)$$

Y el vector de calor total:

$$\underline{\underline{Q}}(T) = \underline{\underline{Q}}_q(T) + \underline{\underline{Q}}_c(T)$$

De esta forma, queda el sistema siguiente:

$$\underline{\underline{M}}(T) \cdot \dot{\underline{T}} + \underline{\underline{E}}(T) \cdot \underline{T} = \underline{\underline{Q}}(T)$$

Para resolver el problema transitorio, se plantea el método de Crank-Nicholson (Diferencias finitas centradas), llegando a la siguiente expresión, con  $\beta = 0,5$ :

$$\underline{\underline{A}} = \frac{1}{\Delta t} \left[ (1 - \beta) {}^t \underline{\underline{M}} + \beta {}^{t+\Delta t} \underline{\underline{M}} \right] + \beta \left[ (1 - \beta) {}^t \underline{\underline{E}} + \beta {}^{t+\Delta t} \underline{\underline{E}} \right]$$

$$\underline{b} = (1 - \beta) {}^t\underline{Q} + \beta {}^{t+\Delta t}\underline{Q} + \left\{ \frac{1}{\Delta t} [(1 - \beta) {}^t\underline{M} + \beta {}^{t+\Delta t}\underline{M}] - (1 - \beta) [(1 - \beta) {}^t\underline{E} + \beta {}^{t+\Delta t}\underline{E}] \right\} \cdot {}^t\underline{T}$$

$${}^{t+\Delta t}\underline{T} = \underline{\underline{A}}^{-1} \cdot \underline{b}$$

Pero como las matrices dependen de la temperatura, no se puede simplemente realizar la operación anterior, porque  ${}^{t+\Delta t}\underline{M}$ ,  ${}^{t+\Delta t}\underline{E}$ ,  ${}^{t+\Delta t}\underline{Q}$  no se conocen.

Este es el inconveniente que surge de considerar la no linealidad del material del disco en el análisis. Pero existen métodos para resolverlo. En este trabajo, se utilizó el método de Picard, que consiste en:

1. Conociendo las temperaturas en tiempo  $t$ , calcular las matrices y el vector de calor en  $t$
2. Proponer las temperaturas nodales en tiempo  $t + \Delta t$ , y con éstas, calcular las matrices y el vector  $\underline{Q}$  en  $t + \Delta t$
3. Con todas las matrices y vectores calculados, obtener la matriz auxiliar  $\underline{\underline{A}}$  y el vector auxiliar  $\underline{b}$
4. Calcular un nuevo vector de temperaturas nodales:  ${}^{t+\Delta t}\underline{T} = \underline{\underline{A}}^{-1} \cdot \underline{b}$
5. Comparar el nuevo vector de temperaturas nodales en  $t + \Delta t$  con el que se había propuesto mediante un error relativo.
  - Si el error relativo calculado es mayor a una determinada tolerancia, se propone el último vector de temperaturas nodales calculado en una nueva iteración **sin avanzar en el tiempo** y se vuelven a repetir los pasos anteriores
  - Si el error relativo es menor a la tolerancia, se guarda el vector de temperaturas nodales calculado, **se avanza un paso temporal** y se vuelven a repetir los pasos anteriores, donde el vector de temperaturas en tiempo  $t$  pasa a ser el último que se calculó:  ${}^{t+\Delta t}\underline{T}$

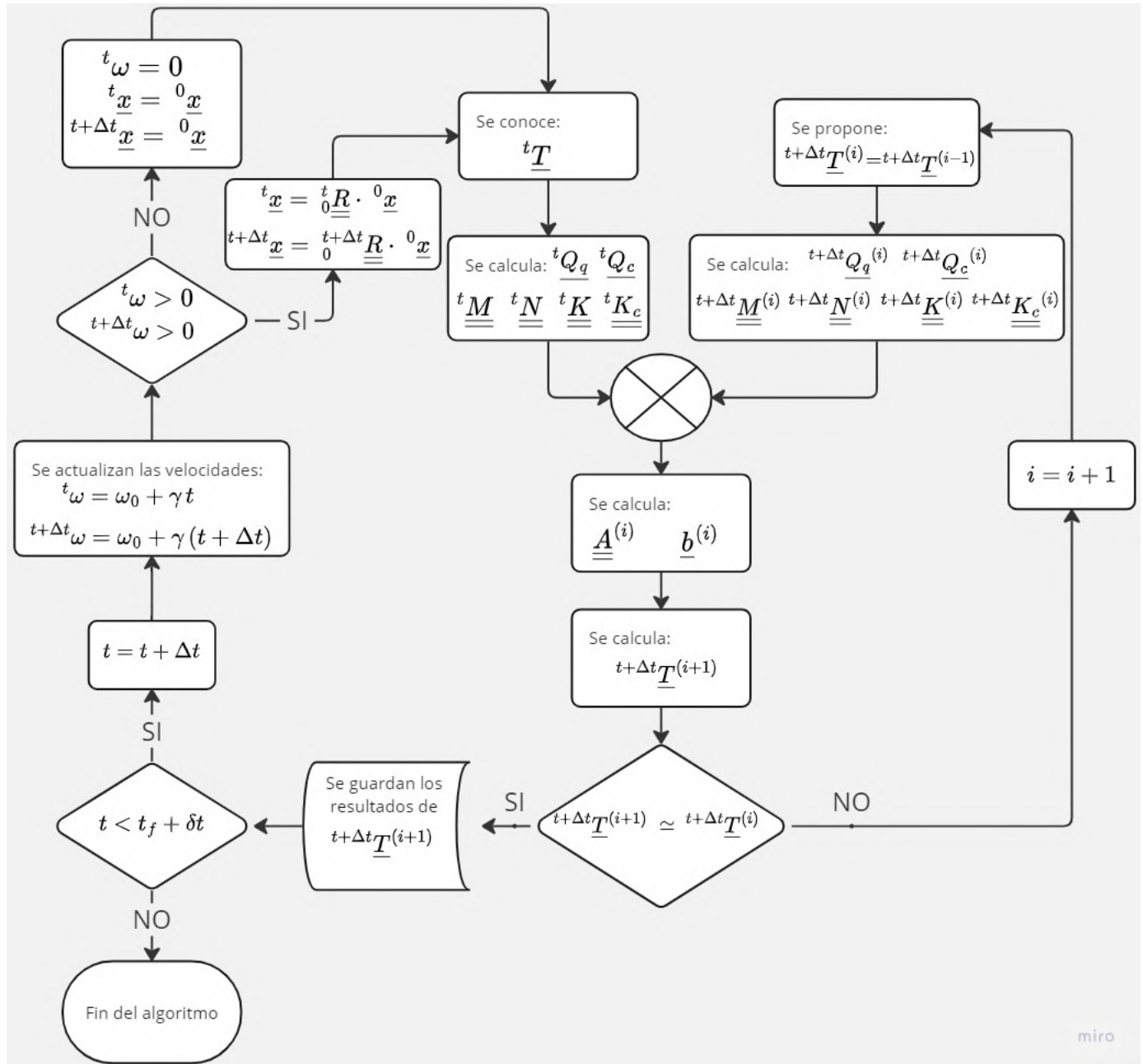


Figura 22: Diagrama de flujo del algoritmo por el método de Picard

En cada tiempo se deben actualizar los valores de la velocidad angular y la de traslación:

$$\begin{aligned}
 t\omega &= \omega_0 + \gamma t = \omega_0 \left(1 - \frac{t}{t_f}\right) \\
 t+\Delta t\omega &= \omega_0 + \gamma(t + \Delta t) = \omega_0 \left(1 - \frac{t + \Delta t}{t_f}\right) \\
 tV &= V_0 \left(1 - \frac{t}{t_f}\right) \\
 t+\Delta tV &= V_0 \left(1 - \frac{t + \Delta t}{t_f}\right)
 \end{aligned}$$

A medida que se incrementa el tiempo, la velocidad disminuye. Eventualmente, si la velocidad se vuelve negativa, se la fuerza a que tenga un valor nulo. Es decir, si el tiempo es superior al tiempo final (para el cual se anula la velocidad), se fija  $\omega = 0$ . Esta ventana de tiempo adicional se denominará  $\delta t$ .

$$t\omega = tV = 0 \quad \text{si} \quad t_f < t < t_f + \delta t$$

De esta manera, se puede simular una ventana de tiempo, con el vehículo detenido, donde se lograrán apreciar los efectos difusivos, sin la interferencia de los efectos convectivos. En esta ventana temporal, al tener ausencia de

velocidad, el sistema es más estable, por lo tanto, se puede utilizar un paso temporal mayor para reducir los tiempos computacionales. Se define entonces un paso temporal variable, que aumenta gradualmente desde un valor mínimo (utilizado en la ventana con el vehículo frenando) hasta un valor máximo:

$$\Delta t(t) = \Delta t_{\max} \left( 1 - e^{-\frac{t_f - t}{t_f}} \right) + \Delta t_{\min}$$

Además, se deben rotar las coordenadas de los nodos en cada tiempo:

$$\begin{aligned} {}^t \underline{x} &= {}^0 \underline{R} \cdot {}^0 \underline{x} \\ {}^{t+\Delta t} \underline{x} &= {}^{t+\Delta t} \underline{R} \cdot {}^0 \underline{x} \end{aligned}$$

Donde  ${}^t \underline{R}$  es el tensor de rotación rígida al rededor del eje  $z$ :

$${}^t \underline{R} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Siendo  $\theta$  el ángulo de rotación:

$$\theta(t) = \int_0^t \omega(t) dt = \omega_0 t + \frac{\gamma}{2} t^2$$

Se guardan los vectores de temperaturas nodales en cada tiempo en una matriz como la siguiente:

$$[\text{Resultados}] = \begin{bmatrix} {}^{t_0} T_1 & {}^{t_0} T_2 & {}^{t_0} T_3 & \dots & {}^{t_0} T_N \\ {}^{t_1} T_1 & {}^{t_1} T_2 & {}^{t_1} T_3 & \dots & {}^{t_1} T_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ {}^{t_i} T_1 & {}^{t_i} T_2 & {}^{t_i} T_3 & \dots & {}^{t_i} T_N \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ {}^{t_f} T_1 & {}^{t_f} T_2 & {}^{t_f} T_3 & \dots & {}^{t_f} T_N \end{bmatrix}$$

Es decir, la fila  $i$  es el vector de temperaturas nodales  ${}^{t_i} \underline{T}$ , siendo  $t_i = i\Delta t$ .

Para verificar la estabilidad del sistema, en cada tiempo, se debe verificar:

$$\rho(\underline{D}) \leq 1$$

Siendo  $\rho(\underline{D})$  el radio espectral (el máximo autovalor) de la matriz:

$$\underline{D} = \underline{A}^{-1} \cdot \underline{C}$$

Con:

$$\underline{C} = \frac{1}{\Delta t} [(1 - \beta) {}^t \underline{M} + \beta {}^{t+\Delta t} \underline{M}] - (1 - \beta) [(1 - \beta) ({}^t \underline{N} + {}^t \underline{K}) + \beta ({}^{t+\Delta t} \underline{N} + {}^{t+\Delta t} \underline{K})]$$

En el código, se escribe:

```
1 #Se inicializa una matriz de resultados
2 Resultados = np.zeros((int((1+(tf+dt-t0)/dt)), nNodos))
3 #La matriz de resultados sera de la forma:
4 #Resultados = [[T1(t0), T2(t0), T3(t0), ... , Tn(t0)],
5 #               [T1(t1), T2(t1), T3(t1), ... , Tn(t1)],
6 #               [ ... , ... , ... , ... , ... ],
7 #               [T1(tf), T2(tf), T3(tf), ... , Tn(tf)]]
8 #Es decir, Resultados[i][j] es la temperatura del nodo j en el tiempo ti
9 #Se coloca la temperatura inicial en la 1era fila de la matriz de resultados
10 Resultados[0] = T_anterior.flatten()
11 resultado = 'T'
12 #Se inicializan las matrices de coordenadas nodales rotadas
13 Xrot_anterior = np.zeros((nNodos,4))
14 Xrot = np.zeros((nNodos,4))
15 #Se avanza en el tiempo
16 with open('resultados.txt','w') as res:
17     for itera in range(nNodos-1):
18         res.write(f'{Resultados[0][itera]},')
19     res.write(f'{Resultados[0][-1]}')
20     res.write('\n')
21     while t<tf+dt:
22         #Angulo de rotacion en tiempo t
23         theta_anterior = omega*t+gamma*t**2/2
24         #Matriz de rotacion en tiempo t
```

```

25 Rθ_anterior = np.array([[np.cos(θ_anterior),-np.sin(θ_anterior),0],
26                        [np.sin(θ_anterior),np.cos(θ_anterior),0],
27                        [0,0,1]])
28 #Se calculan las coordenadas de los nodos en el tiempo t
29 for j in range(nNodos):
30     xyz_rot_anterior = np.matmul(Rθ_anterior,np.reshape(X[j][1:4],(3,1))).flatten()
31     Xrot_anterior[j] = [i+1,xyz_rot_anterior[0],xyz_rot_anterior[1],xyz_rot_anterior[2]]
32 #Angulo de rotacion en tiempo t+Δt
33 θ = ω0*t+γ*(t+Δt)**2/2
34 #Matriz de rotacion en tiempo t+Δt
35 Rθ = np.array([[np.cos(θ),-np.sin(θ),0],
36               [np.sin(θ),np.cos(θ),0],
37               [0,0,1]])
38 #Se calculan las coordenadas de los nodos en el tiempo t+Δt
39 for j in range(nNodos):
40     xyz_rot = np.matmul(Rθ,np.reshape(X[j][1:4],(3,1))).flatten()
41     Xrot[j] = [j+1,xyz_rot[0],xyz_rot[1],xyz_rot[2]]
42
43 #Se calculan las velocidades en el tiempo t
44 ω_anterior = ω0*(1-t/tf)
45 V_anterior = ω_anterior*Rr
46 #Se calculan las velocidades en el tiempo t+Δt
47 ω = ω0 + γ*(t+Δt)
48 V = ω*Rr
49 #Si la velocidad en t+Δt se vuelve negativa:
50 if ω <= 0 and ω_anterior>0:
51     #Se fijan las velocidades en (t+Δt) = 0
52     V = 0
53     ω = 0
54     #Se obtienen las matrices y vectores en t
55     M_anterior,N_anterior,K_anterior,Q_anterior = ensamble(X,X,T_anterior,elem_neumann,MC3D,
nElementos3D,CDA,ω_anterior,γ,m,n)
56     #Se obtienen las matrices y vectores en t+Δt
57     M,N,K,Q = ensamble(X,X,T,elem_neumann,MC3D,nElementos3D,CDA,ω,gamma,m,n)
58 #Si la velocidad en t se vuelve negativa:
59 elif ω_anterior <= 0:
60     #Se fijan las velocidades en (t) = 0
61     V_anterior = 0
62     ω_anterior = 0
63     #Se fijan las velocidades en (t+Δt) = 0
64     V = 0
65     ω = 0
66     #Se obtienen las matrices y vectores en t
67     M_anterior,N_anterior,K_anterior,Q_anterior = ensamble(X,X,T_anterior,elem_neumann,MC3D,
nElementos3D,CDA,ω_anterior,γ,m,n)
68     #Se obtienen las matrices y vectores en t+Δt
69     M,N,K,Q = ensamble(X,X,T,elem_neumann,MC3D,nElementos3D,CDA,ω,γ,m,n)
70     #Se define el paso temporal variable
71     Δt = Δt_max*(1-np.exp((tf-t)/tf)) + Δt_min
72     print(f'Paso temporal: {Δt}[s]')
73 #En cualquier otro caso (vehículo frenando):
74 else:
75     #Se obtienen las matrices y vectores en t
76     M_anterior,N_anterior,K_anterior,Q_anterior = ensamble(X,Xrot_anterior,T_anterior,
elem_neumann,MC3D,nElementos3D,CDA,ω_anterior,
77     γ,m,n)
78     #Se obtienen las matrices y vectores en t+Δt
79     M,N,K,Q = ensamble(X,Xrot,T,elem_neumann,MC3D,nElementos3D,CDA,ω,γ,m,n)
80
81 #Se calcula una matriz auxiliar
82 A = 1/Δt*((1-β)*M_anterior+β*M) + β*((1-β)*(N_anterior+K_anterior)+β*(N+K))
83 #Se calcula un vector auxiliar
84 b = (1-β)*Q_anterior + β*Q + (1/Δt*((1-β)*M_anterior+β*M)-(1-β)*((1-β)*(N_anterior+
K_anterior)+β*(N+K))).dot(T_anterior)
85 #Se calcula el nuevo vector de temperaturas en t+Δt (iteracion k+1)
86 T_nueva = np.linalg.solve(A,b)
87 #Chequeo de estabilidad (radio espectral <=1)
88 #Matrices auxiliares
89 C = 1/Δt*((1-β)*M_anterior+β*M)-(1-β)*((1-β)*(N_anterior+K_anterior)+β*(N+K))
90 D = np.matmul(np.linalg.inv(A),C)
91 #Radio espectral
92 radio_espectral = abs(max(np.linalg.eigvals(D)))
93 if radio_espectral<1.2:
94     print(f'Radio espectral: {radio_espectral} ≈ 1 => Estabilidad asegurada')
95 else:
96     print(f'Radio espectral: {radio_espectral} > 1 => Estabilidad no asegurada')
97 #Se compara la nueva temperatura (k+1) con la anterior (k)
98 for j in range(nNodos):
99     er[j] = 100*(abs(T_nueva[j][0]-T[j][0]))/T_nueva[j][0]

```



```

100     #Si el error maximo es menor o igual a la tolerancia:
101     if max(er)<=tol:
102         #Se guarda el vector de temperaturas T(t+Δt) en la matriz de resultados
103         Resultados[p] = T_nueva.flatten()
104         print(f'Se ha guardado el vector T({t+Δt}[s]) en {time()-tiempo_inicial}[s] de tiempo
computacional acumulado')
105         print(f'Maximo error relativo: {max(er)} % < {tol} %')
106         print()
107         #Se actualiza el valor de los vectores de temperaturas: T(t) y
108         #T(t+Δt)(k=0) por el nuevo: T_nueva
109         T_anterior = T_nueva
110         T = T_nueva
111         #Se incrementa un paso temporal
112         t += Δt
113         #Se guardan los resultados en un .txt en formato CSV (separado por ,)
114         for i in range(nNodos-1):
115             res.write(f'{Resultados[p][i]},')
116             res.write(f'{Resultados[p][-1]}')
117             res.write('\n')
118             p += 1
119         #Se vuelve a inicializar el contador de iteraciones
120         i = 0
121         #Salida de emergencia del bucle
122         if t>tf+δt:
123             break
124     #En caso contrario,
125     else:
126         print(f't={t+Δt} s, iteracion {i} completada en {time()-tiempo_inicial}[s] de tiempo
computacional acumulado')
127         print(f'Maximo error relativo: {max(er)} % > {tol} %')
128         print()
129         #Se actualiza el valor del vector de temperaturas con las nuevas
130         #para seguir iterando en el mismo paso temporal
131         T = T_nueva
132         i += 1

```

### 3.3. Post-procesado

En esta sección, se realiza el tratamiento de los resultados obtenidos para su visualización. Al tratarse de un problema tridimensional transitorio, la visualización más completa de los resultados es a través de un video que muestre la evolución de la temperatura en el tiempo en todo el dominio. Para realizarlo, se utiliza la librería **Pyvista** de **Python**.

```

1  #POST-PROCESSING
2  #En esta seccion, se realiza el tratamiento de los resultados para su visualizacion
3  #Se eliminan las filas con ceros que quedaron
4  Resultados = Resultados[~np.all(Resultados == 0, axis=1)]
5  #Temperatura maxima alcanzada
6  Tmax = max(Resultados[-1])
7  #Nombre del video que sera guardado
8  nombre = 'video.mp4'
9  #Objeto de la malla inicial
10 msh = pv.read('Malla5//disco.msh')
11 #Temperatura inicial de la malla
12 msh.point_data['Temperatura [K]'] = Resultados[0]
13 #Seteo de la camara
14 camera = pv.Camera()
15 camera.position = (-0.5, 0.5, 1)
16 camera.focal_point = (0, 0, 0)
17 #Inicializacion del graficador
18 p = pv.Plotter()
19 #Agregado de los ejes coordenados
20 p.add_axes_at_origin()
21 #Apertura del modo video
22 p.open_movie(nombre, framerate=1/Δt)
23 #Asignacion de la camara
24 p.camera = camera
25 #Asignacion de la malla inicial
26 p.add_mesh(msh, scalars='Temperatura [K]', show_edges=True, clim=[T0, Tmax], cmap='autumn_r')
27 print('Orient the view, then press "q" to close window and produce movie')
28 #Grafico inicial (sin cerrar la ventana)
29 p.show(auto_close=False)
30 p.write_frame()
31 #Recorrido de los pasos temporales con rotacion del disco
32 for i in range(int((1+(tf-t0)/Δt))):
33     #Definicion de la variable tiempo
34     t = Δt*i
35     #Asignacion del vector de temperaturas T(t=ti)

```

```

36 msh.point_data['Temperatura [K]'] = Resultados[i]
37 #Eliminacion de la malla en tiempo anterior
38 p.clear()
39 #Rotacion de la malla en tiempo t
40 rot = msh.rotate_z( $\omega_0*t + \gamma*t**2/2$  , inplace=False)
41 #Asignacion de la malla rotada con las temperaturas actualizadas
42 p.add_mesh(rot, scalars='Temperatura [K]', show_edges=True, clim=[T0, Tmax], cmap='autumn_r')
43 #Se escribe el tiempo actual
44 p.add_text(f't = {round(t,1)} s', font='times')
45 #Actualizacion del graficador
46 p.update()
47 p.write_frame()
48 #Recorrido de los pasos temporales siguientes sin rotacion del disco
49 for i in range(int((tf-t0)/ $\Delta t$ ), int(1+(tf-t0)/ $\Delta t$  +  $\delta t/\Delta t_{max}$ )):
50     #Definicion de la variable tiempo
51     t = round( $\Delta t*i + \Delta t_{max}*(i-(tf-t0)/\Delta t)$ , 1)
52     #Asignacion del vector de temperaturas T(t=ti)
53     msh.point_data['Temperatura [K]'] = Resultados[i]
54     #Eliminacion de la malla en tiempo anterior
55     p.clear()
56     #Asignacion de la malla inicial con las temperaturas actualizadas
57     p.add_mesh(msh, scalars='Temperatura [K]', show_edges=True, clim=[T0, Tmax], cmap='autumn_r')
58     #Se escribe el tiempo actual
59     p.add_text(f't = {t} s', font='times')
60     #Actualizacion del graficador
61     p.update()
62     p.write_frame()
63 #Cierre del graficador
64 p.close()

```

Adicionalmente, con la misma librería, abriendo otro graficador, se realizan imágenes de secciones del disco. Se realiza un corte con un plano vertical, es decir, con ángulos  $\theta = \{\frac{\pi}{2}; \frac{3\pi}{2}\}$  y un otro con un plano horizontal, es decir,  $\theta = \{0; \pi\}$  (cortes A-A y B-B respectivamente de la Figura 23).

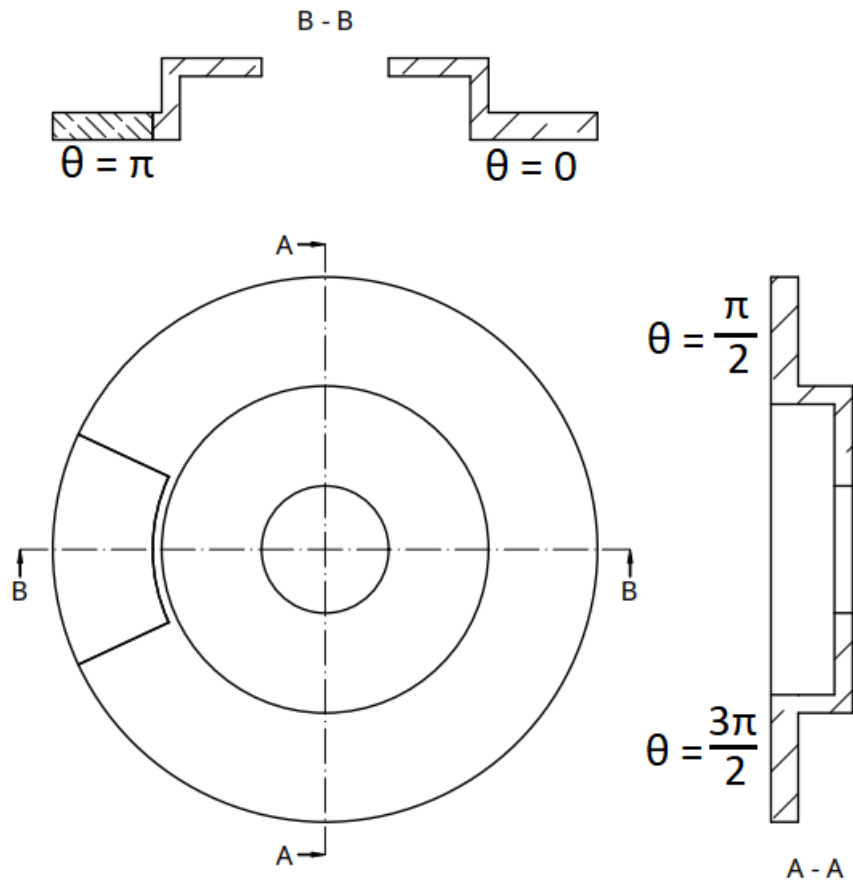


Figura 23: Cortes realizados. El corte vertical A-A y el horizontal B-B.

Tal como se verá en los resultados, se rotó el corte horizontal B-B 90 grados en sentido horario, de manera que la sección resultante sea coincidente con la del corte A-A, para lograr una mayor claridad en la comparación de resultados.

## 4. Verificación del código

Antes de usar el programa desarrollado para resolver el problema del disco de frenos, se lo utiliza para resolver un problema con solución analítica conocida: un problema de transferencia de calor unidimensional en régimen transitorio con una condición de borde de Neumann en un extremo. La idea es comparar la solución obtenida numéricamente con una solución analítica para verificar el modelo.

Se genera una nueva geometría y se realiza una malla no estructurada de elementos tetragonales y triangulares tal como se explicó en la sección 3.1.

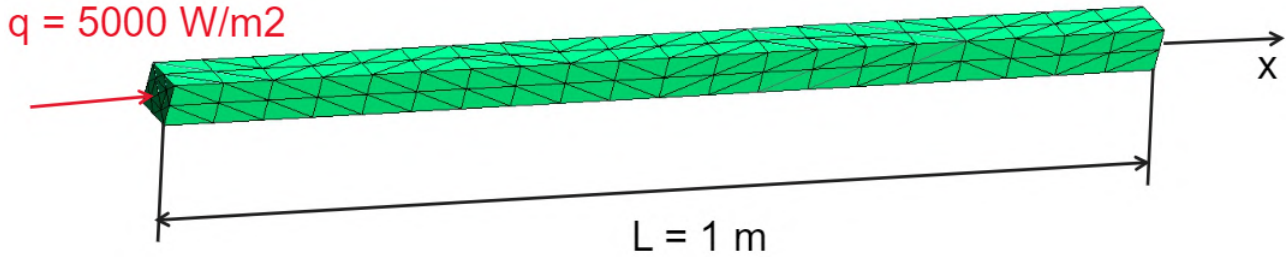


Figura 24: Malla de la barra de longitud unitaria utilizada para la verificación del modelo

Se sabe de la teoría que para tiempos cortos, es decir, si:

$$Fo = \frac{\alpha t}{L^2} < 0,05$$

Se puede utilizar la solución analítica del modelo de sólido semi-infinito:

$$T(x, t) = T_0 + \frac{q}{k} \left[ \sqrt{\frac{4\alpha t}{\pi}} e^{-\frac{x^2}{4\alpha t}} - x \left( 1 - \text{ferr} \left( \frac{x}{\sqrt{4\alpha t}} \right) \right) \right]$$

Se toma un valor fijo de la difusividad térmica de la fundición gris del disco, eligiendo una temperatura cualquiera arbitrariamente:

$$\alpha = \alpha|_{T=500\text{ K}} = 1,1 \cdot 10^{-5} \frac{\text{m}^2}{\text{s}}$$

Siendo la longitud característica igual a 1 m, se puede determinar hasta qué tiempo es prudente utilizar este modelo:

$$t < \frac{0,05 L^2}{\alpha} = 4545,7 \text{ s}$$

Entonces, ejecutando el código desarrollado previamente, para una temperatura inicial  $T_0 = 500 \text{ K}$ , imponiendo velocidad nula  $\omega = 0$ , con un flujo de calor constante  $q = 5000 \frac{\text{W}}{\text{m}^2}$ , se grafican las soluciones analítica y de elementos finitos.

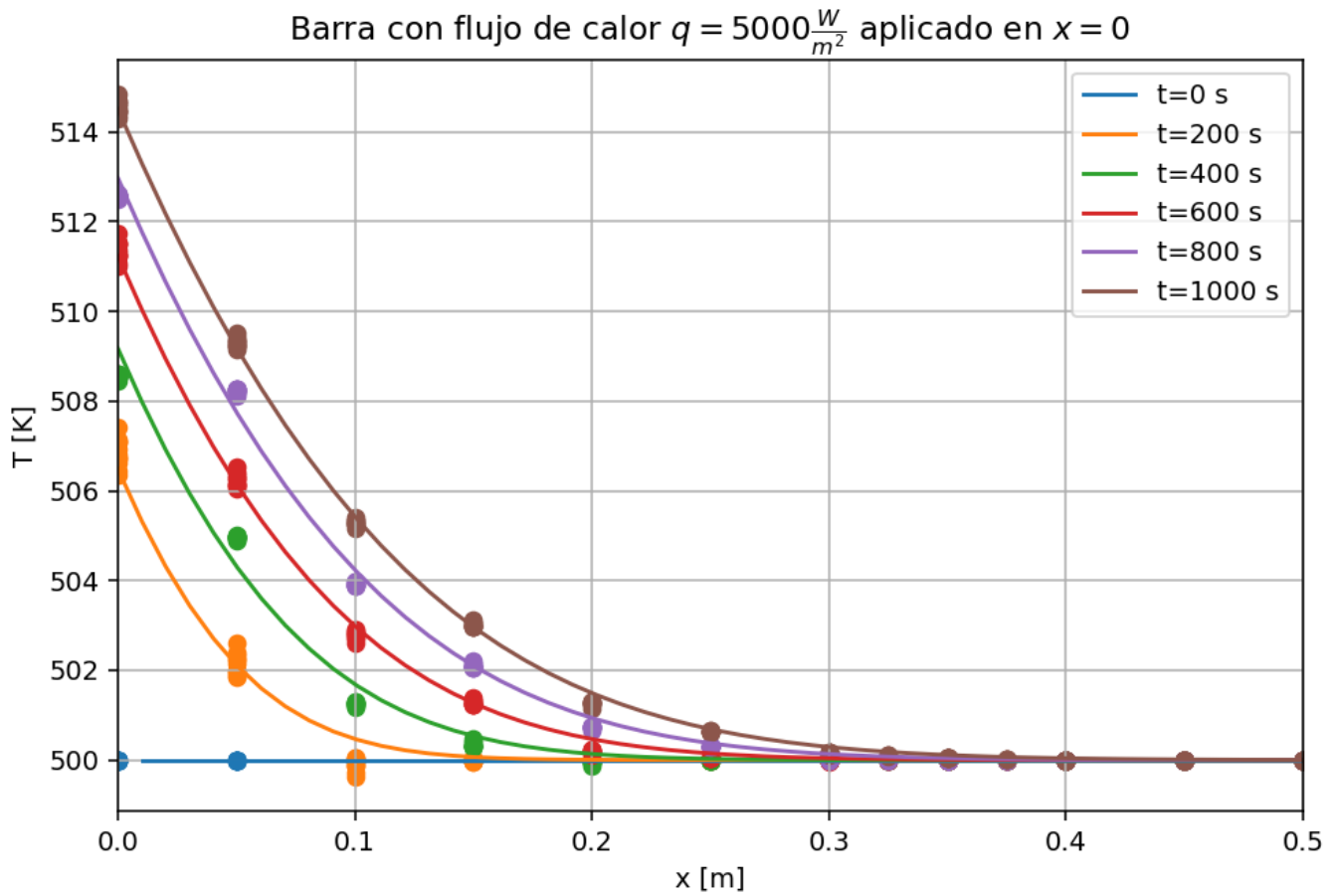


Figura 25: Gráfico de la solución analítica de sólido semi-infinito (trazo continuo) y la solución de elementos finitos (puntos)

Se observan buenos resultados obtenidos mediante el método de elementos finitos, aun con una malla gruesa y con un paso temporal grosero de 200 segundos. Se considera entonces verificado el código de elementos finitos, por lo tanto, se procede a su implementación para resolver el problema del disco de frenos.

## 5. Implementación del código

Volviendo al problema del disco de frenos, se comienza eligiendo los parámetros físicos de la simulación.

$t_f$ [s]	$\delta t$ [s]	$m$ [kg]	$C_D A_R$ [m <sup>2</sup> ]	$\rho_{\text{aire}}$ [ $\frac{\text{kg}}{\text{m}^3}$ ]	$T_\infty$ [°C]	$T_0$ [°C]	$R_r$ [mm]
5	300	1200	0,5	1,2	25	25	300

Tabla 2: Parámetros físicos escogidos para la simulación

Siendo:

- $t_f$  el tiempo de frenada
- $\delta t$  la ventana de tiempo adicional con el vehículo detenido
- $m$  la masa del vehículo
- $C_D A_R$  el coeficiente de arrastre multiplicado por el área de referencia
- $\rho_{\text{aire}}$  la densidad del aire que arrastra el vehículo
- $T_\infty$  la temperatura del aire exterior
- $T_0$  la temperatura inicial del disco
- $R_r$  el radio de las ruedas del vehículo

Además, se obtiene el área de la superficie de contacto por fricción ( $S_q$ ) del **Onshape**:  $A_f = 0,00587958 \text{ m}^2$

Luego, se eligen los siguientes parámetros numéricos:

$\Delta t_{\min} [\text{s}]$	$\Delta t_{\max} [\text{s}]$	tol [%]	$n_{\text{Gauss}}$
0,1	10	1	2

Tabla 3: Parámetros numéricos escogidos para la simulación

- $\Delta t_{\min}$  es el paso temporal utilizado en la ventana de tiempo inicial, con  $\omega > 0$  (vehículo frenando)
- $\Delta t_{\max}$  es el máximo paso temporal alcanzado en la ventana de tiempo siguiente, con  $\omega = 0$  (vehículo detenido)
- tol es la tolerancia que determina la convergencia de cada iteración para un tiempo  $t$
- $n_{\text{Gauss}}$  es la cantidad de puntos de Gauss considerados en cada dirección

El paso temporal varía tal como se muestra en la Figura 26, de acuerdo a la ecuación:

$$\Delta t = \begin{cases} \Delta t_{\min} & \text{si } t \leq t_f \quad \omega \geq 0 \\ \Delta t_{\max} \left(1 - e^{-\frac{t_f - t}{t_f}}\right) + \Delta t_{\min} & \text{si } t > t_f \quad \omega = 0 \end{cases}$$

Notar que, como el tiempo está discretizado según:  $t_{i+1} = t_i + \Delta t$ , el paso temporal es una función discreta del tiempo.

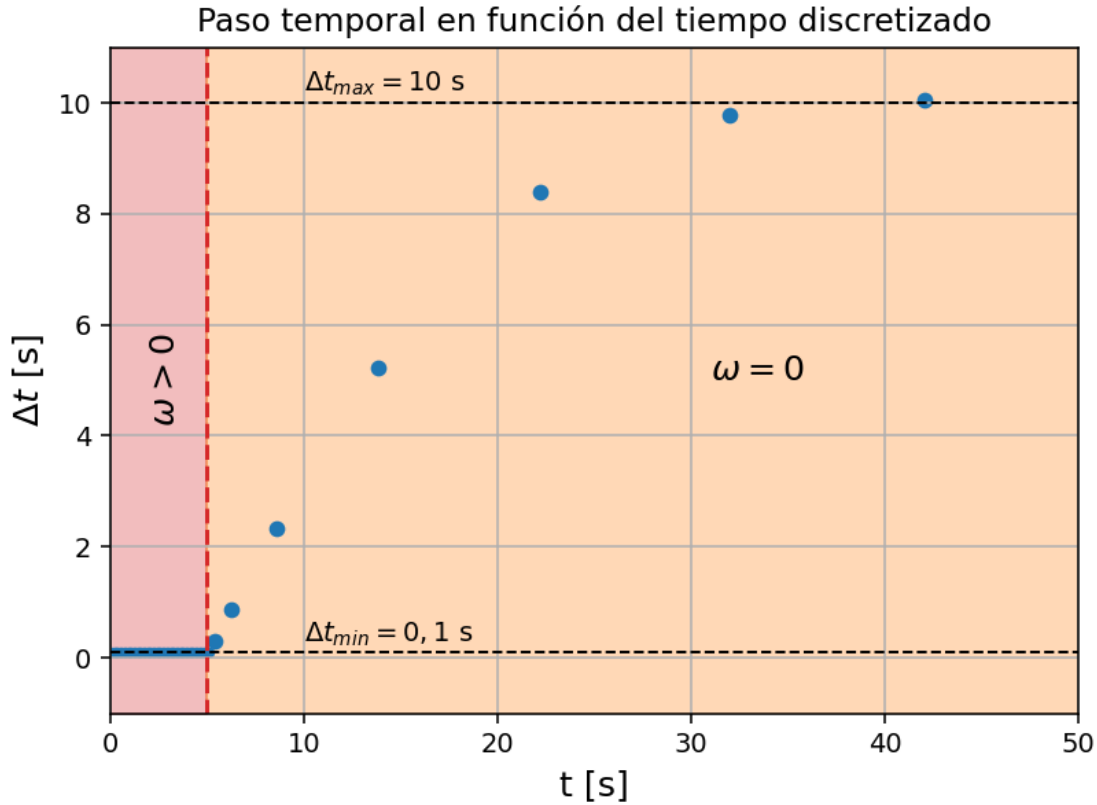


Figura 26: Variación del paso temporal en el tiempo. Se remarcan dos zonas temporales: en rojo, el periodo de tiempo con el vehículo frenando, en el cual el paso temporal es constante e igual al mínimo; y en naranja, el periodo de tiempo con el vehículo detenido, en el cual el paso temporal aumenta hasta llegar a un valor máximo.

A continuación, se realizan simulaciones con distintas velocidades iniciales, manteniendo constantes los datos anteriores.

## 5.1. Resultados para distintas velocidades iniciales

### 5.1.1. Caso 1: $V_0 = 50 \frac{\text{km}}{\text{h}}$

En un primer caso, se fija la velocidad inicial en  $V_0 = 50 \frac{\text{km}}{\text{h}}$ , obteniendo los resultados siguientes ([video](#)), donde se observa una temperatura máxima:  $T_{\max} = 311,77 \text{ K}$ . Tal como se explicó en la sección de Post-procesado, además del video de la malla tridimensional, se grafican los cortes vertical y horizontal del disco para los tiempos  $t = 2,5 \text{ s}$  y  $t = 5 \text{ s}$ .

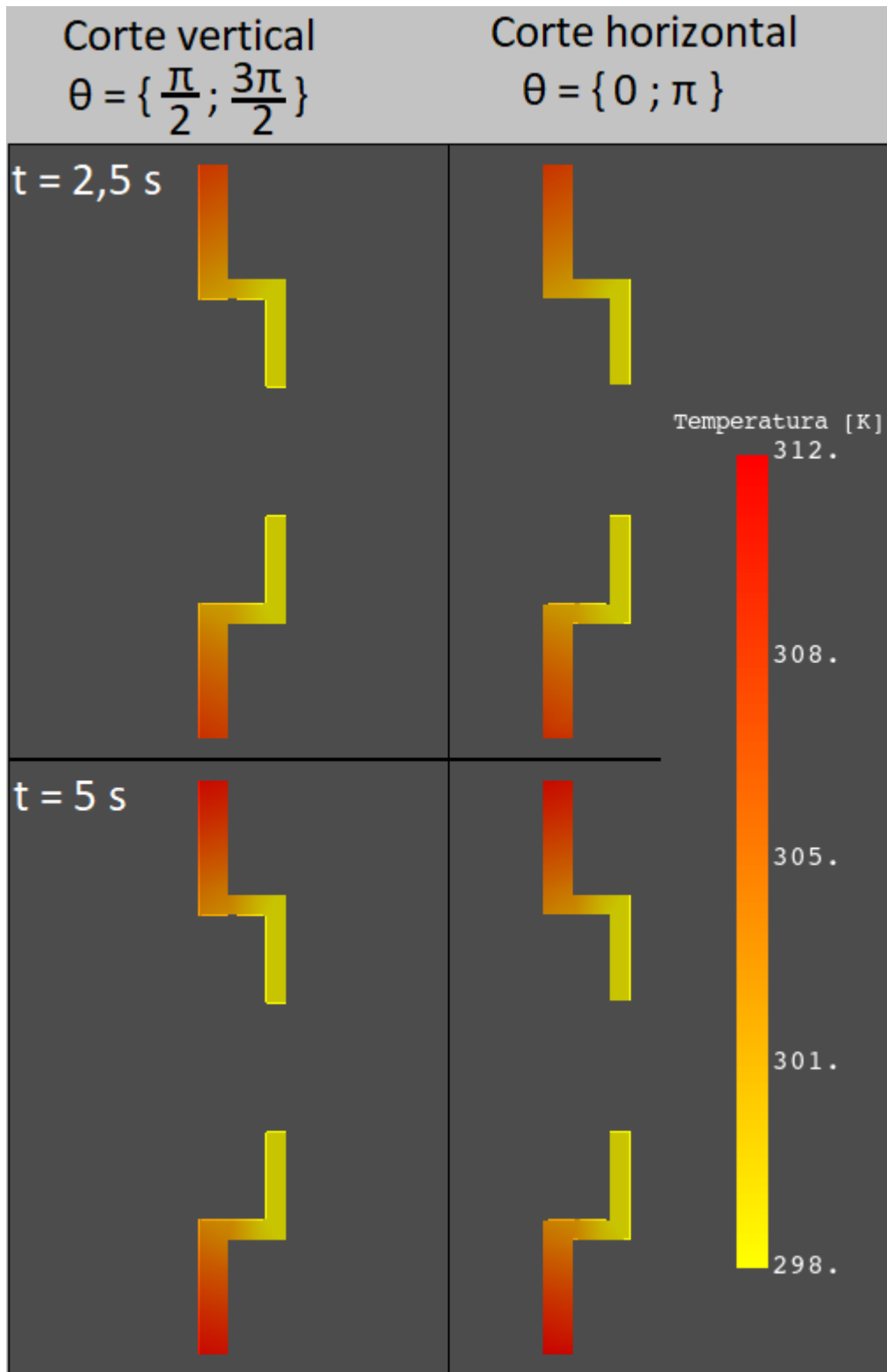


Figura 27: Cortes vertical y horizontal del disco, para  $t = 2,5 \text{ s}$  y  $t = 5 \text{ s}$ , con velocidad inicial  $V_0 = 50 \frac{\text{km}}{\text{h}}$

#### 5.1.2. Caso 2: $V_0 = 100 \frac{\text{km}}{\text{h}}$

En un segundo caso, se ejecuta el código desarrollado con una velocidad inicial:  $V_0 = 100 \frac{\text{km}}{\text{h}}$ , obteniendo los resultados expuestos en [este video](#). Tal como se muestra en el video, la temperatura máxima alcanzada por el disco fue:  $T_{\text{max}} = 349,59 \text{ K}$ . Se muestran también los cortes:

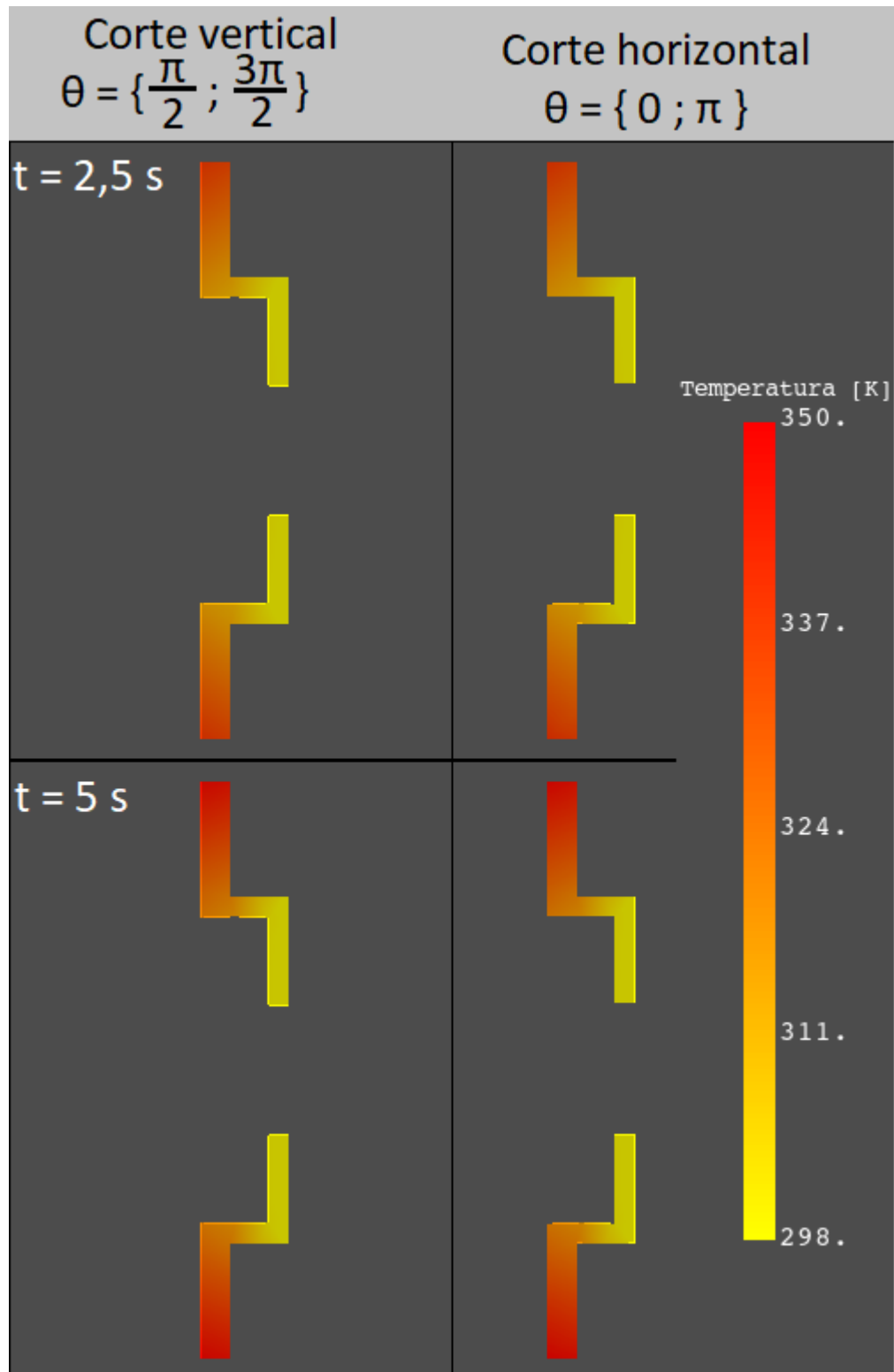


Figura 28: Cortes vertical y horizontal del disco, para  $t = 2,5 \text{ s}$  y  $t = 5 \text{ s}$ , con velocidad inicial  $V_0 = 100 \frac{\text{km}}{\text{h}}$

### 5.1.3. Caso 3: $V_0 = 150 \frac{\text{km}}{\text{h}}$

En un tercer caso, se impone una velocidad inicial de  $150 \frac{\text{km}}{\text{h}}$ , manteniendo el tiempo de frenada en 5 segundos. Se obtienen los resultados de [este video](#). En este caso, se obtuvo una temperatura máxima de 406,72 K.

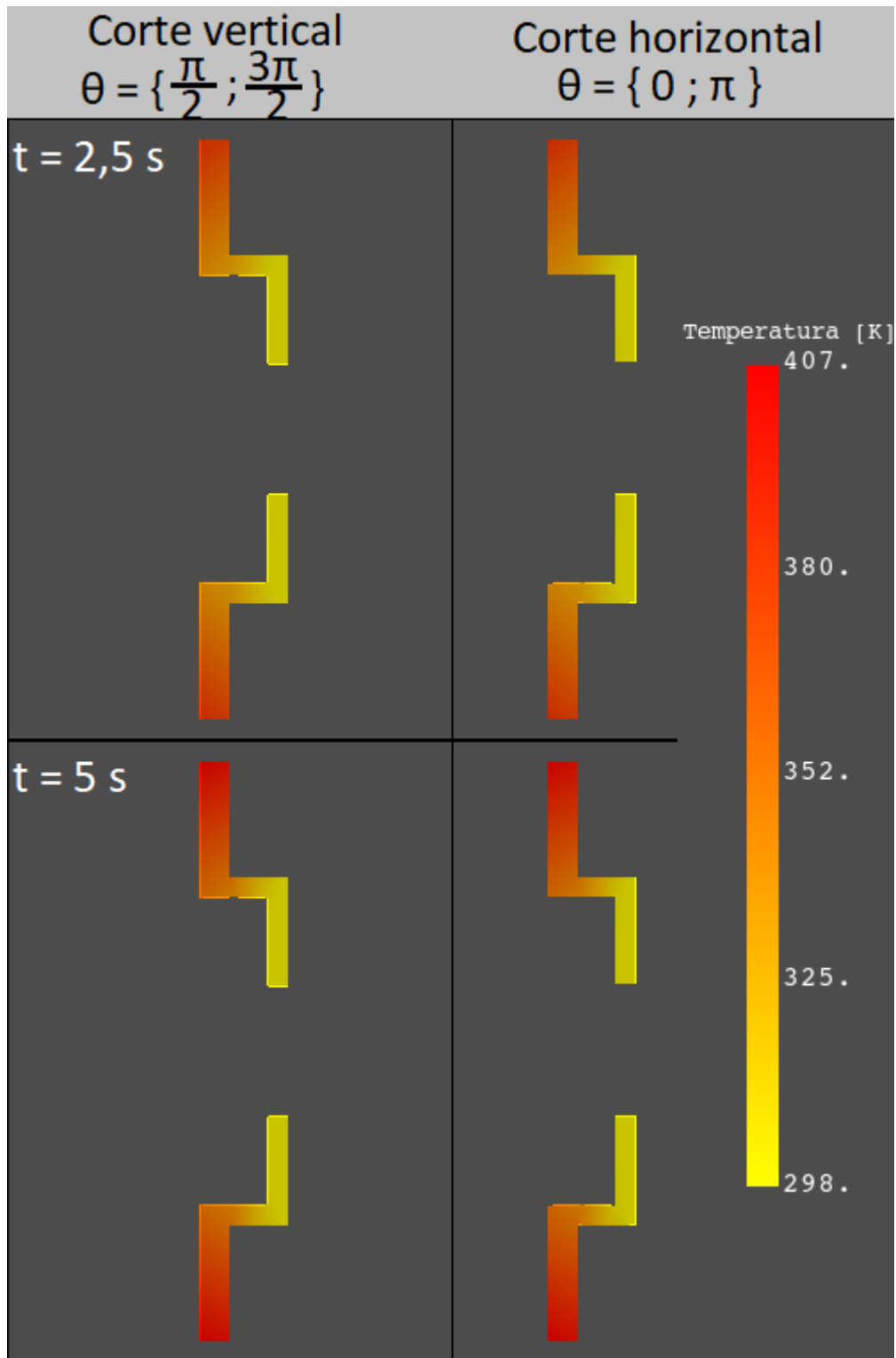


Figura 29: Cortes vertical y horizontal del disco, para  $t = 2,5 \text{ s}$  y  $t = 5 \text{ s}$ , con velocidad inicial  $V_0 = 150 \frac{\text{km}}{\text{h}}$

#### 5.1.4. Caso 4: $V_0 = 200 \frac{\text{km}}{\text{h}}$

Si se define como condición inicial una velocidad de  $200 \frac{\text{km}}{\text{h}}$ , se obtienen los resultados mostrados en [este video](#), donde se puede ver que  $T_{\text{max}} = 481,28 \text{ K}$ .



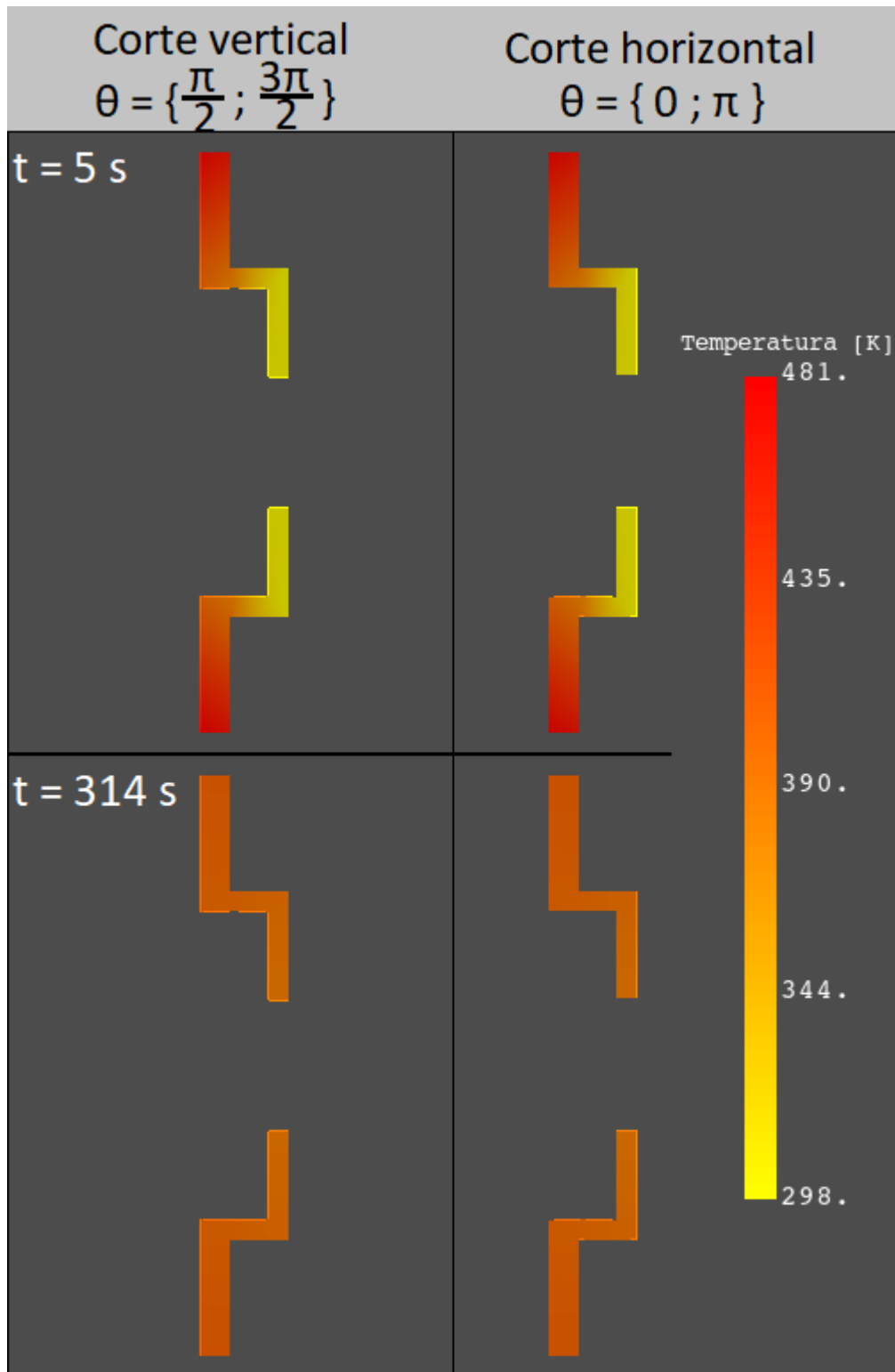


Figura 30: Cortes vertical y horizontal del disco, para  $t = 5 \text{ s}$  y  $t = 314 \text{ s}$ , con velocidad inicial  $V_0 = 200 \frac{\text{km}}{\text{h}}$

#### 5.1.5. Caso 5: $V_0 = 250 \frac{\text{km}}{\text{h}}$

Ejecutando el código con velocidad inicial  $V_0 = 250 \frac{\text{km}}{\text{h}}$ , se obtienen resultados como los mostrados en [este video](#), del cual se extraen capturas de algunos tiempos, expuestas en la Figura 32, donde se observa una temperatura máxima de 568,1 K. Se grafican también los cortes como en los casos anteriores:



Figura 31: Cortes vertical y horizontal del disco, para  $t = 5 \text{ s}$  y  $t = 314 \text{ s}$ , con velocidad inicial  $V_0 = 250 \frac{\text{km}}{\text{h}}$

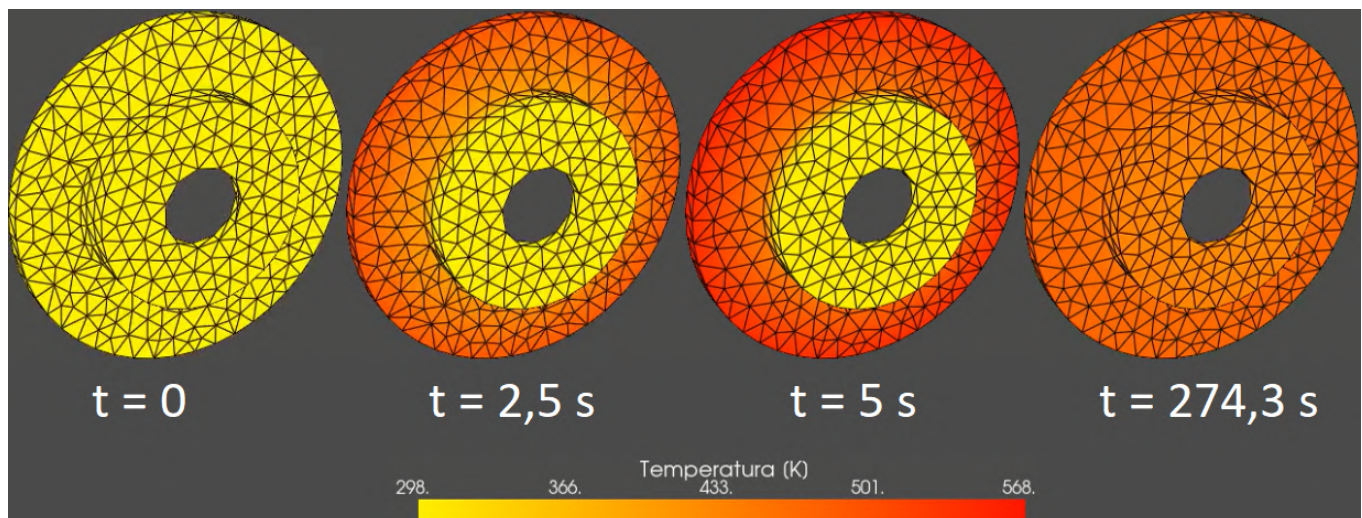


Figura 32: Distribución térmica para algunos tiempos, con velocidad inicial de  $V_0 = 250 \frac{\text{km}}{\text{h}}$

## 5.2. Análisis de los resultados

En base a los resultados obtenidos, se puede hacer un breve análisis. Se observa como en el comienzo de la simulación, la temperatura aumenta rápidamente en toda la periferia del disco, debido probablemente a los efectos convectivos de la formulación Euleriana que se ha planteado, alcanzando su valor máximo en el tiempo final de frenada ( $t_f = 5$  s). Luego, se aprecia como, una vez que el vehículo se detiene, al no haber efectos convectivos, el calor difunde hacia el centro del disco, uniformizando la temperatura en todo su volumen.

Es importante aclarar que no se introdujo una condición de borde de convección para extraer el calor del disco una vez que se frenó el vehículo, por eso, se mantiene la temperatura y no disminuye considerablemente. Disminuye levemente en las zonas más calientes (exterior del disco) porque el calor fluye de éstas hacia las zonas más frías (interior). El objetivo de dejar correr la simulación un tiempo adicional con el vehículo detenido era simplemente observar este fenómeno de difusión de calor hacia el centro del disco.

## 5.3. Consideraciones del modelo

### 5.3.1. Influencia del planteo tridimensional

Se puede apreciar en los resultados, que mismo si la generación de calor se produce únicamente en la superficie de contacto  $S_q$ , toda la región circunferencial para los mismos radios eleva su temperatura rápidamente debido al término convectivo en la ecuación (esto es especialmente notable a grandes velocidades de rotación). Por lo tanto, se pudo haber planteado un modelo en el cual se aplicara la condición de borde sobre toda la región circunferencial (entre los radios que definen la zona de contacto). En ese caso, el problema pasaría a ser uno con axil-simetría, por lo que, se podría eliminar la coordenada circunferencial y trabajar en 2 dimensiones:  $r$ , y  $z$ . Esto se puede corroborar con los cortes realizados en cada caso (Figuras 27, 28, 29, 30, 31), donde se observa la misma distribución de temperaturas en ambos cortes vertical y horizontal, para cada tiempo. Es decir, se puede concluir entonces que es totalmente válido plantear un modelo axil-simétrico. Además, se puede apreciar también en los cortes que la temperatura no varía considerablemente en la coordenada  $z$  seguramente debido al pequeño espesor del disco, excepto por la región donde cambia el espesor (el cambio de plano  $xy$ ). Entonces, análogamente a los estados planos de tensión en la teoría lineal de la elasticidad, despreciando la variación en la dirección  $z$ , el problema se podría plantear como un estado plano con simetría de revolución. De esta manera, se elimina la dependencia de las coordenadas  $\theta$  (por la simetría de revolución) y  $z$  (por el pequeño espesor), quedando únicamente la coordenada radial  $r$ . Con este planteo simplificado, se reducirían los tiempos computacionales y hasta se podría trabajar con mallas más finas.

Se define una coordenada pseudo-radial  $r'$  como la línea media del espesor del disco. Es básicamente la componente radial de las coordenadas cilíndricas, con la salvedad de que aumenta también en el cambio de plano, tal como se muestra en la Figura 33.

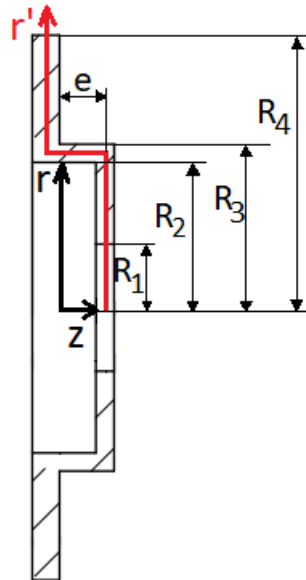


Figura 33: Coordenada pseudo-radial

Matemáticamente, se puede definir esta coordenada como:

$$r' = \begin{cases} r & \text{si } R_1 \leq r < R_2 \\ r + e - z & \text{si } R_2 \leq r \leq R_4 \end{cases}$$

Siendo  $r$  la coordenada radial de cilíndricas:  $r = \sqrt{x^2 + y^2}$ .

Con las dimensiones del plano del disco, se puede determinar:

$$\begin{aligned}
 e &= 30 \text{ mm} - \frac{10 \text{ mm}}{2} = 25 \text{ mm} \\
 R_1 &= \frac{70 \text{ mm}}{2} = 35 \text{ mm} \\
 R_2 &= \frac{160 \text{ mm}}{2} = 80 \text{ mm} \\
 R_3 &= \frac{160 \text{ mm}}{2} + 10 \text{ mm} = 90 \text{ mm} \\
 R_4 &= \frac{300 \text{ mm}}{2} = 150 \text{ mm}
 \end{aligned}$$

Se define además el valor máximo que puede tomar la coordenada pseudo-radial:  $R_T = 190 \text{ mm}$ .

De esta manera, se puede graficar la temperatura en función de la coordenada pseudo-radial adimensionalizada respecto del valor máximo, para distintos instantes de tiempo, y para cada caso de velocidad inicial.

En las Figuras mostradas a continuación, se observa dispersión en los resultados. Esto era esperable ya que se condensaron datos tridimensionales en una única dimensión, tal como se había hecho en la verificación del modelo (sección 4).

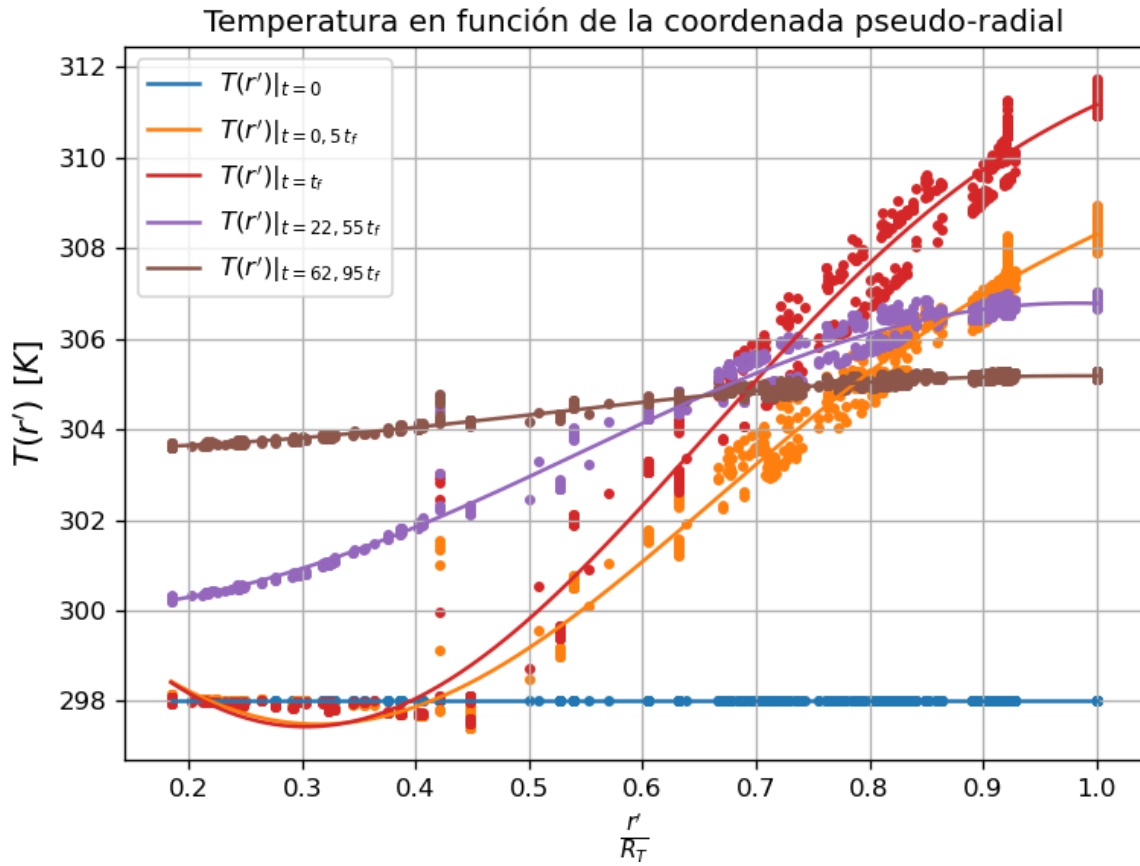


Figura 34: Temperaturas nodales en función de la coordenada pseudo-radial  $r'$ , para  $V_0 = 50 \frac{\text{km}}{\text{h}}$ , en distintos instantes de tiempo en relación al tiempo de frenada  $t_f$

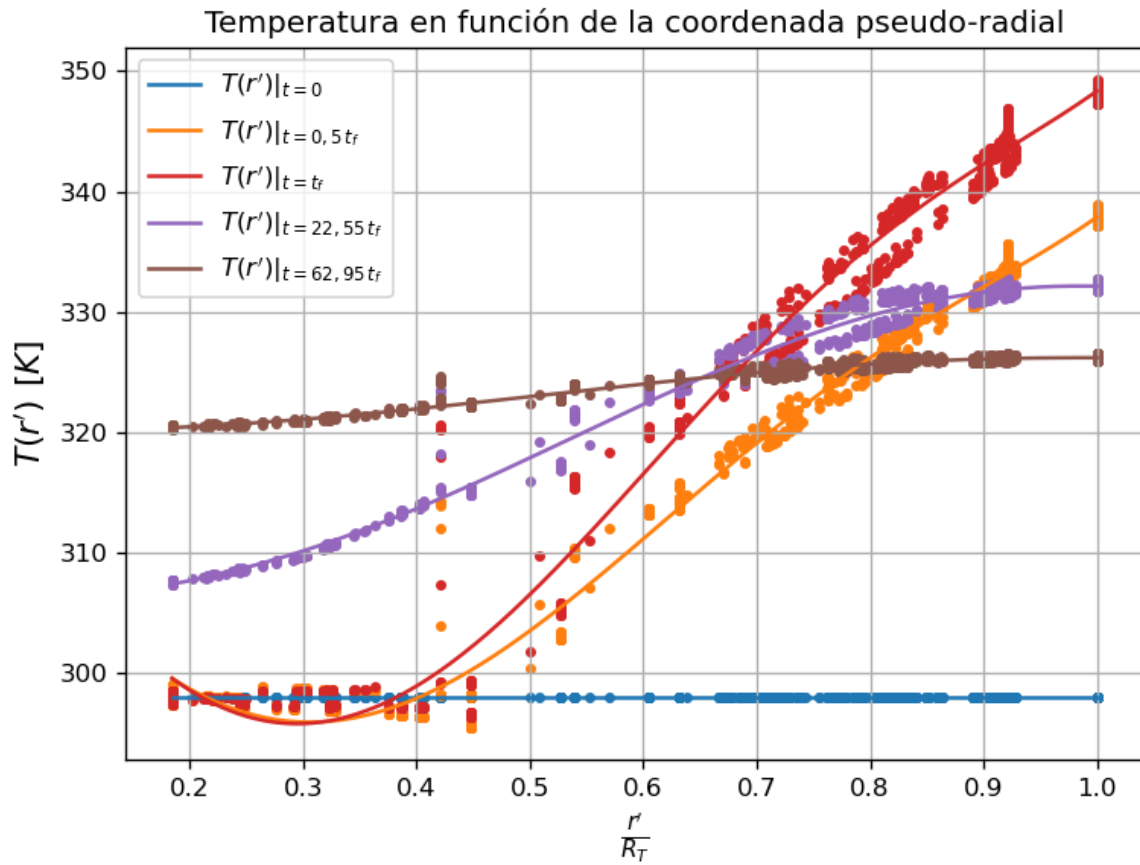


Figura 35: Temperaturas nodales en función de la coordenada pseudo-radial  $r'$ , para  $V_0 = 100 \frac{\text{km}}{\text{h}}$ , en distintos instantes de tiempo en relación al tiempo de frenada  $t_f$

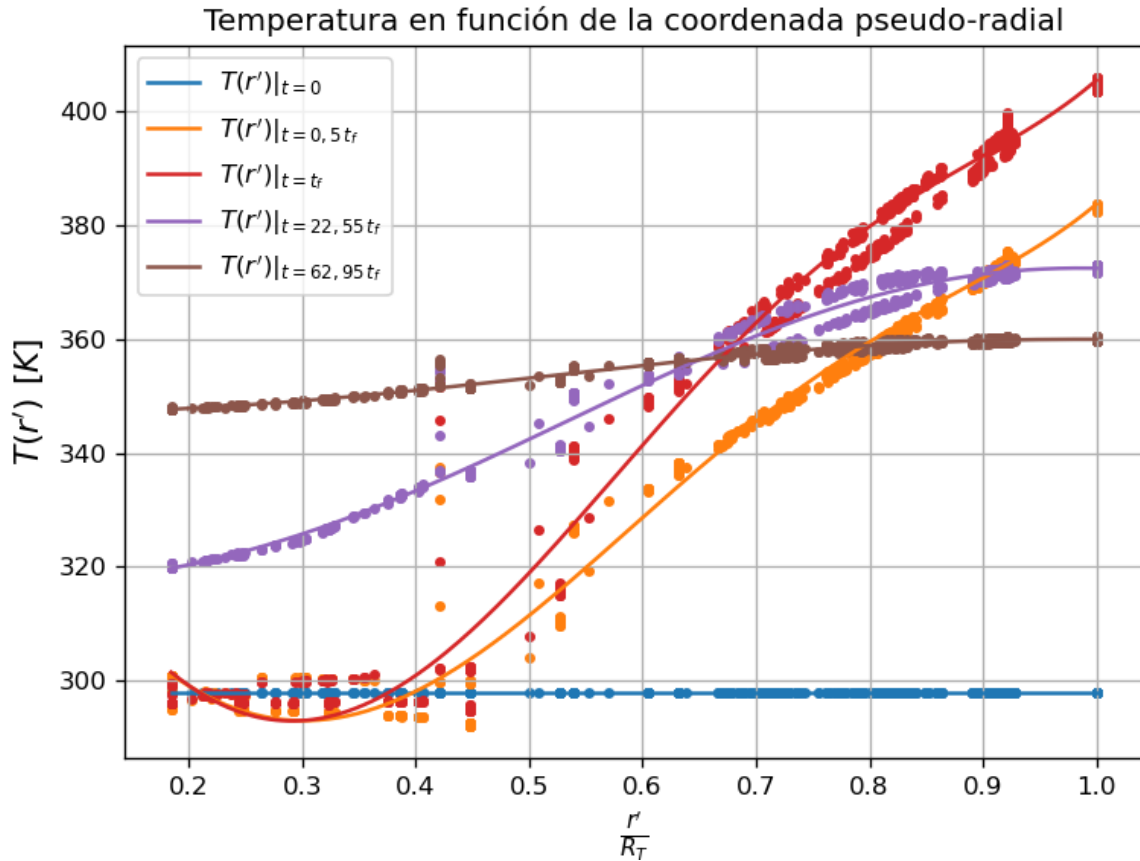


Figura 36: Temperaturas nodales en función de la coordenada pseudo-radial  $r'$ , para  $V_0 = 150 \frac{\text{km}}{\text{h}}$ , en distintos instantes de tiempo en relación al tiempo de frenada  $t_f$

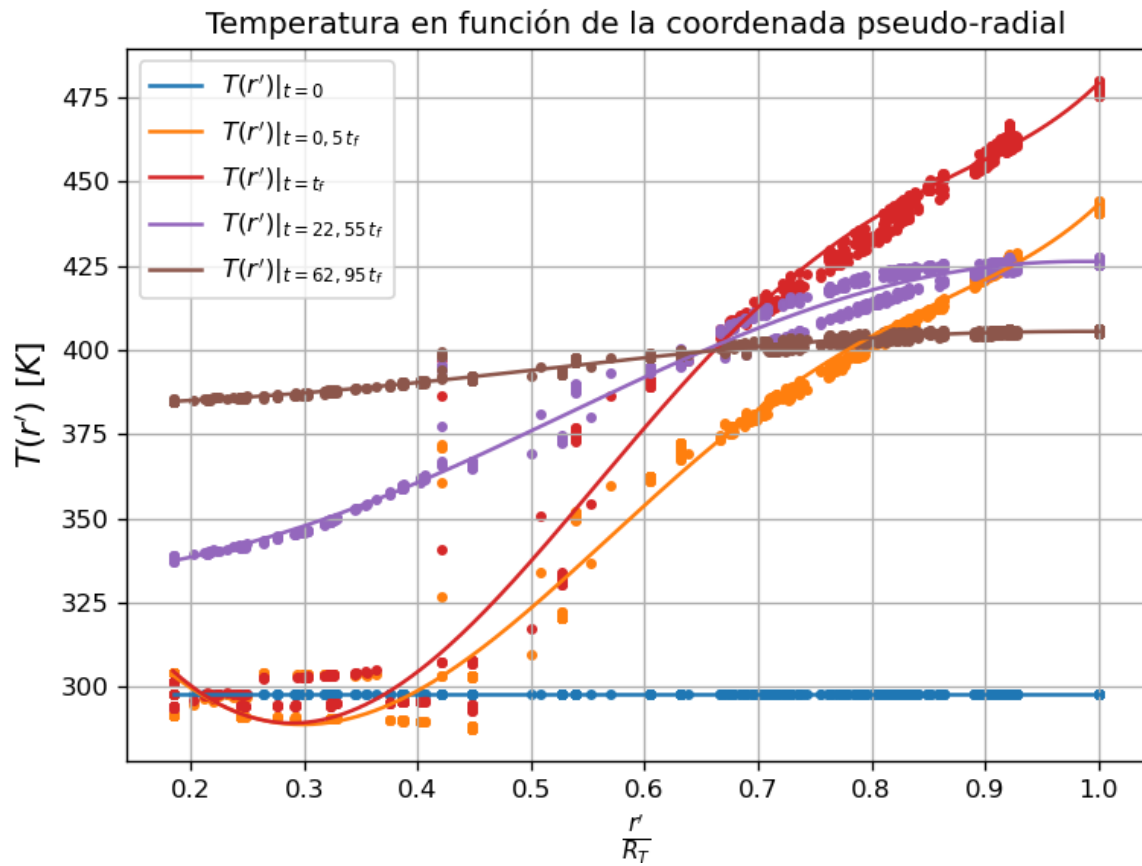


Figura 37: Temperaturas nodales en función de la coordenada pseudo-radial  $r'$ , para  $V_0 = 200 \frac{\text{km}}{\text{h}}$ , en distintos instantes de tiempo en relación al tiempo de frenada  $t_f$

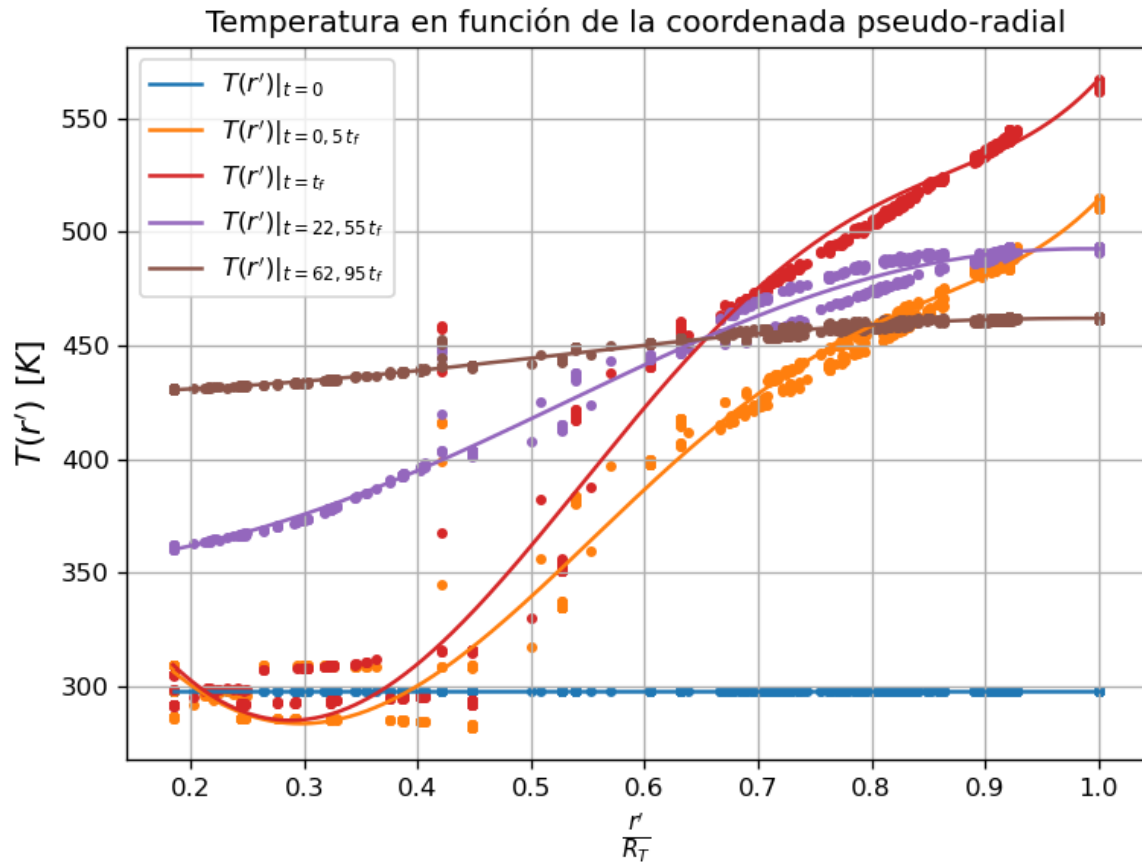


Figura 38: Temperaturas nodales en función de la coordenada pseudo-radial  $r'$ , para  $V_0 = 250 \frac{\text{km}}{\text{h}}$ , en distintos instantes de tiempo en relación al tiempo de frenada  $t_f$

### 5.3.2. Influencia de la no linealidad material

Para estimar qué tan importante es considerar la no linealidad del material, es decir, la dependencia de la temperatura en relación a las propiedades como la conductividad y la difusividad térmica, se resolverá el mismo problema pero considerando propiedades físicas constantes, y se compararán los resultados. Se toman las propiedades constantes para una determinada temperatura de referencia, que se puede calcular o tomar arbitrariamente. En este caso se proponen dos métodos:

- Calcular la temperatura media en base a los resultados anteriores:  $T_{\text{ref}} = \frac{T_{\text{max}} + T_{\text{min}}}{2} = \frac{T_{\text{max}} + T_0}{2}$
- Tomar la temperatura inicial:  $T_{\text{ref}} = T_0$

En el primer caso, se puede suponer que los resultados serán más acertados en relación a los obtenidos mediante el análisis no lineal. Pero, para ello, se necesitó realizar el análisis no lineal para determinar la temperatura máxima alcanzada. Si se tuviera que resolver el problema desde cero, y se lo planteara linealmente, habría que estimar la temperatura máxima de alguna otra forma. Por lo tanto, imaginando la situación hipotética en la cual no se puede estimar el rango de temperaturas (suponiendo que no se realizó el análisis no lineal), se tomará la temperatura inicial como la de referencia:  $T_{\text{ref}} = T_0$ .

Se grafica a continuación la temperatura máxima alcanzada en las simulaciones para cada velocidad inicial, comparando los modelos lineal y no lineal.

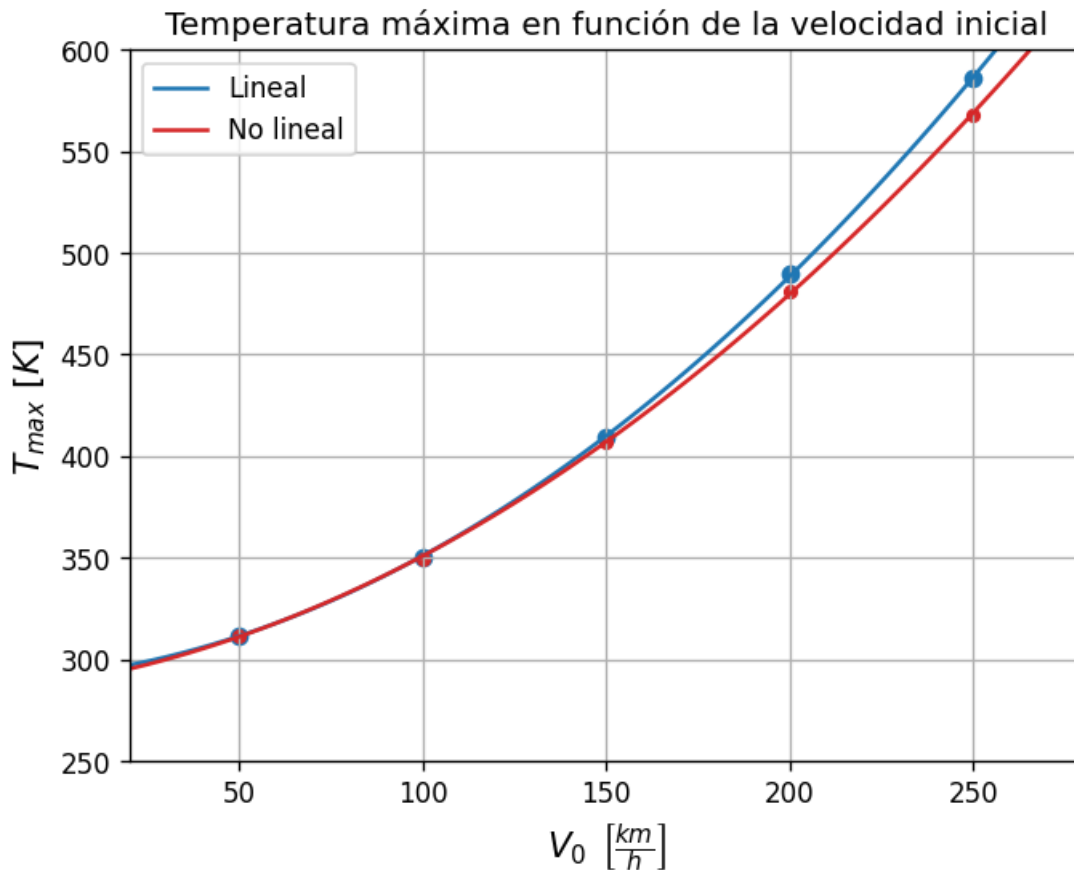


Figura 39: Temperatura máxima alcanzada en cada caso en función de la velocidad inicial, para un mismo tiempo de frenada  $t_f = 5$  s, con el modelo lineal (azul) y el no lineal (rojo)

Como era de esperarse, la temperatura máxima aumenta con el incremento de la velocidad inicial, para un mismo tiempo de frenada. Se puede estimar una ley de variación cuadrática haciendo un ajuste de los datos, tal como se muestra en la Figura anterior. En este caso, se decidió hacer variar la velocidad inicial, pero se podría variar también el tiempo de frenada. Se puede intuir lógicamente, que a menor tiempo de frenada, mayores serán las temperaturas alcanzadas.

Se observan resultados muy similares en ambos modelos, aunque se tienen temperaturas máximas ligeramente superiores en el modelo lineal, especialmente, a medida que aumenta la velocidad inicial. Por lo tanto, para el rango de temperaturas analizado, se puede linealizar el problema suponiendo propiedades constantes. sin mucho error. De esta forma, se reducirían enormemente los tiempos computacionales, porque ya no sería necesario que el programa itere en cada paso temporal.

Además, se recuerda de la Figura 4, que los datos de las propiedades físicas se extendían desde una temperatura mínima de 293 K hasta una máxima de 598 K. Por lo tanto, si bien se realizaron ajustes para interpolar las propiedades en temperaturas intermedias, como las temperaturas obtenidas en las simulaciones caen dentro del rango anterior, no se extrapolaron los datos. Es decir, los resultados obtenidos se pueden considerar fieles respecto de los datos de entrada de las propiedades físicas.

## 6. Análisis estructural: termoelasticidad

En base a los resultados obtenidos de las temperaturas, se pueden calcular las tensiones térmicas a las cuales se somete el disco. En el proceso de frenado, el disco experimenta tensiones debidas a la presión ejercida sobre el disco y la fuerza de fricción, además de tensiones debidas al gradiente de temperatura por el fenómeno de expansión térmica.

Las tensiones mecánicas que la pastilla/pinza de frenos ejerce sobre el disco son:

$$\underline{\underline{\sigma}}^{(M)} = {}^t\tau^{\theta z} {}^t\underline{g}_\theta {}^t\underline{g}_z + {}^t\tau^{z\theta} {}^t\underline{g}_z {}^t\underline{g}_\theta + {}^t\sigma^z {}^t\underline{g}_z {}^t\underline{g}_z$$

Con:

$${}^t\tau^{\theta z} = {}^t\tau^{z\theta} = {}^t f = \frac{{}^t q_T}{{}^t \omega} = -\frac{m R_r^2 \gamma + \frac{1}{2} \rho_a C_D A_R R_r^3 {}^t \omega^2}{8 A_f} \quad \forall {}^t \underline{x} \in S_q$$

Como se supuso el flujo de calor constante en la superficie  $S_q$ , se puede suponer también que las tensiones tangenciales son uniformes en dicha superficie. Para no involucrar la teoría de flexión de placas, se supondrá también que:  $\sigma_z = -p$ , siendo  $p$  la presión hidráulica ejercida sobre el disco.

Pasando a coordenadas cartesianas,

$$\underline{\underline{\sigma}}^{(M)} = \begin{bmatrix} 0 & 0 & -{}^t f r \sin({}^t \theta) \\ 0 & 0 & {}^t f r \cos({}^t \theta) \\ -{}^t f r \sin({}^t \theta) & {}^t f r \cos({}^t \theta) & -{}^t p \end{bmatrix} \underline{e}_i \underline{e}_j$$

Si el disco tuviera sus desplazamientos restringidos, las tensiones debidas al gradiente de temperatura se calcularían como:

$$\underline{\underline{\sigma}}^{(T)} = -\frac{E \alpha_E}{1 - 2\nu} ({}^t T - T_R) \delta_{ij} \underline{e}_i \underline{e}_j$$

Donde,  $E$  es el módulo de elasticidad,  $\alpha_E$  es el coeficiente de expansión térmica (no confundir con la difusividad térmica  $\alpha$ ),  $\nu$  es el coeficiente de Poisson,  $T_R$  es una temperatura de referencia del sólido no deformado,  $\delta_{ij}$  es la delta de Kronecker, y  $\underline{e}_i, \underline{e}_j$  son los vectores base en coordenadas cartesianas.

Notar que, si  $T > T_R$ , es decir, si se calienta el material, la pieza tiende a expandirse, por lo que uno pensaría que estaría sometida a tensiones de tracción (positivas), pero por el signo negativo que precede la expresión, las tensiones son de compresión. Esto se debe a que las tensiones térmicas sólo actúan cuando existe una restricción al movimiento natural del material. Por lo tanto, si éste se calienta y quiere expandirse, al restringir esa posibilidad (mediante diversas fuerzas de vínculo), se termina comprimiéndolo. Es importante notar que si se deja que el material se deforme libremente, no habrán tensiones de origen térmico.

El tensor de tensiones total es entonces:

$$\underline{\underline{\sigma}} = \begin{bmatrix} -\frac{E \alpha_E}{1 - 2\nu} ({}^t T - T_R) & 0 & -{}^t f r \sin({}^t \theta) \\ 0 & -\frac{E \alpha_E}{1 - 2\nu} ({}^t T - T_R) & {}^t f r \cos({}^t \theta) \\ -{}^t f r \sin({}^t \theta) & {}^t f r \cos({}^t \theta) & -\frac{E \alpha_E}{1 - 2\nu} ({}^t T - T_R) - {}^t p \end{bmatrix} \underline{e}_i \underline{e}_j$$

Con las componentes de este tensor, se puede calcular la tensión equivalente de Rankine para evaluar la resistencia de un material relativamente frágil como lo es la fundición gris.

$${}^t \sigma_{eq} = {}^t \sigma_I$$

Siendo  ${}^t \sigma_I$  la máxima tensión principal. Se comienza graficando las tensiones tangenciales desarrolladas por la fricción:



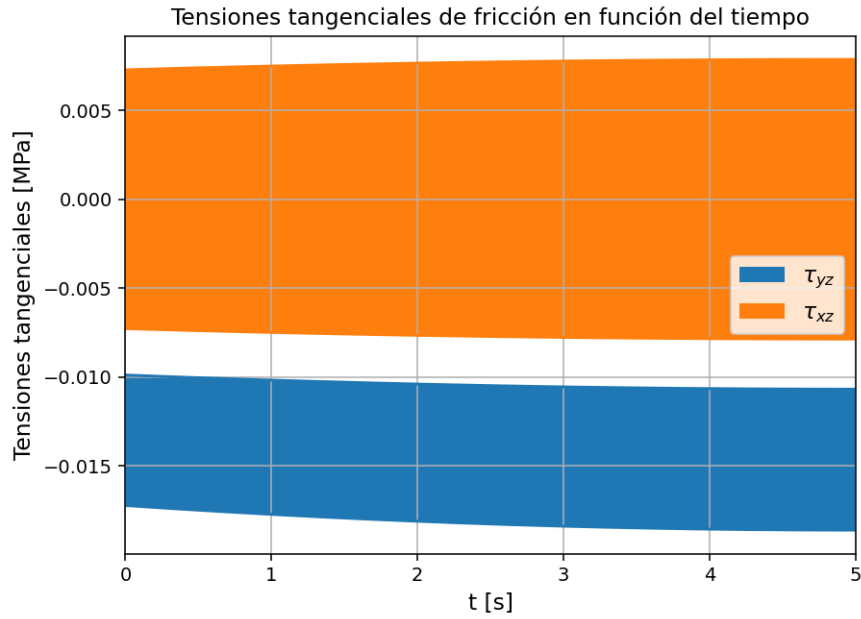


Figura 40: Tensiones tangenciales desarrolladas por la fricción. Se remarcen las posibles zonas de valores que pueden tomar las tensiones tangenciales en cada dirección.

Tal como se observa en la Figura 40, estas tensiones son muy bajas y no presentan riesgo alguno a la integridad del material. Por lo tanto, se las desprecia y se procede a estimar las tensiones térmicas. Además, se puede estimar la presión como:  ${}^t p = \frac{{}^t f}{\mu_d}$ , siendo  $\mu_d$  el coeficiente de fricción dinámico, que tiene un valor de alrededor de 0,4. Por lo tanto, la presión también se considera despreciable a los efectos estructurales del disco.

Entonces, el tensor de tensiones será el debido simplemente a las tensiones térmicas, y suponiendo una restricción a al menos uno de sus desplazamientos en coordenadas cartesianas, la máxima tensión principal será:

$${}^t \sigma_{eq} = -\frac{E \alpha_E}{1 - 2\nu} ({}^t T - T_R)$$

Se extraen los datos de las propiedades de la fundición gris, nuevamente, de la norma [ASME BPVC 2021 Section II part D \(metric\)](#) y se muestran las curvas aproximadas en la Figura 41. Se halló también la resistencia a la tracción, y el coeficiente de Poisson:

$$S_{UTS} = 300 \text{ MPa} \quad \nu = 0,29$$

Se puede estimar la resistencia a la compresión en:

$$S_{UCS} = -700 \text{ MPa}$$

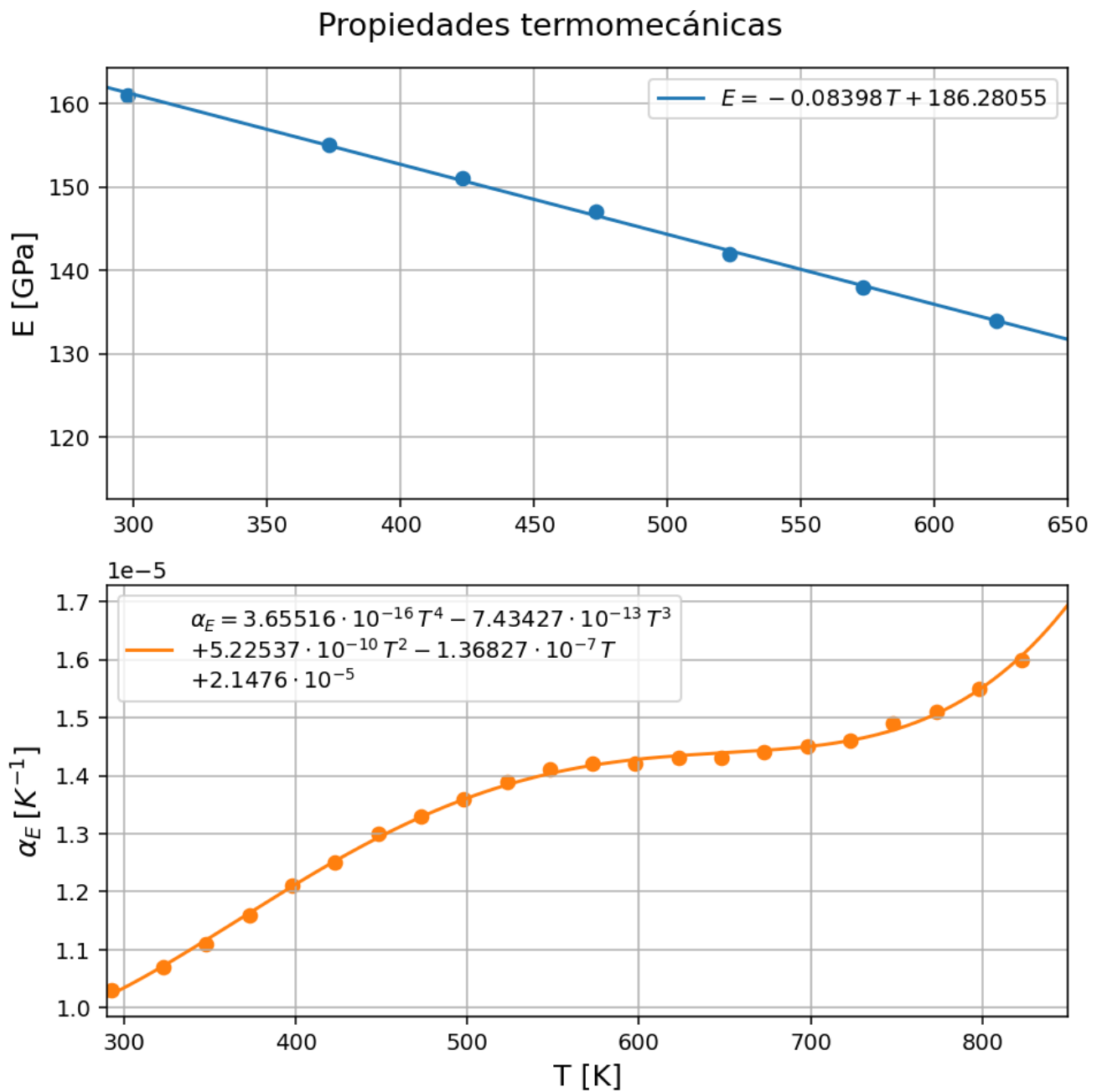


Figura 41: Módulo de elasticidad y coeficiente de expansión térmica en función de la temperatura

Entonces, teniendo las propiedades en función de la temperatura, se puede graficar las tensiones térmicas en función de ella. Tomando la temperatura inicial como referencia ( $T_R = T_0 = 298 \text{ K}$ ), se obtiene el gráfico de la Figura 42.

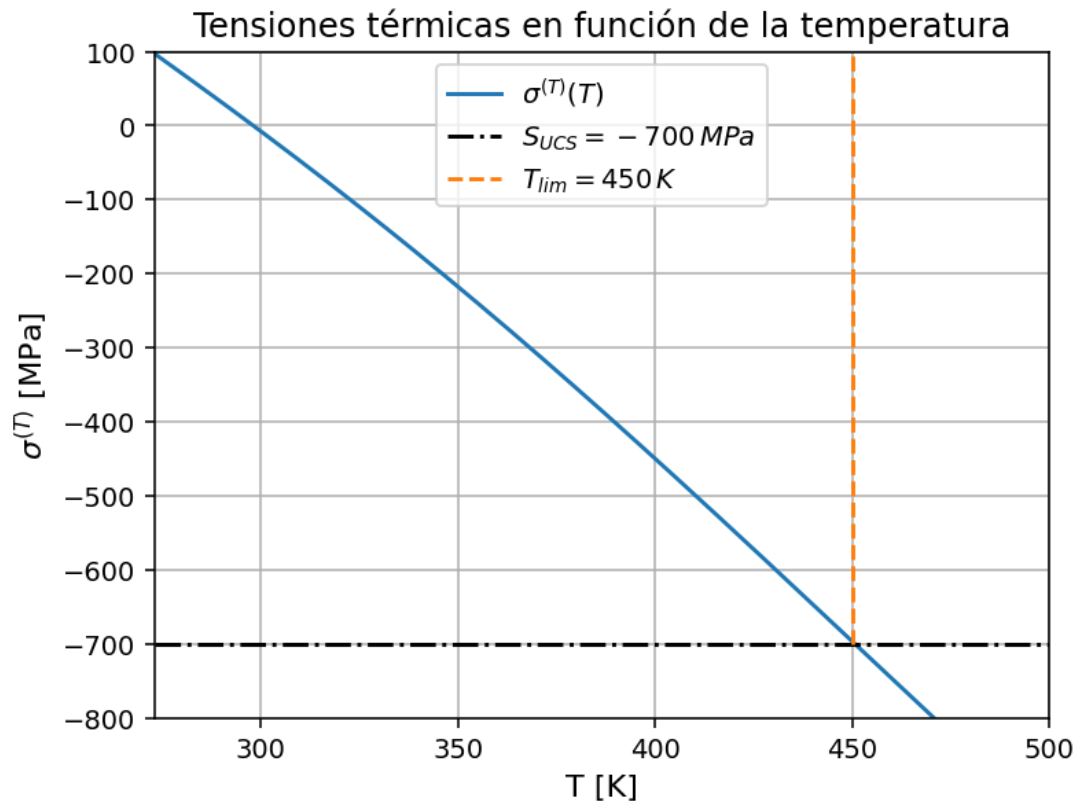


Figura 42: Tensiones térmicas en función de la temperatura. Se remarca la resistencia a la compresión  $S_{UCS}$  y la temperatura límite asociada  $T_{lim}$

En base a la tensión límite de rotura del material y de sus propiedades mecánicas en función de la temperatura, se puede establecer una temperatura límite a la cuál se puede someter el material, que es del orden de 450 K, tal como se muestra en la Figura anterior. Con una temperatura límite, se puede estimar, gracias al análisis térmico realizado previamente, una velocidad inicial límite (para un tiempo de frenada de 5 segundos).

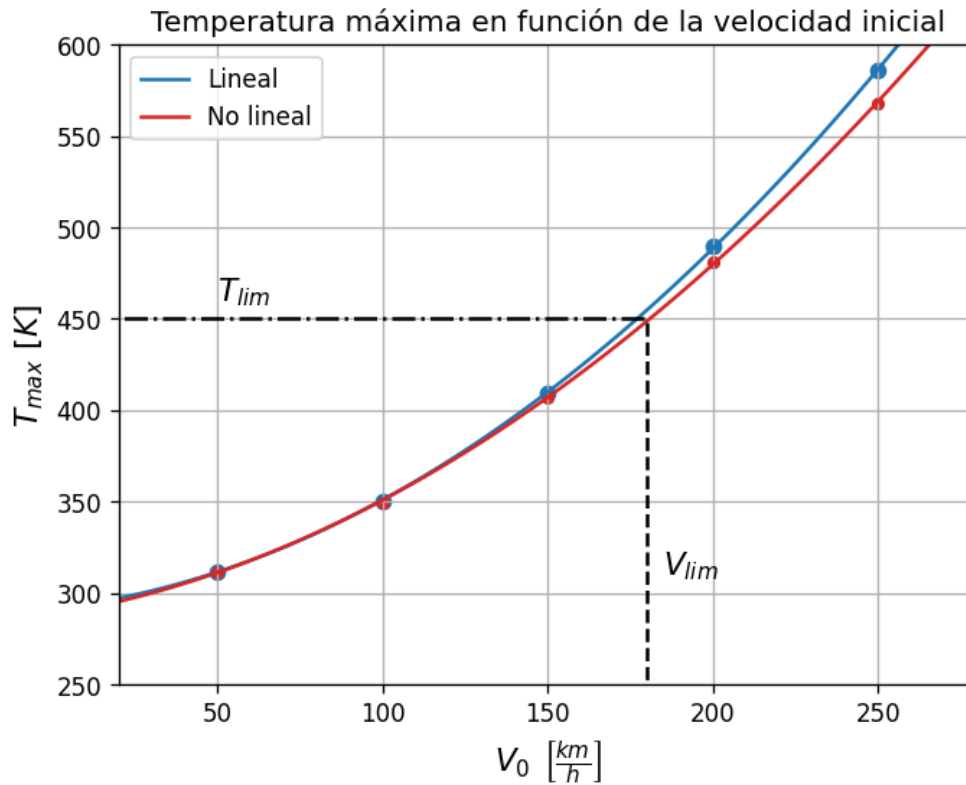


Figura 43: Temperatura máxima alcanzada por el disco de frenos en función de la velocidad inicial, para un tiempo de frenada de 5 segundos. Se remarca la temperatura límite para asegurar la resistencia a la compresión, y la velocidad inicial límite asociada.

De esta forma, se estima que para un tiempo de frenada de 5 segundos, la velocidad inicial límite que produce la falla del disco es  $V_{lim} \simeq 180 \frac{\text{km}}{\text{h}}$ , suponiendo que alguno de los desplazamientos  $u_x, u_y, u_z$  está restringido.

Por último, se puede hacer un análisis más detallado, contemplando distintas condiciones de borde para los desplazamientos y tensiones del disco. Tal como se explicó en la sección 5.3.1, se puede plantear el disco como un estado plano de tensiones con simetría de revolución, debido a su geometría cilíndrica y pequeño espesor. En ese caso, se puede modelar el problema mediante la siguiente ecuación diferencial en desplazamientos radiales  $u_r$ :

$$r^2 \frac{\partial^2 u_r(r, t)}{\partial r^2} + r \frac{\partial u_r(r, t)}{\partial r} - u_r(r, t) = \alpha_E(T)(1 + \nu) r^2 \frac{\partial T(r, t)}{\partial r} - \frac{1 - \nu^2}{E(T)} \rho(T) \sqrt{\omega^4(t) + \gamma^2} r^3$$

Donde se agregaron las fuerzas volumétricas inerciales provenientes de la rotación del disco, además de la carga térmica.

No se resolverá la ecuación en este informe, pero se la deja planteada para futuros trabajos.

## 7. Conclusión

En este trabajo, se modeló y se realizó el análisis térmico de un disco de frenos en un proceso de frenado para distintas velocidades iniciales. Se planteó el problema como un caso tridimensional, transitorio, no lineal (con propiedades dependientes de la temperatura), con una malla no estructurada de elementos tetragonales.

Se verificó el modelo con un problema conocido, gracias a su solución analítica, y se obtuvieron resultados para el problema del disco de frenos.

Finalmente, se hizo un breve análisis de termoelasticidad para verificar la resistencia estructural del disco y estimar una velocidad inicial límite para un tiempo de frenada fijo de 5 segundos.

Se determinó que el problema en este caso, puede ser simplificado en gran medida reduciendo dimensiones llevándolo a un caso unidimensional, y linealizando las propiedades del material. Pero es importante notar que toda la metodología utilizada para calcular las matrices en tres dimensiones con elementos tetragonales, o considerar la no linealidad material se puede trasladar a cualquier otro problema. Es decir, se puede aplicar todo el desarrollo anterior en un problema donde realmente sea necesario utilizar una malla tridimensional y/o un material con propiedades dependientes de su temperatura, por lo que, se destaca sobre todo, en este informe, la metodología utilizada y el planteo del caso más general posible.