



UNIVERSIDAD  
POLITÉCNICA  
DE MADRID

---

# ANÁLISIS TÉRMICO DE UNA PLACA DE CIRCUITO IMPRESO (PCB)

TRANSFERENCIA DE CALOR Y CONTROL TÉRMICO

---

*Autor:* Andrés Pedraza Rodríguez  
*Profesor:* Isidoro Martínez Herranz

MADRID, 26 DE MARZO, 2021



## **Resumen**

El objetivo del presente trabajo es determinar la distribución de temperaturas sobre una placa de circuito impreso poniendo especial énfasis en el cálculo de la temperatura máxima. A lo largo de las diferentes secciones se irá refinando el modelo de ingeniería hasta llegar a una distribución de temperatura bidimensional en la que se tendrá en cuenta también la radiación.



## Índice

<b>Índice de figuras</b>	<b>I</b>
<b>Índice de tablas</b>	<b>II</b>
<b>1. Introducción</b>	<b>1</b>
<b>2. Metodología</b>	<b>2</b>
<b>3. Resultados</b>	<b>3</b>
3.1. Primer modelo . . . . .	5
3.2. Segundo modelo . . . . .	8
3.3. Tercer modelo . . . . .	11
3.4. Cuarto modelo . . . . .	14
3.5. Quinto modelo . . . . .	16
<b>4. Conclusiones</b>	<b>22</b>
<b>A. Código empleado</b>	<b>24</b>

## Índice de figuras

1.	Representación gráfica del problema. . . . .	1
2.	Representación gráfica de los flujos de calor en el plano de la placa. . . . .	5
3.	Perfil de temperaturas para los casos de potencia puntual (azul) y potencia uniformemente distribuida (naranja) para el primer modelo de ingeniería. . . . .	7
4.	Representación gráfica de la placa en alzado. . . . .	8
5.	Perfil de temperaturas para los casos de $k_{IC}$ dada (azul) y $k_{IC}$ dada (naranja) para el segundo modelo de ingeniería en el cual se ha considerado la presencia de los circuitos integrados. . . . .	10
6.	Perfil de temperaturas para el tercer modelo de ingeniería en el cual se ha linealizado la radiación y se ha considerado disipación uniforme. . . . .	13
7.	Perfil de temperaturas para el cuarto modelo de ingeniería en el cual se ha considerado la radiación y la distribución de los IC correspondiente. . . . .	15
8.	Representación gráfica de la vista en planta de la placa de circuito impreso. . . . .	16
9.	Mapa de temperaturas para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional. . . . .	17
10.	Representación tridimensional de la distribución de temperaturas donde la altura representa la temperatura. . . . .	18
11.	Mapa de conductividades efectivas para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional. . . . .	19
12.	Mapa del producto de densidad por capacidad térmica específica para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional. . . . .	19
13.	Mapa de disipación volumétrica para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional. . . . .	20
14.	Perfil de temperaturas para los distintos modelos planteados. En azul el primer modelo con potencia puntual, en naranja el primer modelo con potencia distribuida uniformemente, en verde el segundo modelo con $k_{IC} \rightarrow \infty$ , en rojo el segundo modelo con la $k_{IC}$ dada, en morado el tercer modelo y en marrón el cuarto. . . . .	21
15.	Perfil de temperaturas para los distintos modelos planteados. En azul el primer modelo con potencia puntual, en naranja el primer modelo con potencia distribuida uniformemente, en verde el segundo modelo con $k_{IC} \rightarrow \infty$ , en rojo el segundo modelo con la $k_{IC}$ dada, en morado el tercer modelo y en marrón el cuarto. . . . .	23

## Índice de tablas

1.	Propiedades del FR4. . . . .	3
2.	Propiedades del cobre. . . . .	3
3.	Propiedades de los circuitos integrados (IC). . . . .	4
4.	Temperaturas máximas para los distintos modelos planteados para la placa. . . . .	21
5.	Temperaturas máximas para los distintos modelos planteados para la placa. . . . .	22

## 1. Introducción

El presente trabajo versa sobre el análisis térmico de una placa de circuito impreso (PCB) sobre la cual se instalan a su vez tres circuitos integrados (IC) y la cual va unida a una caja de electrónica por sus lados más estrechos. El análisis térmico de este tipo de componentes resulta determinante ya que son los fenómenos térmicos (sobrecalentamiento y degradación) los que comúnmente determinan la vida útil de esos dispositivos.

Así, resulta especialmente interesante determinar la temperatura máxima del circuito. Para ello en un primer lugar se parte de un modelo simplificado y poco a poco se va refinando para obtener resultados cada vez más precisos. Resulta interesante hacer notar que, en caso de que uno de estos modelos simplificados demuestre que la PBC es capaz de cumplir con los requisitos térmicos no será necesario seguir refinando el modelo (ahorrándose así mucho esfuerzo y horas de cálculo) ya que en modelos más simples se toman situaciones más adversas (*i.e.* con menor disipación y por tanto temperaturas más altas).

En la figura 1 se representa gráficamente el problema completo:

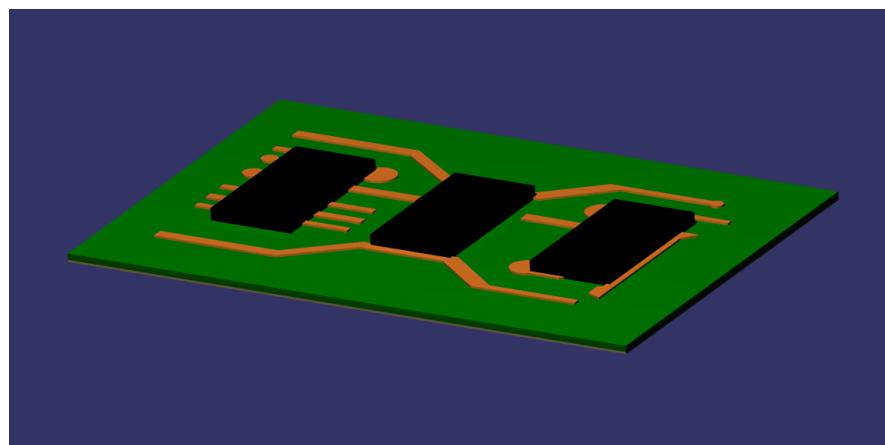


Figura 1: Representación de gráfica del problema.

## 2. Metodología

La realización de este trabajo se ha llevado a cabo mediante programación en *Python* haciendo uso de las librerías *numpy* para poder operar con matrices, *math* para incluir funciones matemáticas y *matplotlib* para la representación gráfica de las soluciones.

Mediante estas herramientas se ha analizado el problema de forma tanto analítica como numérica. En los modelos susceptibles de análisis analítico, una vez determinadas las ecuaciones se han usado las herramientas para completar los cálculos, obtener resultados y representarlos gráficamente. En lo que al análisis numérico se refiere se han implementado todos los modelos.

La evolución del modelo de ingeniería ha sido la siguiente:

- En un primer lugar, se considerará que toda la potencia se concentra en un punto y no se considerará la conductividad de los circuitos integrados. La PCB se analizará de forma unidimensional.
- Despues, se considerará que la potencia se encuentra distribuida uniformemente por la placa.
- A continuación, se considerará qué zona ocupan los circuitos integrados y se analizará el efecto de su conductividad, también de forma unidimensional.
- Luego, se volverá al modelo de potencia distribuida sin IC pero considerando la disipación radiativa (según su modelo lineal).
- De nuevo, se volverán a considerar los IC y se añadirá el efecto de la radiación (en este caso sin linealizar)
- Por último, se estudiará la placa bidimensionalmente, considerando qué área ocupa cada circuito integrado y también considerando la radiación.

A lo largo de la siguiente sección 3 se analizarán en más profundidad cada uno de ellos y se mostrarán los resultados obtenidos.

### 3. Resultados

Los elementos tratados en el desarrollo de este trabajo son:

**FR4** Es el núcleo de la placa de circuito impreso (PCB) y tiene propiedades ortotrópicas, es decir que presenta un comportamiento distinto en el plano y en la dirección normal a este. En la Tabla 1 se recogen las propiedades de este elemento.

Tabla 1: Propiedades del FR4.

Geometría	[mm]
$L_x$	140
$L_y$	100
$L_z$	1.5
Conductividad	[W/(m·K)]
$k_{xy}$	0.5
$k_z$	0.25
Densidad	[kg/m <sup>3</sup> ]
$\rho$	3000
Capacidad térmica	[J/K]
$c$	1850

**Cobre** El cobre (Cu) es el material que conforma los circuitos eléctricos externos a los circuitos integrados (IC). Al ser un metal y no estar forjado tiene propiedades isotrópicas. Sin embargo, en la parte superior de la PCB el cobre no se encuentra uniformemente distribuido en forma de lámina, en posteriores secciones se abordará este problema. En la Tabla 2 se recogen las propiedades de las zonas revestidas de este material.

Tabla 2: Propiedades del cobre.

Geometría	[mm]
$L_x$	140
$L_y$	100
$L_z$	0.05
Conductividad	[W/(m·K)]
$k$	395
$f$ (parte superior)	0.1
Densidad	[kg/m <sup>3</sup> ]
$\rho$	8260
Capacidad térmica	[J/K]
$c$	385

**Circuitos integrados** Los circuitos integrados (IC) son la razón de ser de la placa. Aunque están compuestos por diversos elementos en este problema se consideran isotrópicos y equiespaciados sobre la placa. En la Tabla 3 se recogen las propiedades de estos elementos.

Tabla 3: Propiedades de los circuitos integrados (IC).

Geometría	[mm]
$L_x$	20
$L_y$	40
$L_z$	3
<i>pitch</i>	20
Conductividad	[W/(m·K)]
$k$	50
Densidad	[kg/m <sup>3</sup> ]
$\rho$	416667
Capacidad térmica	[J/K]
$c$	20
Potencia	[W]
$\dot{W}$	5

Además los lados cortos de la PCB están en contacto con la pared de la caja de electrónica a una temperatura:

$$T = 298,15\text{K} . \quad (1)$$

De ahora en adelante se emplearán dichas variables con un subíndice que indique a qué elemento de la PCB hacen referencia y se reservará el término sin índice para hacer referencia a las dimensiones y propiedades totales del conjunto.

### 3.1. Primer modelo

Para una primera aproximación al problema se trata el conjunto como si los circuitos impresos no estuviesen y la placa fuese un solo material unidimensional. Para esto último se han de determinar las propiedades equivalentes tales que el flujo de calor total sea el mismo. Así, como se muestra en la Figura 2, el flujo de calor total ha de ser igual a la suma de calores por cada una de las partes:

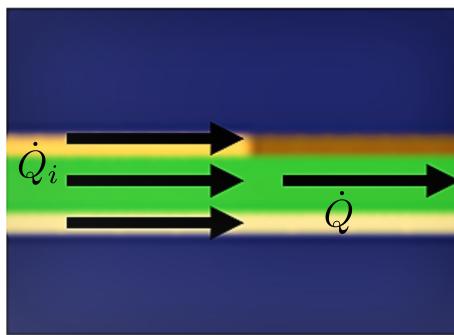


Figura 2: Representación gráfica de los flujos de calor en el plano de la placa.

$$\dot{Q} = \sum_i \dot{Q}_i . \quad (2)$$

Haciendo uso de la ecuación de Fourier:

$$\dot{Q} = kA \frac{\Delta T}{L} , \quad (3)$$

que para cada elemento es

$$\dot{Q}_i = K_i A_i f_i \frac{\Delta T}{L} . \quad (4)$$

De forma que mediante 2 se puede definir la conductividad efectiva como:

$$k = \frac{\sum K_i A_i f_i}{A} , \quad (5)$$

donde  $k$  es la conductividad efectiva,  $A = L_y L_z$  es el área efectiva de paso,  $k_i$  es la conductividad de cada material,  $A_i = L_y L_{z,i}$  es el área de paso de cada material y  $f_i$  es el factor de ocupación. La introducción de este último término está motivada por la disposición irregular de las pistas de cobre en la parte superior de la placa, en media se puede estimar que el volumen que ocupa el cobre es el 10% (en este caso) y que por tanto la conductividad y densidad del cobre de la parte superior es de una décima parte a la que correspondería al mismo espesor si no hubiera huecos. Existen otras formas

de considerar este fenómeno pero en este trabajo se ha optado por mantener la geometría y variar las propiedades físicas.

Si la ecuación de Fourier antes mencionada se combina con la del balance térmico:

$$mC \frac{dT}{dt} = \dot{W} + \dot{Q}, \quad (6)$$

(donde  $m$  es la masa y  $C$  la capacidad térmica) a lo largo de la placa, ya considerada como un mismo material de área efectiva  $A$  y conductividad  $k$ , se tiene la ecuación del calor

$$\rho c \frac{\partial T}{\partial t} = k \nabla^2 T + \phi, \quad (7)$$

donde  $\rho$  es la densidad,  $c$  es la capacidad térmica específica y  $\phi$  es el calor volumétrico que en este caso (dado que el único mecanismo de disipación es la conducción, se puede estimar que es la potencia total de los IC entre el volumen total de la placa).

Para el caso estacionario unidimensional, se tiene una ecuación diferencial ordinaria de segundo orden que una vez integrada conduce a una solución del tipo:

$$T(x) = a + b x - \frac{\phi x^2}{2k}, \quad (8)$$

donde los coeficientes  $a$  y  $b$  están determinados por las condiciones de contorno

$$\begin{cases} T(0) = a = T_{wall} \\ \frac{\partial T}{\partial x}(0) = b = \frac{\dot{Q}_{wall}}{kA} \end{cases}. \quad (9)$$

Térmicamente, el peor caso posible al que se puede enfrentar esta placa sería si estuviese toda la potencia concentrada en el centro por lo que la temperatura en este caso estaría determinada por:

$$T(x) = T_{wall} + \frac{\dot{Q}_{wall}}{kA} x, \quad (10)$$

donde  $\dot{Q}_{wall}$  es el calor que atraviesa la frontera en  $x = 0$  que para este caso, por la simetría del problema, es la mitad de la potencia total. De la expresión anterior (11) se puede dilucidar que el perfil de temperaturas es lineal alcanzándose el máximo en el centro de la placa  $T_{max} = 532$  K, lo cual es bastante alto para una placa de circuito impreso ya que sus temperaturas máximas rondan los 400 K (a esta temperatura el FR4 empieza a delaminarse) por lo que habrá de continuar el estudio de la placa.

Si por el contrario la disipación se considera uniformemente distribuida, el perfil de temperaturas tiene forma parabólica aunque su máximo sigue estando en el centro.

$$T(x) = T_{wall} + \frac{\dot{Q}_{wall}}{kA}x + \frac{\phi x^2}{2k} . \quad (11)$$

La temperatura máxima en este caso es  $T_{max} = 415$  K, lo cual relaja bastante la estimación anterior.

En la Figura 3 se muestran los resultados obtenidos para los dos casos anteriormente descritos.

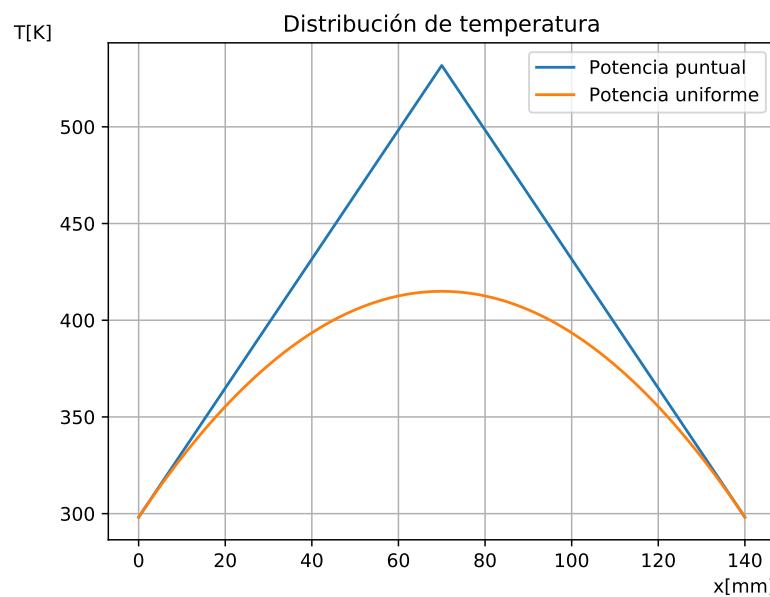


Figura 3: Perfil de temperaturas para los casos de potencia puntual (azul) y potencia uniformemente distribuida (naranja) para el primer modelo de ingeniería.

### 3.2. Segundo modelo

En este segundo modelo, se va a considerar además la presencia de los circuitos integrados sobre la placa de forma que su conductividad se añadirá a la del resto de elementos y la disipación de los IC se ubicará ahora bajo los mismos. En la Figura 4 se puede ver la distribución de componentes en el caso unidimensional:



Figura 4: Representación gráfica de la placa en alzado.

Es interesante destacar que, para una mayor sencillez de cálculo, se mantendrá el área efectiva de la sección anterior (véase 3.1) por lo que la placa puede ser tratada como si la geometría no cambiase dado que en la ecuación de Fourier 3 el producto  $k A$  se mantiene. De esta forma se designará como  $k_1$  a la conductividad efectiva de las zonas en las que no hay circuito integrado y se designará como  $k_2$  a la conductividad efectiva de las zonas en las que sí hay circuito integrado.

En este caso, dado que existe una discontinuidad de propiedades se pueden tomar dos alternativas. La primera es definir la conductividad y la disipación como una función a trozos o mediante la Función H de Heaviside y a partir de ahí integrar la ecuación del calor 7. La segunda alternativa es resolver el problema a trozos planteando la continuidad de temperaturas y de flujos de calor.

El problema se divide en dos y se distinguen cuatro zonas:

- Una primera zona entre el borde de la placa en contacto con la pared y el borde del primer circuito integrado.
- Una segunda zona que abarca el primer circuito integrado.
- Una tercera zona que abarca el espacio entre el primer IC y el IC central.
- Una cuarta zona que abarca la mitad del IC central.

Los coeficientes  $a$ ,  $b$  y  $c$  (donde  $c$  es el que multiplica a  $x^2$ ) de las distintas zonas son por tanto:

$$1 : \begin{cases} a_1 = T_{\text{wall}} \\ b_1 = \frac{Q_{\text{wall}}}{k_1 A} \\ c_1 = 0 \end{cases} \quad (12)$$

$$2 : \begin{cases} a_2 = a_1 + b_1 L_1 \\ b_2 = \frac{Q_{\text{wall}}}{k_2 A} \\ c_2 = \frac{-\phi}{2k_2} \end{cases} \quad (13)$$

$$3 : \begin{cases} a_3 = a_2 + b_2 L_2 + c_2 L_2^2 \\ b_3 = \frac{\dot{W}_{IC}/2}{k_1 A} \\ c_3 = 0 \end{cases} \quad (14)$$

$$4 : \begin{cases} a_4 = a_3 + b_3 L_3 \\ b_4 = \frac{\dot{W}_{IC}/2}{k_2 A} \\ c_4 = \frac{-\phi}{2k_2} \end{cases} \quad (15)$$

Es interesante estudiar cómo la conductividad de los circuitos integrados sobre la placa afecta a la distribución de temperaturas, es por ello que en la Figura 5 se muestra el perfil de temperaturas resultantes para la  $k_{IC}$  dada y para una  $k_{IC}$  dada.

Para el caso  $k_{IC} \rightarrow \infty$  se tiene una temperatura máxima de  $T_{max} = 387K$  y para la  $k_{IC}$  dada se tiene una temperatura máxima de  $T_{max} = 402K$ . Como se puede observar la presencia de circuitos integrados, al tener una conductividad mayor que la del FR4 y al aumentar el área de paso, favorece la disipación de calor por conducción. Además, como cabría de esperar, a medida que crece la conductividad las temperaturas entre los IC son menores. Sin embargo, aunque la temperatura es menor que en modelos anteriores, la placa sigue estando a una temperatura cercana al máximo por lo que en los siguientes modelos se tendrá en cuenta también la disipación por radiación.

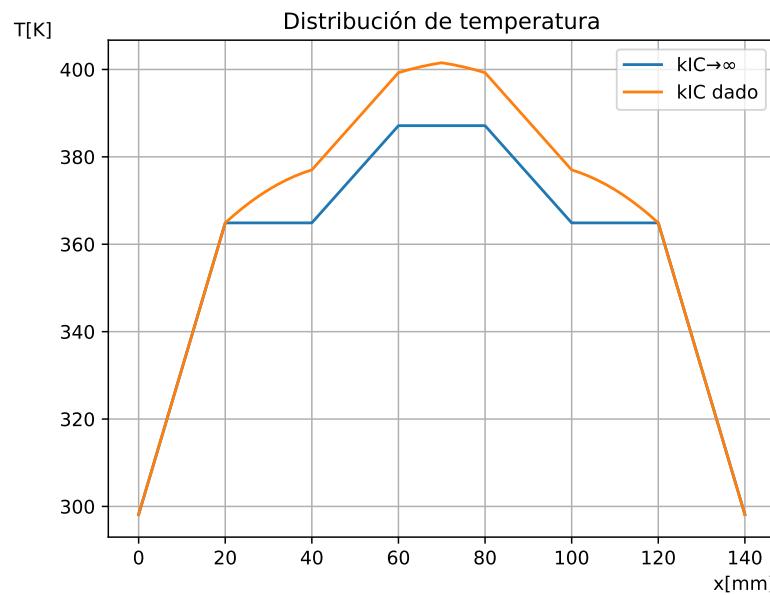


Figura 5: Perfil de temperaturas para los casos de  $k_{IC}$  dada (azul) y  $k_{IC}$  dada (naranja) para el segundo modelo de ingeniería en el cual se ha considerado la presencia de los circuitos integrados.

### 3.3. Tercer modelo

En el tercer modelo de la placa se vuelve a despreciar la presencia de circuitos integrados, se supone que la disipación está uniformemente distribuida y se linealiza la radiación haciendo uso del su desarrollo en serie de Taylor en torno a  $T_\infty$ :

$$\dot{Q}_{rad}(T) = \sum_{n=0}^{\infty} \frac{\dot{Q}_{rad}^{(n)}(T_\infty)}{n!} (T - T_\infty)^n \quad (16)$$

El flujo de calor por radiación entre el cuerpo a temperatura  $T$  y el entorno a temperatura  $T_\infty$  es:

$$\dot{Q}_{rad}(T) = \varepsilon \sigma_B F A_{rad} (T^4 - T_\infty^4) \quad (17)$$

donde  $\varepsilon$  es la emisividad,  $\sigma_B = 5,6710^{-8} \text{W/m}^2\text{K}^4$  es la constante de Stefan-Boltzmann,  $F$  es el factor de vista entre el cuerpo y el entorno (en este caso 1) y  $A_{rad}$  es el área de la superficie que radia. En este caso el entorno es una caja electrónica a  $T_\infty = 318,15 \text{ K}$  y la emisividad de la cara superior es de  $\varepsilon_1 = 0,7$  y la emisividad de la cara inferior es de  $\varepsilon_1 = 0,5$ . Linealizando esta ecuación en torno a  $T_\infty$  se llega a:

$$\dot{Q}_{rad}(T) = 4\varepsilon \sigma_B F A_{rad} T_\infty^3 (T - T_\infty) \quad (18)$$

de esta forma, se puede incluir en la ecuación del calor diferencial unidimensional, haciendo uso de  $dA_{rad} = pdx$ , para obtener:

$$\rho c A dx \frac{\partial T}{\partial t} = -k A \frac{\partial T}{\partial x} + k A \left( \frac{\partial T}{\partial x} + \frac{\partial^2 T}{\partial x^2} dx \right) - 4pdx\varepsilon \sigma_B T_\infty^3 (T - T_\infty) + \phi Adx, \quad (19)$$

que para el caso unidimensional y estacionario, tiene la ventaja de ser una ecuación lineal ordinaria de segundo orden:

$$0 = k A \frac{dT^2}{dx^2} - 4p\varepsilon \sigma_B T_\infty^3 (T - T_\infty) + \phi, \quad (20)$$

cuya solución es de la forma:

$$T(x) = c_1 e^{\eta x} + c_2 e^{-\eta x} + T_\xi \quad (21)$$

donde

$$\eta = \sqrt{\frac{4(L_y \varepsilon_1 + L_y \varepsilon_2) \sigma_B T_\infty^3}{kA}} \quad (22)$$

$$T_\xi = T_\infty + \frac{\phi A}{4(L_y \varepsilon_1 + L_y \varepsilon_2) \sigma_B T_\infty^3} \quad (23)$$

y cuyas condiciones de contorno son:

$$\begin{cases} T(0) = T_{wall} = c_1 + c_2 \\ T(L_x) = c_1 e^{\eta L_x} + c_2 e^{-\eta L_x} \end{cases} \quad (24)$$

por lo que los coeficientes de la solución general son:

$$\begin{cases} c_1 = (T_{wall} - T_\xi) \left( 1 - \frac{1 - e^{\eta L_x}}{e^{-\eta L_x} - e^{\eta L_x}} \right) \\ c_2 = (T_{wall} - T_\xi) \frac{1 - e^{\eta L_x}}{e^{-\eta L_x} - e^{\eta L_x}} \end{cases} \quad (25)$$

De nuevo, la temperatura máxima se ubica en el centro de la placa y es de  $T_{max} = 373$  K lo cual significa que la placa está ya por debajo de la temperatura de delaminación de FR4 común, sin embargo habrá que considerar un modelo más complejo ya que la temperatura sigue siendo muy alta y los circuitos integrados funcionan mejor con una menor temperatura. En la figura 6 se puede ver el perfil de temperaturas el cual tiene de nuevo una forma parabólica (debido a la conducción) pero tiene una menor altura (debido a la radiación).

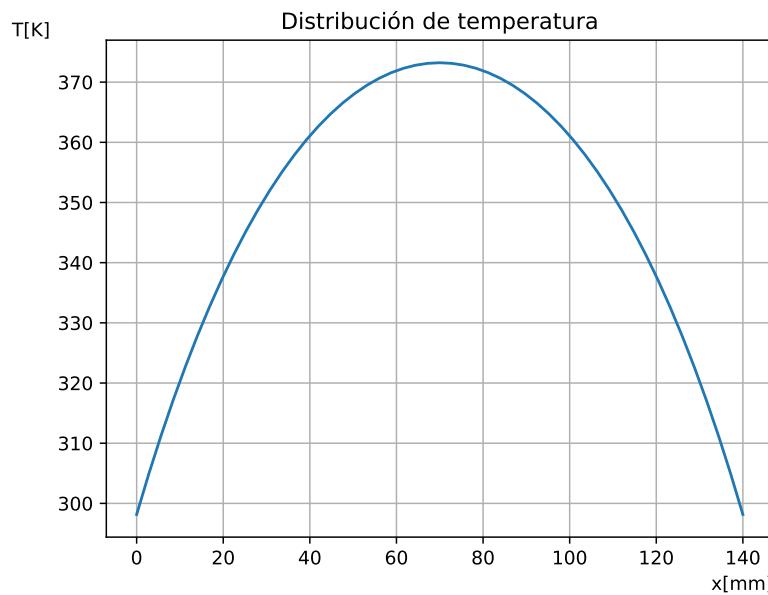


Figura 6: Perfil de temperaturas para el tercer modelo de ingeniería en el cual se ha linealizado la radiación y se ha considerado disipación uniforme.

### 3.4. Cuarto modelo

En este cuarto modelo, se ha considerado la presencia de los circuitos impresos por lo que de nuevo se vuelve a una distribución por zonas como en la sección 3.2 y además se incluye la radiación como en 3.3 pero esta vez sin linealizar. Esto último supone trabajar con una ecuación diferencial de segundo orden no lineal y además por tramos. Si bien existen métodos analíticos para este tipo de soluciones resulta mucho más conveniente realizar una discretización del modelo para así poder emplear las herramientas de cálculo numérico. Dado que la ecuación del calor estacionaria es elíptica resulta mucho más sencillo estudiar el caso no estacionario mediante diferencias finitas ya que se trata de una ecuación parabólica. Si se aplica un esquema de integración tipo Euler explícito se tiene que:

$$T_i^{k+1} = T_i^k + \frac{\Delta t}{(\rho c)_i A} \left[ k_{i+} A \frac{T_{i+1}^k - T_i^k}{\Delta x^2} - k_{i-} A \frac{T_i^k - T_{i-1}^k}{\Delta x^2} + \phi_i A - (\varepsilon_1 + \varepsilon_2) L_y \sigma_B (T_i^{k/4} - T_\infty^4) \right] \quad (26)$$

$i \in [1, N-1],$   
 $k \in [1, M],$

donde  $T_i^k$  es la temperatura del nodo  $i$  en el tiempo  $k$ ;  $\Delta t$  es el intervalo temporal;  $(\rho c)_i$  es el producto de densidad y capacidad térmica específica del nodo  $i$ ;  $A$  es el área efectiva de paso de la placa y es constante para todo el dominio;  $k_{i+}$  es la conductividad térmica efectiva (consistente con la  $A$  efectiva considerada) entre el nodo  $i$  y  $i+1$ ;  $\Delta x$  es el intervalo espacial (que en este caso es constante porque la distribución es equiespaciada) y  $\phi_i$  es la disipación volumétrica del nodo  $i$ . El resto de variables están descritas en posteriores secciones.

A este esquema hay que añadir también las condiciones de contorno que en este caso son:

$$\begin{cases} T_0^k = T_{wall} \\ T_N^k = T_{wall} \end{cases}, \quad (27)$$

y la distribución inicial de temperatura

$$T_i^0 = T_{wall}. \quad (28)$$

Una vez hecho esto tan solo queda determinar el tiempo para el cual la solución habrá llegado a un estado estacionario con el fin de no alargar en exceso la simulación. Existen varios métodos de implementar esto, como establecer un criterio de parada o calcular el tiempo característico del problema. En este último método basta con realizar un estudio de los órdenes de magnitud del problema:

$$t_c \sim \frac{\rho c \Delta T}{k \Delta T / L_c^2 - \frac{1}{\delta_c} \varepsilon \sigma_B T_{ini}^4 + \phi}, \quad (29)$$

que en una primera aproximación supone un tiempo de unos 2500 s así que se ha de elegir un tiempo de simulación de ese orden de magnitud. Además el método numérico se ha validado modificando el término radiativo y la distribución de conductividades con los modelos anteriormente estudiados.

El perfil de temperaturas obtenido gracias a este método se plasma en la Figura 7 y la temperatura máxima es de  $T_{max} = 366\text{K}$  lo cual aleja aún más la placa de la zona de peligro, sin embargo, dado que los circuitos integrados no llegan hasta los bordes largos, será necesario un quinto y último modelo que considere unos IC más pequeños (y por tanto con una disipación más densa) de forma bidimensional.

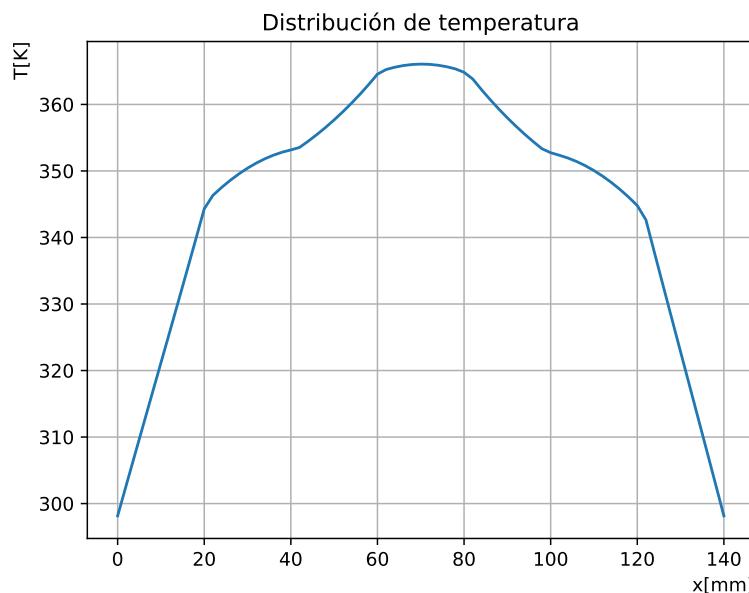


Figura 7: Perfil de temperaturas para el cuarto modelo de ingeniería en el cual se ha considerado la radiación y la distribución de los IC correspondiente.

### 3.5. Quinto modelo

En el quinto y último modelo estudiado trata el problema de forma bidimensional en el plano, así los circuitos integrados no llegan hasta los bordes y la distribución en planta de los mismos se muestra en la Figura 8:

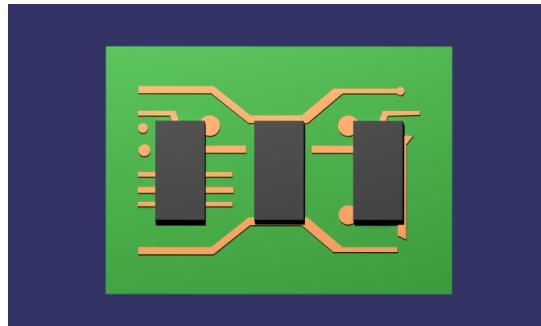


Figura 8: Representación gráfica de la vista en planta de la placa de circuito impreso.

De nuevo, se recurre al cálculo numérico para la determinación del perfil de temperaturas. Se elige también un esquema Euler explícito y la expresión que toma ahora es:

$$T_{i,j}^{k+1} = T_{i,j}^k + \frac{\Delta t}{(\rho c)_i A} \left[ F_k(k, A, T) + \phi_{i,j} A - (\varepsilon_1 + \varepsilon_2) L_y \sigma_B (T_{i,j}^{k,4} - T_\infty^4) \right], \quad (30)$$

donde

$$F_k(k, A, T) = k_{i+,j} A \frac{T_{i+1,j}^k - T_{i,j}^k}{\Delta x^2} - k_{i-,j} A \frac{T_{i,j}^k - T_{i-1,j}^k}{\Delta x^2} + k_{i,j+} A \frac{T_{i,j+1}^k - T_{i,j}^k}{\Delta x^2} - k_{i,j-1} A \frac{T_{i,j}^k - T_{i,j-1}^k}{\Delta x^2},$$

con

$$i \in [1, N_x - 1],$$

$$j \in [1, N_y - 1],$$

$$k \in [1, M],$$

y con las condiciones de contorno de: temperatura de la pared constante a lo largo de los bordes  $x = 0$  y  $x = L_x$ ; y de borde adiabático en  $y = 0$  e  $y = L_y$  definidas por:

$$\begin{cases} T_{0,j}^k = T_{wall} \\ T_{N_x,j}^k = T_{wall} \\ T_{i,0}^k = T_{i,1}^k \\ T_{i,N_y}^k = T_{i,N_y-1}^k \end{cases}, \quad (31)$$

y la distribución inicial de temperatura

$$T_{i,j}^0 = T_{wall}. \quad (32)$$

El tiempo de simulación en este caso es del mismo orden que el calculado en la sección 3.4. La simulación en este caso proporciona un temperatura mayor como cabía de esperar al estar más concentrada la disipación térmica. La temperatura máxima alcanzada es de  $T_{max} = 389$  K lo que acerca de nuevo la placa a la zona de peligro. El mapa de temperaturas se representa en la Figura 9. Además en la Figura 11, 12 y 13.

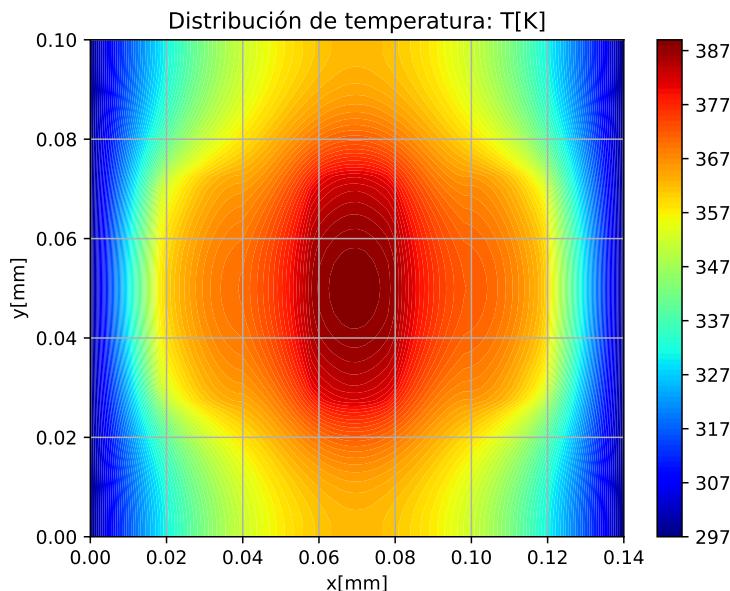


Figura 9: Mapa de temperaturas para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional.

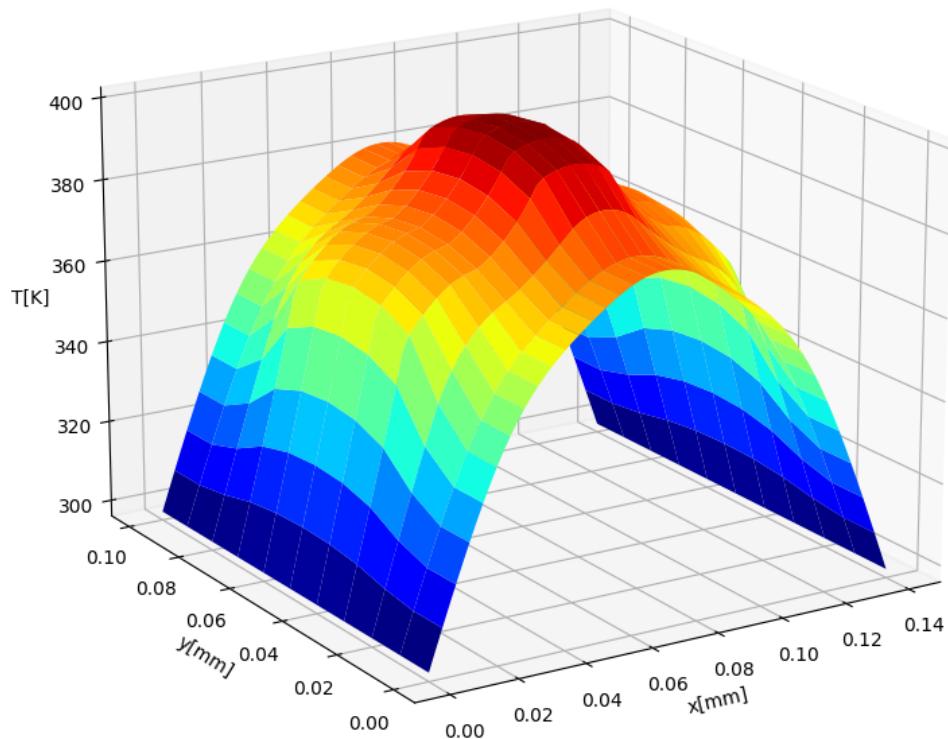


Figura 10: Representación tridimensional de la distribución de temperaturas donde la altura representa la temperatura.

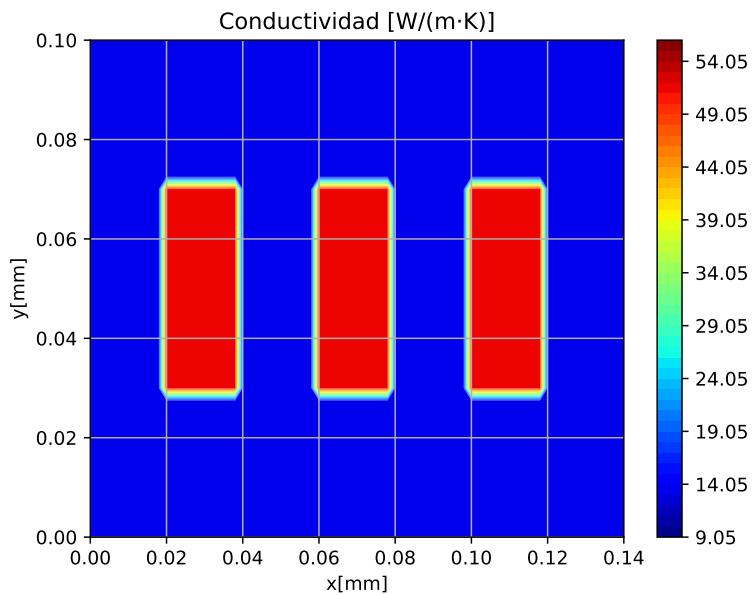


Figura 11: Mapa de conductividades efectivas para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional.

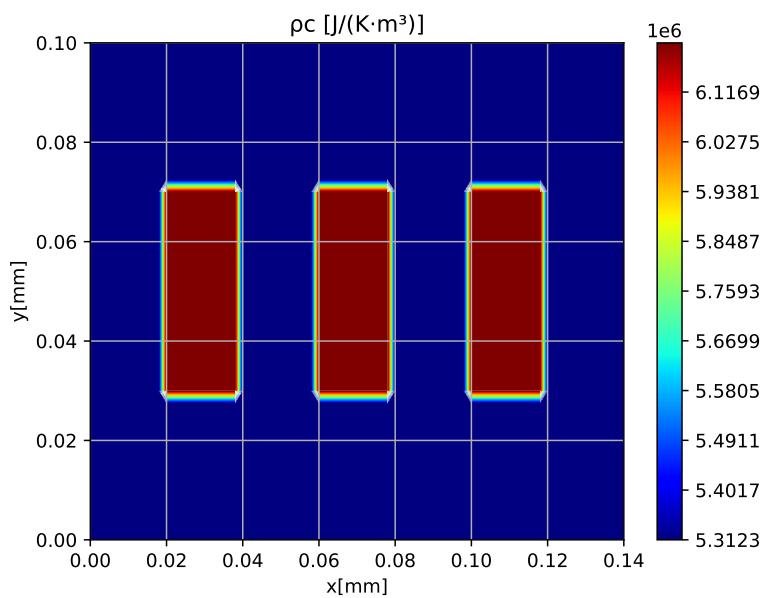


Figura 12: Mapa del producto de densidad por capacidad térmica específica para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional.

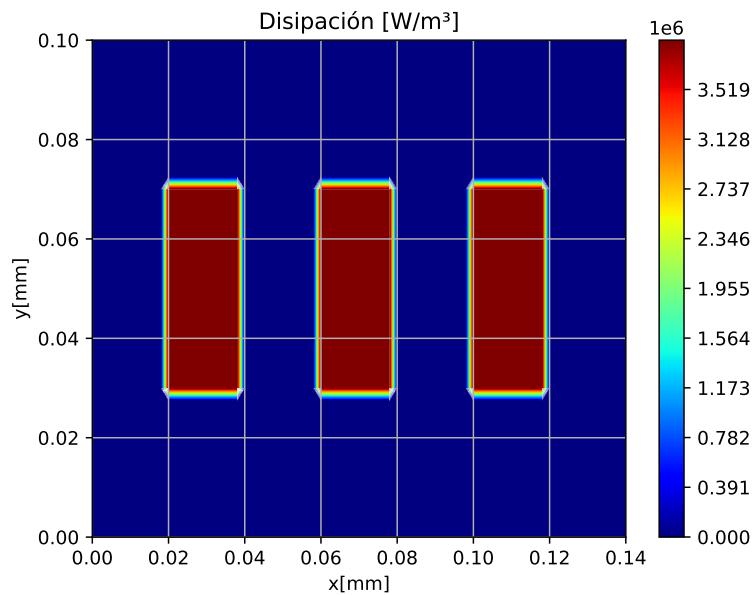


Figura 13: Mapa de dissipación volumétrica para el quinto modelo de ingeniería en el cual se ha considerado tanto la radiación como la distribución de los IC bidimensional.

Las temperaturas máximas obtenidas muestran que la placa trabajará en torno a los 380 K por lo que habrá de diseñarse para poder trabajar a esa temperatura o se habrán de incluir elementos refrigerativos como pueden ser radiadores o *heat pipes*. Los valores obtenidos en las distintas subsecciones se recogen en la Tabla 4 y se recogen los perfiles de temperatura de los distintos modelos estudiados en la Figura 14.

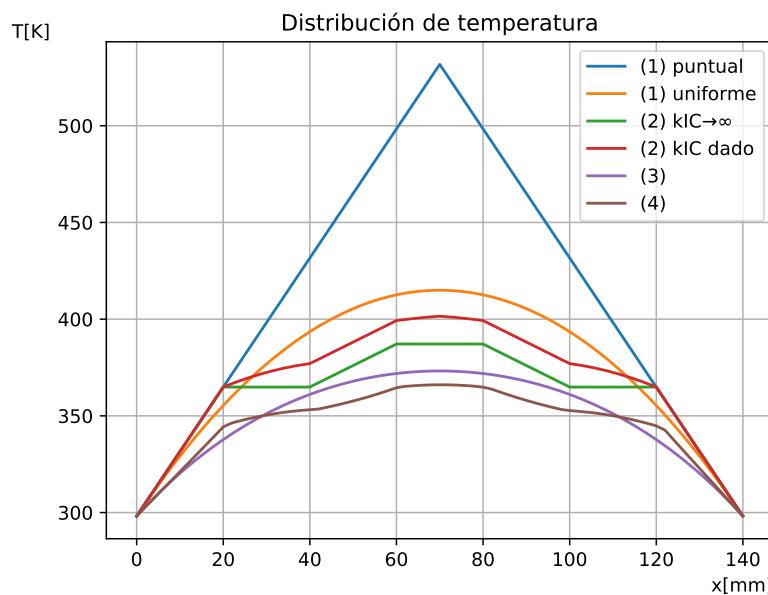


Figura 14: Perfil de temperaturas para los distintos modelos planteados. En azul el primer modelo con potencia puntual, en naranja el primer modelo con potencia distribuida uniformemente, en verde el segundo modelo con  $k_{IC} \rightarrow \infty$ , en rojo el segundo modelo con la  $k_{IC}$  dada, en morado el tercero modelo y en marrón el cuarto.

Tabla 4: Temperaturas máximas para los distintos modelos planteados para la placa.

	$T_{\max}[K]$
Primer modelo (potencia puntual)	532
Primer modelo (potencia distribuida)	415
Segundo modelo ( $k_{IC} \rightarrow \infty$ )	387
Segundo modelo ( $k_{IC}$ dada)	402
Tercer modelo	373
Cuarto modelo	366
Quinto modelo	389

## 4. Conclusiones

A la vista del desarrollo de este trabajo se han extraído las siguientes conclusiones:

- Un modelo sencillo puede ser una muy buena primera aproximación a un problema complejo ya que permite estimar valores característicos y hacer estimaciones que a posteriori reducen mucho el tiempo de cálculo de modelos más complejos.
- Las temperaturas máximas obtenidas se recogen en la Tabla 5 y muestran la evolución de los distintos modelos estudiados.

Tabla 5: Temperaturas máximas para los distintos modelos planteados para la placa.

	$T_{\max}[K]$
Primer modelo (potencia puntual)	532
Primer modelo (potencia distribuida)	415
Segundo modelo ( $k_{IC} \rightarrow \infty$ )	387
Segundo modelo ( $k_{IC}$ dada)	402
Tercer modelo	373
Cuarto modelo	366
Quinto modelo	389

- Los perfiles de temperaturas de los casos unidimensionales que se muestran en la Figura 15 confirman la primera conclusión e ilustran el proceso llevado a cabo.

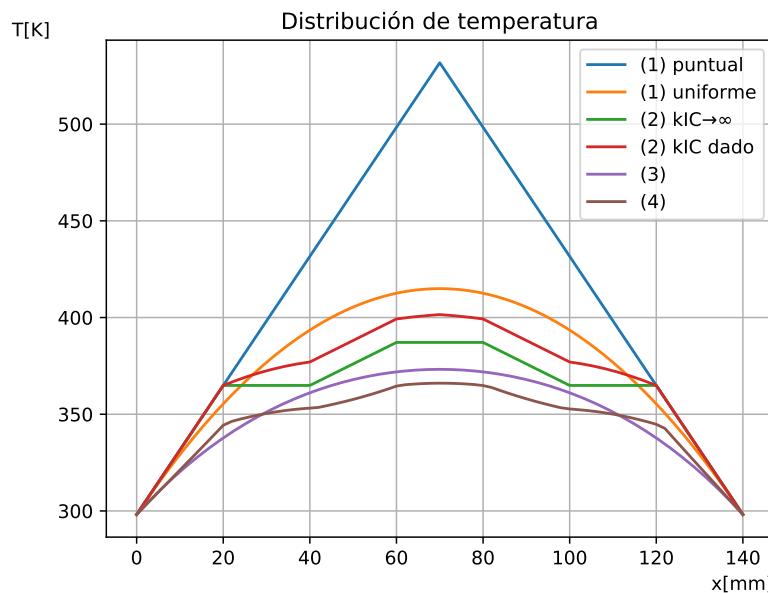


Figura 15: Perfil de temperaturas para los distintos modelos planteados. En azul el primer modelo con potencia puntual, en naranja el primer modelo con potencia distribuida uniformemente, en verde el segundo modelo con  $k_{IC} \rightarrow \infty$ , en rojo el segundo modelo con la  $k_{IC}$  dada, en morado el tercer modelo y en marrón el cuarto.

- También ha de destacarse que en ocasiones las simplificaciones realizadas puede llevar a error si por ser demasiado sencillas, es por ello que al considerar la distribución bidimensional de temperaturas la temperatura estimada es mayor.

Quedaría pendiente para un análisis más preciso la simulación tridimensional del problema puesto que también existe una componente transversal del flujo de calor y también podrían considerarse otros elementos disipativos como radiadores o *heat pipes* integrados en el conjunto de la placa

## A. Código empleado

El presente código está escrito en Python y hace uso de las librerías *numpy* y *math* de distribución libre. Además, para un uso más sencillo, se puede seleccionar el apartado por medio de un archivo *data*.

```
1 import os
2 import numpy as np
3 from math import exp
4 import copy
5
6 # Para guardar los resultados en un csv
7 results_dir = './Results/' # Ruta
8 if not os.path.exists(results_dir):
9     os.makedirs(results_dir)
10 def save_result(name,result):
11     np.savetxt(results_dir+str(name)+'.csv', result, delimiter=',', fmt='%s')
12
13 # Para saber qué apartado correr
14 data = 'data' #Nombre del archivo
15 try:
16     with open(data, 'r') as data:
17         for line in data:
18             if 'apartado_a' in line:
19                 p = line.split()
20                 apartado_a = p[2]
21             if 'apartado_b' in line:
22                 p = line.split()
23                 apartado_b = p[2]
24             if 'apartado_c' in line:
25                 p = line.split()
26                 apartado_c = p[2]
27             if 'apartado_d' in line:
28                 p = line.split()
29                 apartado_d = p[2]
30             if 'apartado_e' in line:
31                 p = line.split()
32                 apartado_e = p[2]
33             if 'numerico' in line:
```

```
34             p = line.split()
35             numerico = p[2]
36             if 'disp' in line:
37                 p = line.split()
38                 disp = float(p[2])
39         except:
40             ### Apartados
41             apartado_a = 'yes'
42             apartado_b = 'yes'
43             apartado_c = 'yes'
44             apartado_d = 'yes'
45             apartado_e = 'yes'
46             ###
47             numerico = 'yes'
48             ###
49             disp = 100
50
51     print('No data configuration file found')
52
53
54 # Clases
55 class elemento(object):
56
57     def __init__(self):
58         # Atributos geométricos
59         self.Lx = ''
60         self.Ly = ''
61         self.Lz = ''
62
63         # Atributos térmicos
64         self.k = ''
65         self.kxy = ''
66         self.kx = ''
67         self.ky = ''
68         self.kz = ''
69
70         self.rho_c = ''
```

```

72     self.W = ''
73
74 def made_of(self,objects):
75
76     self.Lx = max(c.Lx for c in objects)
77     self.Ly = max(c.Ly for c in objects)
78     self.Lz = sum(c.Lz for c in objects)
79
80     self.V = self.Lx * self.Ly * self.Lz
81
82     self.Ax = self.Ly*self.Lz #Área efectiva de paso en la dirección x
83
84     self.kx = sum((c.kx*c.Ly*c.Lz) for c in objects)/self.Ax
85     self.rho_c = sum((c.rho_c*c.Ly*c.Lz*c.Lx) for c in objects)/self.V
86
87 def add(self,objects):
88
89     self.kx2 = sum((c.kx*c.Ly*c.Lz) for c in objects)/self.Ax + self.kx
90     self.rho_c2 = sum((c.rho_c*c.Ly*c.Lz*c.Lx) for c in objects)/self.V + self.
91             rho_c
92
93 #### ENUNCIADO ####
94 ## Considérese una tarjeta electrónica (PCB) de 140 1001 ,5 mm3 de FR-4, con un
95     recubrimiento de 50 m de
96     ## cobre por cada lado, que en una de las caras es continuo, y en la otra sólo ocupa
97         el 10% de la superficie, en la
98     ## cual van montados tres circuitos integrados (IC), cada uno de 40 203 mm3
99     ## , disipando 5 W, con kIC=50 W/(m K)
100    ## de conductividad térmica, CIC=20 J/K de capacidad térmica, y distribuidos
101        uniformemente en la PCB (20 mm de
102    ## separación entre ellos). Se supondrá que los lados cortos de la PCB tienen contacto
103        térmico perfecto con paredes
104    ## permanentemente a 25 C , y que los otros dos bordes están térmicamente aislados. T
105        ómese para el FR-4 k=0,5
106    ## W/(m K) en el plano y la mitad a su través. Se pide:
107
108 # FR4
109 FR4 = elemento()

```

```
104 FR4.Lx = 140e-3 #m
105 FR4.Ly = 100e-3 #m
106 FR4.Lz = 1.5e-3 #m
107 FR4.kx = 0.5 #W/( m K )
108 FR4.ky = 0.5 #W/( m K )
109 FR4.kz = 0.25 #W/( m K )
110 FR4.rho_c = 1850 * 3000 # J/( K m^3)
111
112
113 # Cu
114 Cu = elemento()
115 Cu.Lx = FR4.Lx #m
116 Cu.Ly = FR4.Ly #m
117 Cu.Lz = 50e-6 #m
118 Cu.F = 0.1
119 Cu.kx = 395.0 #W/( m K )
120 Cu.ky = 395.0 #W/( m K )
121 Cu.kz = 395.0 #W/( m K )
122 Cu.rho_c = 385 * 8260 # J/( K m^3)
123
124 Cu_up = copy.deepcopy(Cu)
125 Cu_up.kx = 395.0 * 0.1 #W/( m K )
126 Cu_up.ky = 395.0 * 0.1 #W/( m K )
127 Cu_up.kz = 395.0 * 0.1 #W/( m K )
128 Cu_up.rho_c = Cu.rho_c * 0.1 # J/( K m^3)
129
130 # IC
131 IC = elemento()
132 IC.Lx = 20e-3 #m
133 IC.Ly = 40e-3 #m
134 IC.Lz = 3e-3 #m
135 IC.W = 5 #W
136 IC.kx = 50 #W/( m K )
137 IC.ky = 50 #W/( m K )
138 IC.kz = 50 #W/( m K )
139 IC.rho_c = 20 * 1/(IC.Lx*IC.Ly*IC.Lz) #J/( K m^3)
140 IC.pitch = 20e-3 #m
141
```

```
142 # PCB
143 PCB = elemento()
144 PCB.made_of([FR4,Cu,Cu_up]) #Los IC van aparte porque a veces no están
145 PCB.add([IC])
146
147 # Paredes
148 T_wall = 25+273.15 #K
149
150 if apartado_a == 'yes':
151     print('*** a ***')
152
153     ## a) Considerando que la tarjeta sólo evacua calor por los bordes, determinar la
154     # temperatura máxima que se
155     ## alcanzaría si toda la disipación estuviese uniformemente repartida en la PCB y
156     # los IC no influyeran.
157
158     # Debido a la simetría del problema se puede separar el problema en dos
159     W_dis = 3*IC.W
160     Q_wall = W_dis/2
161     phi = W_dis/(PCB.V)
162
163     A_eff = PCB.Ax
164     k_eff = PCB.kx
165     L = PCB.Lx/2
166
167     ## Solución analítica
168     print('Solución analítica')
169
170     # Primera aproximación, toda la potencia concentrada en el centro: T(x) = a + b*x
171     a1 = T_wall
172     b1 = Q_wall/(k_eff * A_eff)
173
174     T_max = a1 + b1* L
175
176     print(' Potencia puntual: T_max = ',round(T_max), 'K ó ',round(T_max-273.15), 'C')
```

```
177 a2 = T_wall
178 b2 = Q_wall/(k_eff * A_eff)
179 c2 = -phi /(2*k_eff)
180
181 T_max = a2 + b2*L + c2*L**2
182
183 print(' Potencia uniforme: T_max = ',round(T_max), 'K ó ',round(T_max-273.15), 'C')
184
185 # Discretización de la solución
186
187 N = 50
188 xa = np.linspace(0,L,N+1)
189 Ta1 = a1 + b1*xa
190 Ta2 = a2 + b2*xa + c2*xa**2
191 xa = np.concatenate((xa,xa+L))
192 Ta1 = np.concatenate((Ta1,np.flip(Ta1)))
193 Ta2 = np.concatenate((Ta2,np.flip(Ta2)))
194
195 # Guardar resultados
196 save_result('xa',xa)
197 save_result('Ta1',Ta1)
198 save_result('Ta2',Ta2)
199
200
201 ## Solución numérica
202 print('Solución numérica')
203
204 if numerico == 'yes':
205     # Discretización
206
207     L = 2*L      # Espacio de simulación
208     T = 5000    # Tiempo de simulación
209     N = 70      # Número de elementos espaciales
210     M = int(1e5) # Número de elementos temporales (ver criterio)
211     Dx = L/N
212     Dt = T/M
213     xan = np.linspace(0,L,N+1)
214     ta = np.linspace(0,T,M+1)
```

```

215
216     rho_c = PCB.rho_c
217     p = 0
218     h = 0
219
220     # Estabilidad
221     a=k_eff/(rho_c)           #Diffusivity [m^2/s]
222     Fo=a*Dt/(Dx*Dx)         #Fourier's number
223     Bi=h*p*Dx/(k_eff*A_eff/Dx) #Biot's number
224
225     if (1-Fo*(2+Bi)) < 0:
226         print('This is unstable increase number of time steps')
227
228     # Potencia puntual
229
230     T = T_wall * np.ones((M+1,N+1)) # T(t,x) iniciales
231
232     for t in range(0,len(ta)-1):
233         if t%disp == 0: print('Tiempo de simulación:',ta[t], 's \Temperatura má
234                                         xima:',npamax(T[t,:]),'K')
235
236         # Condiciones de contorno
237         T[t,0]=T_wall
238         T[t,N]=T_wall
239
240         for x in range(1,len(xan)-1):
241             # Potencia puntual
242             if x == int(N/2): phii = W_dis/(Dx*PCB.Ly*PCB.Lz)
243             else: phii=0
244             # Euler explícito
245             T[t+1,x] = T[t,x] + Dt/(rho_c*A_eff)*(k_eff*A_eff*(T[t,x+1]-T[t,x])/Dx
246                                         **2-k_eff*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phii*A_eff)
247
248             Ta1n = np.zeros((len(xan)))
249             Ta1n[:]=T[-1,:]
250             T_max = max(Ta1n)

```

```
250     print(' Potencia puntual: T_max = ',round(T_max),'K ó',round(T_max-273.15),'C')
251
252 # Potencia uniforme
253
254 T = T_wall * np.ones((M+1,N+1)) # T(t,x) inicial
255
256 for t in range(0,len(ta)-1):
257     if t%disp == 0: print('Tiempo de simulación:',ta[t], 's \Temperatura má
258         xima:',npamax(T[t,:]),'K')
259
260 # Condiciones de contorno
261 T[t,0]=T_wall
262 T[t,N]=T_wall
263
264 for x in range(1,len(xan)-1):
265     # Euler explícito
266     T[t+1,x] = T[t,x] + Dt/(rho_c*A_eff)*(k_eff*A_eff*(T[t,x+1]-T[t,x])/Dx
267         **2-k_eff*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phi*A_eff)
268
269 Ta2n = np.zeros((len(xan)))
270 Ta2n[:]=T[-1,:]
271 T_max = max(Ta2n)
272
273 print(' Potencia uniforme: T_max = ',round(T_max),'K ó',round(T_max-273.15),'C
274         ')
275
276 # Guardar resultados
277 save_result('xan',xan)
278 save_result('Ta1n',Ta1n)
279 save_result('Ta2n',Ta2n)
280
281 if apartado_b == 'yes':
282     print('*** b ***')
```

```
283     ## b) Considerando que la tarjeta sólo evacua calor por los bordes, determinar la
284     ## temperatura máxima que se
285     ## alcanzaría con un modelo unidimensional en el que los IC llegaran hasta los
286     ## bordes aislados, en el límite
287     ## k I C , y con la kIC dada.
288
289     ### Para los dos problemas
290
291
292     W_dis = 3*IC.W
293     Q_wall = W_dis/2
294
295     A_eff = PCB.Ax
296     V_eff = A_eff * IC.Lx
297     phi = IC.W/V_eff
298     L = PCB.Lx/2
299
300
301     # Con ejes en el borde de cada una de las 4 zonas
302     L1 = L - IC.Lx - IC.pitch - IC.Lx/2
303     L2 = IC.Lx
304     L3 = IC.pitch
305     L4 = IC.Lx/2
306
307     ## Solución analítica
308     print('Solución analítica')
309
310     N = 50
311     x1 = np.linspace(0,L1,N+1)
312     x2 = np.linspace(0,L2,N+1)
313     x3 = np.linspace(0,L3,N+1)
314     x4 = np.linspace(0,L4,N+1)
315
316     xb = np.concatenate((x1,x2+L1,x3+L1+L2,x4+L1+L2+L3))
317
318     ## Con k I C
```

```
319 # Zona 1: T(x) = a1 + b1*x
320 k_eff = k_eff1
321 a1 = T_wall
322 b1 = Q_wall/(k_eff * A_eff)
323 T1 = a1 + b1*x1
324 # Zona 2: T(x) = a2 + b2*x + c2*x^2 donde c2 = phi /(2k)
325 a2 = a1 + b1*L1
326 b2 = 0
327 c2 = 0
328 T2 = a2 + b2*x2 + c2*x2**2
329 # Zona 3: T(x) = a3 + b3*x
330 k_eff = k_eff1
331 a3 = a2 + b2*L2 + c2*L2**2
332 b3 = (IC.W/2)/(k_eff * A_eff)
333 T3 = a3 + b3*x3
334 # Zona 4: T(x) = a4 + b4*x + c4*x^2 donde c4 = -phi /(2k)
335 a4 = a3 + b3*L3
336 b4 = 0
337 c4 = 0
338 T4 = a4 + b4*x4 + c4*x4**2
339
340 Tb1 = np.concatenate((T1, T2, T3, T4))
341 T_max = max(Tb1)
342
343 print(' Con k I C : T_max = ', round(T_max), 'K ó', round(T_max-273.15), 'C')
344
345 ## Con kIC dada
346
347 k_eff1 = PCB.kx
348 k_eff2 = PCB.kx2
349
350 # Zona 1: T(x) = a1 + b1*x
351 k_eff = k_eff1
352 a1 = T_wall
353 b1 = Q_wall/(k_eff * A_eff)
354 T1 = a1 + b1*x1
355 # Zona 2: T(x) = a2 + b2*x + c2*x^2 donde c2 = -phi /(2k)
356 k_eff = k_eff2
```

```
357 a2 = a1 + b1*L1
358 b2 = Q_wall/(k_eff * A_eff)
359 c2 = - phi /(2*k_eff)
360 T2 = a2 + b2*x2 + c2*x2**2
361 # Zona 3: T(x) = a3 + b3*x
362 k_eff = k_eff1
363 a3 = a2 + b2*L2 + c2*L2**2
364 b3 = (IC.W/2)/(k_eff * A_eff)
365 T3 = a3 + b3*x3
366 # Zona 4: T(x) = a4 + b4*x + c4*x^2 donde c4 = -phi /(2k)
367 k_eff = k_eff2
368 a4 = a3 + b3*L3
369 b4 = (IC.W/2)/(k_eff * A_eff)
370 c4 = - phi/2 /(2*k_eff)
371 T4 = a4 + b4*x4 + c4*x4**2
372
373 Tb2 = np.concatenate((T1, T2, T3, T4))
374 T_max = max(Tb2)
375
376 print(' Con la kIC dada: T_max = ', round(T_max), 'K ó', round(T_max-273.15), 'C')
377
378 # Doblar la solución
379
380 xb = np.concatenate((xb,np.flip(2*L-xb)))
381 Tb1 = np.concatenate((Tb1,np.flip(Tb1)))
382 Tb2 = np.concatenate((Tb2,np.flip(Tb2)))
383
384 # Guardar resultados
385 save_result('xb',xb)
386 save_result('Tb1',Tb1)
387 save_result('Tb2',Tb2)
388
389 ## Solución numérica
390 print('Solución numérica')
391
392 # Discretización
393
394 if numerico == 'yes':
```

```
395
396     L1 = PCB.Lx/2 - IC.Lx- IC.pitch- IC.Lx/2
397     L2 = IC.Lx + L1
398     L3 = IC.pitch + L2
399     L4 = IC.Lx/2 + L3
400     L5 = IC.Lx/2 + L4
401     L6 = IC.pitch + L5
402     L7 = IC.Lx + L6
403     L8 = PCB.Lx
404
405     L = PCB.Lx      # Espacio de simulación
406     T = 6000        # Tiempo de simulación
407     N = 70          # Número de elementos espaciales
408     M = int(1e5)    # Número de elementos temporales (ver criterio)
409     Dx = L/N
410     Dt = T/M
411     xbn = np.linspace(0,L,N+1)
412     tb  = np.linspace(0,T,M+1)
413
414     p = 0
415     h = 0
416
417     # Estabilidad
418     a=PCB.kx/(PCB.rho_c)           #Diffusivity [m^2/s]
419     Fo=a*Dt/(Dx*Dx)               #Fourier's number
420     Bi=h*p*Dx/(k_eff*A_eff/Dx)   #Biot's number
421
422     if (1-Fo*(2+Bi)) < 0:
423         print('This is unstable increase number of time steps')
424
425     # Propiedades
426     def k_fun(pos):
427         x = xbn[pos]
428         k1 = PCB.kx
429         k2 = 1e2
430         from numpy import heaviside as H
431         val = k1 + (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)-H(x-L7,dis))*(k2-k1)
```

```
432         return val
433
434     def rho_c_fun(pos):
435         x = xbn[pos]
436         rho_c1 = PCB.rho_c
437         rho_c2 = PCB.rho_c2
438         from numpy import heaviside as H
439         val = rho_c1 + (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)
440             -H(x-L7,dis))*(rho_c2-rho_c1)
441         return val
442
443     def phii_fun(pos):
444         x = xbn[pos]
445         from numpy import heaviside as H
446         val = (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)-H(x-L7,
447             dis))*(phi)
448         return val
449
450     k=np.zeros((len(xbn)))
451     rho_c=np.zeros((len(xbn)))
452     phii=np.zeros((len(xbn)))
453     for x in range(0,len(xbn)):
454         k[x] = k_fun(x)
455         rho_c[x] = rho_c_fun(x)
456         phii[x] = phii_fun(x)
457
458     ## Con k I C
459
460     T = T_wall * np.ones((M+1,N+1)) # T(t,x) inicial
461     for t in range(0,len(tb)-1):
462         if t%disp == 0: print('Tiempo de simulación:',tb[t], 's \Temperatura má
463             xima:',npamax(T[t,:]),'K')
464
465         # Condiciones de contorno
466         T[t,0]=T_wall
467         T[t,N]=T_wall
468
469         for x in range(1,len(xbn)-1):
```

```

467      # Propiedades
468      kp = (k[x+1]+k[x])/2
469      kn = (k[x]+k[x-1])/2
470      # Euler explícito
471      T[t+1,x] = T[t,x] + Dt/(rho_c[x]*A_eff)*(kp*A_eff*(T[t,x+1]-T[t,x])/Dx
472          **2-kn*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phii[x]*A_eff)
473
474      Tb1n = np.zeros((len(xbn)))
475      Tb1n[:]=T[-1,:]
476      T_max = max(Tb1n)
477
478      print('Con k I C : T_max = ',round(T_max), 'K ó ',round(T_max-273.15), 'C')
479
480      ## Con kIC dada
481
482      def k_fun(pos):
483          x = xbn[pos]
484          k1 = PCB.kx
485          k2 = PCB.kx2
486          from numpy import heaviside as H
487          val = k1 + (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)-H(
488              x-L7,dis))*(k2-k1)
489          return val
490
491      k=np.zeros((len(xbn)))
492      rho_c=np.zeros((len(xbn)))
493      phii=np.zeros((len(xbn)))
494      for x in range(0,len(xbn)):
495          k[x] = k_fun(x)
496          rho_c[x] = rho_c_fun(x)
497          phii[x] = phii_fun(x)
498
499      T = T_wall * np.ones((M+1,N+1)) # T(t,x) inicial
500      for t in range(0,len(tb)-1):
501          if t%disp == 0: print('Tiempo de simulación:',tb[t], 's \Temperatura má
502          xima:',npamax(T[t,:]), 'K')
```

501

```

502         # Condiciones de contorno
503         T[t,0]=T_wall
504         T[t,N]=T_wall
505
506         for x in range(1,len(xbn)-1):
507             # Propiedades
508             kp = (k[x+1]+k[x])/2
509             kn = (k[x]+k[x-1])/2
510             # Euler explícito
511             T[t+1,x] = T[t,x] + Dt/(rho_c[x]*A_eff)*(kp*A_eff*(T[t,x+1]-T[t,x])/Dx
512                           **2-kn*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phii[x]*A_eff)
513
514             Tb2n = np.zeros((len(xbn)))
515             Tb2n[:]=T[-1,:]
516             T_max = max(Tb2n)
517
518             print('Con la KIC dada: T_max = ',round(T_max,'K ó',round(T_max-273.15),'C'))
519
520             # Guardar resultados
521             save_result('xbn',xbn)
522             save_result('Tb1n',Tb1n)
523             save_result('Tb2n',Tb2n)
524
525 if apartado_c == 'yes':
526     print('*** c ***')
527     ## c) Considerando que se transmite calor por radiación, con una emisividad media
528     ## de 0,7 por el lado de los
529     ## componentes, y de 0,5 por la cara opuesta, con una caja electrónica que se
530     ## puede suponer negra y a 45 C ,
531     ## determinar la temperatura máxima linealizando las pérdidas radiativas y con
532     ## dissipación uniforme.
533
534     eps1 = 0.7
535     p1 = PCB.Ly
536     eps2 = 0.5
537     p2 = PCB.Ly
538     T_inf = 45+273.15 #K

```

```
536     sigma = 5.67e-8 #W/m^2 K ^4
537     T_media = T_inf
538
539     phi = 3*IC.W/PCB.V
540     A_eff = PCB.Ax
541     k_eff = PCB.kx
542     L = PCB.Lx
543
544     ## Solución analítica
545     print('Solución analítica')
546
547     # T(x) = c1 * exp(a**0.5*x) + c2 * exp(-a**0.5*x) + T_chi
548
549     a = 4*(p1*eps1+p2*eps2)*sigma*T_media**3 / (k_eff*A_eff)
550     eta = a**0.5
551     T_chi = T_inf + phi*A_eff/(4*(p1*eps1+p2*eps2)*sigma*T_media**3)
552     T_gorro0 = T_wall - T_chi
553
554     c2 = T_gorro0 * (1-exp(eta*L))/(exp(-eta*L)-exp(eta*L))
555     c1 = T_gorro0 - c2
556
557     # Discretización de la solución
558
559     N = 50
560     xc = np.linspace(0,L,N+1)
561     Tc = c1 * np.exp(eta*xc) + c2 * np.exp(-eta*xc) + T_chi
562
563     T_max = c1 * np.exp(eta*L/2) + c2 * np.exp(-eta*L/2) + T_chi
564
565     print(' T_max = ',round(T_max,'K'),round(T_max-273.15),'C')
566
567     # Guardar resultados
568     save_result('xc',xc)
569     save_result('Tc',Tc)
570
571     ## Solución numérica
572     print('Solución numérica')
573
```

```
574 # Discretización
575
576 if numerico == 'yes':
577
578     L1 = PCB.Lx/2 - IC.Lx- IC.pitch- IC.Lx/2
579     L2 = IC.Lx + L1
580     L3 = IC.pitch + L2
581     L4 = IC.Lx/2 + L3
582     L5 = IC.Lx/2 + L4
583     L6 = IC.pitch + L5
584     L7 = IC.Lx + L6
585     L8 = PCB.Lx
586
587     L = PCB.Lx      # Espacio de simulación
588     T = 5000        # Tiempo de simulación
589     N = 70          # Número de elementos espaciales
590     M = int(1e5)    # Número de elementos temporales (ver criterio)
591     Dx = L/N
592     Dt = T/M
593     xcn = np.linspace(0,L,N+1)
594     tc  = np.linspace(0,T,M+1)
595
596     p = 0
597     h = 0
598
599 # Estabilidad
600 a=PCB.kx/(PCB.rho_c)           #Diffusivity [m^2/s]
601 Fo=a*Dt/(Dx*Dx)                #Fourier's number
602 Bi=h*p*Dx/(k_eff*A_eff/Dx)    #Biot's number
603
604 if (1-Fo*(2+Bi)) < 0:
605     print('This is unstable increase number of time steps')
606
607 # Propiedades
608 k_eff = PCB.kx
609 rho_c = PCB.rho_c
610 phi = 3*IC.W/PCB.V
611
```

```
612     T = T_wall * np.ones((M+1,N+1)) # T(t,x) inicial
613     for t in range(0,len(tc)-1):
614         if t%disp == 0: print('Tiempo de simulación:',tc[t], 's \Temperatura má
615                         xima:',npamax(T[t,:]),'K')
616
617         # Condiciones de contorno
618         T[t,0]=T_wall
619         T[t,N]=T_wall
620
621         for x in range(1,len(xcn)-1):
622             # Euler explícito
623             T[t+1,x] = T[t,x] + Dt/(rho_c*A_eff)*(k_eff*A_eff*(T[t,x+1]-T[t,x])/Dx
624                           **2-k_eff*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phi*A_eff - 4*(eps1*p1+
625                           eps2*p2)*sigma*T_media**3*(T[t,x] - T_inf))
626
627             Tcn = np.zeros((len(xcn)))
628             Tcn[:]=T[-1,:]
629             T_max = max(Tcn)
630
631             print(' T_max = ',round(T_max),'K ó ',round(T_max-273.15),'C')
632
633             # Guardar resultados
634             save_result('xcn',xcn)
635             save_result('Tcn',Tcn)
636
637             if apartado_d == 'yes':
638                 print('*** d ***')
639                 ## d) Resolver el caso anterior pero sin linealizar y con la disipación no
640                 ## uniforme.
641
642                 eps1 = 0.7
643                 p1 = PCB.Ly
644                 eps2 = 0.5
645                 p2 = PCB.Ly
646                 T_inf = 45+273.15 #K
647                 sigma = 5.67e-8 #W/m^2 K ^4
```

```
646     phi = IC.W/(IC.Lx*PCB.Ly*PCB.Lz)
647     A_eff = PCB.Ax
648     k_eff = PCB.kx
649
650     ## Solución numérica
651     print('Solución numérica')
652
653     # Discretización
654
655     if numerico == 'yes':
656
657         L1 = PCB.Lx/2 - IC.Lx- IC.pitch- IC.Lx/2
658         L2 = IC.Lx + L1
659         L3 = IC.pitch + L2
660         L4 = IC.Lx/2 + L3
661         L5 = IC.Lx/2 + L4
662         L6 = IC.pitch + L5
663         L7 = IC.Lx + L6
664         L8 = PCB.Lx
665
666         L = PCB.Lx      # Espacio de simulación
667         T = 5000        # Tiempo de simulación
668         N = 70          # Número de elementos espaciales
669         M = int(1e5)    # Número de elementos temporales (ver criterio)
670         Dx = L/N
671         Dt = T/M
672         xdn = np.linspace(0,L,N+1)
673         td  = np.linspace(0,T,M+1)
674
675         p = 0
676         h = 0
677
678         # Estabilidad
679         a=PCB.kx/(PCB.rho_c)           #Diffusivity [m^2/s]
680         Fo=a*Dt/(Dx*Dx)              #Fourier's number
681         Bi=h*p*Dx/(k_eff*A_eff/Dx)   #Biot's number
682
683         if (1-Fo*(2+Bi)) < 0:
```

```
684         print('This is unstable increase number of time steps')
685
686     # Propiedades
687     def k_fun(pos):
688         x = xdn[pos]
689         k1 = PCB.kx
690         k2 = PCB.kx2
691         dis = 1
692         from numpy import heaviside as H
693         val = k1 + (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)-H(
694             x-L7,dis))*(k2-k1)
695         return val
696
697     def rho_c_fun(pos):
698         x = xdn[pos]
699         rho_c1 = PCB.rho_c
700         rho_c2 = PCB.rho_c2
701         dis = 1
702         from numpy import heaviside as H
703         val = rho_c1 + (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis
704             )-H(x-L7,dis))*(rho_c2-rho_c1)
705         return val
706
707     def phii_fun(pos):
708         x = xdn[pos]
709         dis = 1
710         from numpy import heaviside as H
711         val = (H(x-L1,dis)-H(x-L2,dis)+H(x-L3,dis)-H(x-L5,dis)+H(x-L6,dis)-H(x-L7,
712             dis))*(phi)
713         return val
714
715     k=np.zeros((len(xdn)))
716     rho_c=np.zeros((len(xdn)))
717     phii=np.zeros((len(xdn)))
718     for x in range(0,len(xdn)):
719         k[x] = k_fun(x)
720         rho_c[x] = rho_c_fun(x)
721         phii[x] = phii_fun(x)
```

```
719
720     T = T_wall * np.ones((M+1,N+1)) # T(t,x) inicial
721
722     for t in range(0,len(td)-1):
723         if t%disp == 0: print('Tiempo de simulación:',td[t], 's \Temperatura má
724             xima:',npamax(T[t,:]),'K')
725
726         # Condiciones de contorno
727         T[t,0]=T_wall
728         T[t,N]=T_wall
729
730         for x in range(1,len(xdn)-1):
731             # Propiedades
732             kp = (k[x+1]+k[x])/2
733             kn = (k[x]+k[x-1])/2
734             # Euler explícito
735             T[t+1,x] = T[t,x] + Dt/(rho_c[x]*A_eff)*(kp*A_eff*(T[t,x+1]-T[t,x])/Dx
736                             **2-kn*A_eff*(T[t,x]-T[t,x-1])/Dx**2 + phii[x]*A_eff - (eps1*p1+
737                             eps2*p2)*sigma*(T[t,x]**4 - T_inf**4))
738
739             Tdn = np.zeros((len(xdn)))
740             Tdn[:]=T[-1,:]
741             T_max = max(Tdn)
742
743             print(' T_max = ',round(T_max), 'K ó ',round(T_max-273.15), 'C')
744
745             # Guardar resultados
746             save_result('xdn',xdn)
747             save_result('Tdn',Tdn)
748
749             if apartado_e == 'yes':
750                 print('*** e ***')
751                 ## e) Resolver el problema térmico bidimensional estacionario y comparar el perfil
752                 ## central de temperaturas con
753                 ## el del caso anterior
754
755             epsl = 0.7
```

```
753     eps2 = 0.5
754     T_inf = 45+273.15 #K
755     sigma = 5.67e-8 #W/m^2 K ^4
756
757     z_eff = PCB.Lz
758     phi = IC.W/(IC.Lx*IC.Ly*z_eff)
759
760
761     ## Solución numérica
762     print('Solución numérica')
763
764     # Discretización
765
766     if numerico == 'yes':
767
768         L1x = round(PCB.Lx/2 - IC.Lx- IC.pitch- IC.Lx/2,8)
769         L2x = IC.Lx + L1x
770         L3x = IC.pitch + L2x
771         L4x = IC.Lx/2 + L3x
772         L5x = IC.Lx/2 + L4x
773         L6x = IC.pitch + L5x
774         L7x = IC.Lx + L6x
775         L8x = PCB.Lx
776
777         L1y = round((PCB.Ly - IC.Ly)/2,8)
778         L2y = IC.Ly + L1y
779         L3y = PCB.Ly
780
781         Lx = PCB.Lx      # Espacio de simulación
782         Ly = PCB.Ly      # Espacio de simulación
783         T = 3250          # Tiempo de simulación
784         Nx = 70           # Número de elementos espaciales
785         Ny = 40           # Número de elementos espaciales
786         M = int(1e5)       # Número de elementos temporales (ver criterio)
787         Dx = Lx/Nx
788         Dy = Ly/Ny
789         Dt = T/M
790         xen = np.linspace(0,Lx,Nx+1)
```

```
791     yen = np.linspace(0,Ly,Ny+1)
792     te  = np.linspace(0,T,M+1)
793
794     p = 0
795     h = 0
796
797     # Estabilidad
798     a=PCB.kx/(PCB.rho_c)           #Diffusivity [m^2/s]
799     Fo=a*Dt/(Dx*Dx)              #Fourier's number
800     Bi=h*p*Dx/(PCB.kx*PCB.Ax/Dx) #Biot's number
801
802     if (1-Fo*(2+Bi)) < 0:
803         print('This is unstable increase number of time steps')
804
805     # Propiedades
806     def k_fun(posx,posy):
807         x = xen[posx]
808         y = yen[posy]
809         k1 = PCB.kx #En el plano son iguales
810         k2 = PCB.kx2
811         dis = 1
812         from numpy import heaviside as H
813         if y>=0 and y<L1y:
814             val = k1
815         elif y>=L1y and y<=L2y:
816             val = k1 + (H(x-L1x,dis)-H(x-L2x,dis)+H(x-L3x,dis)-H(x-L5x,dis)+H(x-
817                         L6x,dis)-H(x-L7x,dis))*(k2-k1) # Los IC en x
818         elif y>L2y and y<=L3y:
819             val = k1
820         return val
821
822     def rho_c_fun(posx,posy):
823         x = xen[posx]
824         y = yen[posy]
825         rho_c1 = PCB.rho_c
826         rho_c2 = PCB.rho_c2
827         dis = 1
828         from numpy import heaviside as H
```

```

828         if y>=0 and y<L1y:
829             val = rho_c1
830         elif y>=L1y and y<=L2y:
831             val = rho_c1 + (H(x-L1x,dis)-H(x-L2x,dis)+H(x-L3x,dis)-H(x-L5x,dis)+H(
832                 x-L6x,dis)-H(x-L7x,dis))*(rho_c2-rho_c1) # Los IC en x
833         elif y>L2y and y<=L3y:
834             val = rho_c1
835         return val
836
837     def phii_fun(posx,posy):
838         x = xen[posx]
839         y = yen[posy]
840         dis = 1
841         from numpy import heaviside as H
842         if y>=0 and y<L1y:
843             val = 0
844         elif y>=L1y and y<=L2y:
845             val = (H(x-L1x,dis)-H(x-L2x,dis)+H(x-L3x,dis)-H(x-L5x,dis)+H(x-L6x,dis
846                 )-H(x-L7x,dis))*phi # Los IC en x
847         elif y>L2y and y<=L3y:
848             val = 0
849         return val
850
851     k=np.zeros((len(xen),len(yen)))
852     rho_c=np.zeros((len(xen),len(yen)))
853     phii=np.zeros((len(xen),len(yen)))
854     for y in range(0,len(yen)):
855         for x in range(0,len(xen)):
856             k[x,y] = k_fun(x,y)
857             rho_c[x,y] = rho_c_fun(x,y)
858             phii[x,y] = phii_fun(x,y)
859
860     save_result('k',k)
861     save_result('rho_c',rho_c)
862     save_result('phii',phii)
863
864     save_result('xen',xen)
865     save_result('yen',yen)

```

```

864
865     T = T_wall * np.ones((M+1,Nx+1,Ny+1)) # T(t,x,y) inicial
866
867     for t in range(0,M):
868         if t%disp == 0: print('Tiempo de simulación:',te[t],'s \Temperatura máxima
869             :',npamax(T[t,:,:]),'K')
870
871         # Condiciones de contorno en la pared
872         T[t,0,:]=T_wall
873         T[t,Nx,:]=T_wall
874
875         for y in range(1,len(yen)-1):
876             for x in range(1,len(xen)-1):
877                 # Propiedades
878                 kpx = (k[x+1,y]+k[x,y])/2
879                 knx = (k[x,y]+k[x-1,y])/2
880                 kpy = (k[x,y+1]+k[x,y])/2
881                 kny = (k[x,y]+k[x,y-1])/2
882                 # Euler explícito
883                 T[t+1,x,y] = T[t,x,y] + Dt/(rho_c[x,y]*z_eff)*(\ \
884                     +kpx*z_eff*(T[t,x+1,y]-T[t,x,y])/Dx**2\ \
885                     -knx*z_eff*(T[t,x,y]-T[t,x-1,y])/Dx**2\ \
886                     +kpy*z_eff*(T[t,x,y+1]-T[t,x,y])/Dy**2\ \
887                     -kny*z_eff*(T[t,x,y]-T[t,x,y-1])/Dy**2\ \
888                     +phii[x,y]*z_eff \
889                     -( ((eps1+eps2)**0.25*sigma**0.25*T[t,x,y])**4 - ((eps1+eps2)
890                         **0.25*sigma**0.25*T_inf)**4))
891
892         # Condiciones de adiabaticidad
893         T[t+1,:,0]=T[t+1,:,1]
894         T[t+1,:,:Ny]=T[t+1,:,:Ny-1]
895
896         Ten = np.zeros((len(xen),len(yen)))
897         Ten[:, :] = T[-1, :, :]
898         T_max = npamax(Ten)
899
900         print(' T_max = ', round(T_max), 'K ó ', round(T_max-273.15), 'C')

```

```
900     # Guardar resultados
901     save_result('Ten',Ten)
```

Para convertir los resultados obtenidos en imágenes se ha escrito el siguiente código, también en Python donde se hace uso de las librerías *numpy* y *matplotlib*. De nuevo se puede elegir el archivo a representar mediante un archivo *data\_plot*.

[dvips]graphicx

```
1 import os
2 import numpy as np
3 from numpy import genfromtxt
4 import matplotlib.pyplot as plt
5
6 # Para saber qué apartado correr
7 data = 'data_plot' #Nombre del archivo
8 try:
9     with open(data, 'r') as data:
10         for line in data:
11             if 'apartado_a' in line:
12                 p = line.split()
13                 apartado_a = p[2]
14             if 'apartado_b' in line:
15                 p = line.split()
16                 apartado_b = p[2]
17             if 'apartado_c' in line:
18                 p = line.split()
19                 apartado_c = p[2]
20             if 'apartado_d' in line:
21                 p = line.split()
22                 apartado_d = p[2]
23             if 'apartado_e' in line:
24                 p = line.split()
25                 apartado_e = p[2]
26             if 'all' in line:
27                 p = line.split()
28                 all = p[2]
29             if 'numerico' in line:
30                 p = line.split()
31                 numerico = p[2]
```

```
32 except:  
33     ### Apartados  
34     apartado_a = 'yes'  
35     apartado_b = 'yes'  
36     apartado_c = 'yes'  
37     apartado_d = 'yes'  
38     apartado_e = 'yes'  
39     all = 'yes'  
40     ###  
41     numerico = 'yes'  
42  
43     print('No data configuration file found')  
44  
45  
46     figures_dir = './Figures/'  
47     if not os.path.exists(figures_dir):  
48         os.makedirs(figures_dir)  
49  
50     results_dir = './Results/'  
51  
52     def read_result(name):  
53         my_data = genfromtxt(results_dir+str(name)+'.csv', delimiter=',')  
54         return my_data  
55  
56     ### Apartado a  
57     if apartado_a == 'yes':  
58         print('*** a ***')  
59  
60         xa = read_result('xa')  
61         Ta1 = read_result('Ta1')  
62         Ta2 = read_result('Ta2')  
63  
64         print('Solución analítica')  
65         print(' Potencia puntual : T_max = ', round(npamax(Ta1)), 'K ó', round(npamax(Ta1)  
66             -273.15), 'C')  
67         print(' Potencia uniforme: T_max = ', round(npamax(Ta2)), 'K ó', round(npamax(Ta2)  
68             -273.15), 'C')
```

```
68     fig = plt.figure()
69 #
70     plt.plot(xa*1e3,Ta1)
71     plt.plot(xa*1e3,Ta2)
72     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
73     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
74     h.set_rotation(0)
75     plt.title('Distribución de temperatura')
76     plt.legend(['Potencia puntual','Potencia uniforme'])
77     plt.grid()
78     plt.savefig(figures_dir+'a_analytic.pdf')
79     plt.close()
80
81 if numerico == 'yes':
82
83     xan = read_result('xan')
84     Ta1n = read_result('Ta1n')
85     Ta2n = read_result('Ta2n')
86
87     print('Solución numérica')
88     print(' Potencia puntual : T_max = ',round(npamax(Ta1n),'K ó',round(npamax(
89         Ta1n)-273.15), 'C')
90     print(' Potencia uniforme: T_max = ',round(npamax(Ta2n),'K ó',round(npamax(
91         Ta2n)-273.15), 'C')
92
93     fig = plt.figure()
94 #
95     plt.plot(xan*1e3,Ta1n)
96     plt.plot(xan*1e3,Ta2n)
97     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
98     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
99     h.set_rotation(0)
100    plt.title('Distribución de temperatura')
101    plt.legend(['Potencia puntual','Potencia uniforme'])
102    plt.grid()
103    plt.savefig(figures_dir+'a_numeric.pdf')
104    plt.close()
```

```
104
105     fig = plt.figure()
106     #
107     plt.plot(xa*1e3,Ta1)
108     plt.plot(xa*1e3,Ta2)
109     plt.plot(xan*1e3,Ta1n)
110     plt.plot(xan*1e3,Ta2n)
111     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
112     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
113     h.set_rotation(0)
114     plt.title('Distribución de temperatura')
115     plt.legend(['Potencia puntual (analítica)', 'Potencia uniforme (analítica)', 'Potencia puntual (numérica)', 'Potencia uniforme (numérica)'])
116     plt.grid()
117     plt.savefig(figures_dir+'a.pdf')
118     plt.close()
119
120 #### Apartado b
121
122 if apartado_b == 'yes':
123     print('*** b ***')
124
125     xb = read_result('xb')
126     Tb1 = read_result('Tb1')
127     Tb2 = read_result('Tb2')
128
129     print('Solución analítica')
130     print(' Con k I C : T_max = ', round(npamax(Tb1)), 'K ó', round(npamax(Tb1)-273.15), 'C')
131     print(' Con kIC dada: T_max = ', round(npamax(Tb2)), 'K ó', round(npamax(Tb2)-273.15), 'C')
132
133     fig = plt.figure()
134     #
135     plt.plot(xb*1e3,Tb1)
136     plt.plot(xb*1e3,Tb2)
137     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
138     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
```

```
139     h.set_rotation(0)
140     plt.title('Distribución de temperatura')
141     plt.legend([' k I C ', 'kIC dado'])
142     plt.grid()
143     plt.savefig(figures_dir+'b_analytic.pdf')
144     plt.close()
145
146 if numerico == 'yes':
147
148     xbn = read_result('xbn')
149     Tb1n = read_result('Tb1n')
150     Tb2n = read_result('Tb2n')
151
152     print('Solución numérica')
153     print(' Con k I C : T_max = ', round(npamax(Tb1n)), 'K ó', round(npamax(
154         Tb1n)-273.15), 'C')
155     print(' Con kIC dada: T_max = ', round(npamax(Tb2n)), 'K ó', round(npamax(Tb2n)
156         -273.15), 'C')
157
158     fig = plt.figure()
159     #
160     plt.plot(xbn*1e3,Tb1n)
161     plt.plot(xbn*1e3,Tb2n)
162     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
163     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
164     h.set_rotation(0)
165     plt.title('Distribución de temperatura')
166     plt.legend([' k I C ', 'kIC dado'])
167     plt.grid()
168     plt.savefig(figures_dir+'b_numeric.pdf')
169     plt.close()
170
171     fig = plt.figure()
172     #
173     plt.plot(xb*1e3,Tb1)
174     plt.plot(xb*1e3,Tb2)
175     plt.plot(xbn*1e3,Tb1n)
176     plt.plot(xbn*1e3,Tb2n)
```

```
175     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
176     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
177     h.set_rotation(0)
178     plt.title('Distribución de temperatura')
179     plt.legend([' k I C (analítica)', 'kIC dado (analítica)', ' k I C (numérica)', 'kIC dado (numérica)'])
180     plt.grid()
181     plt.savefig(figures_dir+'b.pdf')
182     plt.close()
183
184 #### Apartado c
185
186 if apartado_c == 'yes':
187     print('*** c ***')
188
189     xc = read_result('xc')
190     Tc = read_result('Tc')
191
192     print('Solución analítica')
193     print(' T_max = ', round(npamax(Tc)), 'K ó ', round(npamax(Tc)-273.15), 'C')
194
195     fig = plt.figure()
196     #
197     plt.plot(xc*1e3,Tc)
198     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
199     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
200     h.set_rotation(0)
201     plt.title('Distribución de temperatura')
202     plt.grid()
203     plt.savefig(figures_dir+'c_analytic.pdf')
204     plt.close()
205
206 if numerico == 'yes':
207
208     xcn = read_result('xcn')
209     Tcn = read_result('Tcn')
210
211     print('Solución numérica')
```

```
212     print(' T_max = ',round(npamax(Tcn)), 'K ó', round(npamax(Tcn)-273.15), 'C')
213
214     fig = plt.figure()
215     #
216     plt.plot(xcn*1e3,Tcn)
217     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
218     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
219     h.set_rotation(0)
220     plt.title('Distribución de temperatura')
221     plt.grid()
222     plt.savefig(figures_dir+'c_numeric.pdf')
223     plt.close()
224
225     fig = plt.figure()
226     #
227     plt.plot(xc*1e3,Tc)
228     plt.plot(xcn*1e3,Tcn)
229     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
230     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
231     h.set_rotation(0)
232     plt.title('Distribución de temperatura')
233     plt.legend(['Solución analítica','Solución numérica'])
234     plt.grid()
235     plt.savefig(figures_dir+'c.pdf')
236     plt.close()
237
238 ### Apartado d
239
240 if apartado_d == 'yes':
241     print('*** d ***')
242
243     xdn = read_result('xdn')
244     Tdn = read_result('Tdn')
245
246     print(' T_max = ',round(npamax(Tdn)), 'K ó', round(npamax(Tdn)-273.15), 'C')
247
248     fig = plt.figure()
249     #
```

```
250 plt.plot(xdn*1e3,Tdn)
251 plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
252 h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
253 h.set_rotation(0)
254 plt.title('Distribución de temperatura')
255 plt.grid()
256 plt.savefig(figures_dir+'d.pdf')
257 plt.close()
258
259 ### Apartado e
260
261 if apartado_e == 'yes':
262     print('*** e ***')
263
264 xen = read_result('xen')
265 yen = read_result('yen')
266 Ten = read_result('Ten')
267
268 print(' T_max = ',round(npamax(Ten)), 'K ó ',round(npamax(Ten)-273.15), 'C')
269
270 x,y = np.meshgrid(xen,yen)
271 z = Ten.transpose()
272
273 fig = plt.figure()
274 ax = fig.add_subplot(111)
275 plt.title('Distribución de temperatura: T[K]')
276 clev = np.arange(z.min()-1.15,z.max()+1-0.15,1)
277 CS3 = plt.contourf(x,y,z,clev,cmap='jet')
278 bar = plt.colorbar()
279 plt.xlabel('x[mm]')
280 plt.ylabel('y[mm]')
281 plt.grid()
282 plt.savefig(figures_dir+'e.png')
283 plt.close()
284
285 fig = plt.figure()
286 ax = fig.gca(projection='3d')
287 surf = ax.plot_surface(x*1e3, y*1e3, z, cmap='jet')
```

```
288 plt.xlabel('x[mm]')
289 plt.ylabel('y[mm]')
290 ax.set_zlabel('T[K]')
291 plt.grid()
292 plt.show()
293 plt.close()
294
295 ### Conductividad
296
297 xen = read_result('xen')
298 yen = read_result('yen')
299 k = read_result('k')
300
301 x,y = np.meshgrid(xen,yen)
302 z = k.transpose()
303
304 fig = plt.figure()
305 ax = fig.add_subplot(111)
306 plt.title('Conductividad [W/(m K)]')
307 clev = np.arange(z.min()-5,z.max()+5,1)
308 CS3 = plt.contourf(x,y,z, clev, cmap='jet')
309 bar = plt.colorbar()
310 plt.xlabel('x[mm]')
311 plt.ylabel('y[mm]')
312 plt.grid()
313 plt.savefig(figures_dir+'k.pdf')
314 plt.close()
315
316 ### rho_c
317
318 xen = read_result('xen')
319 yen = read_result('yen')
320 rho_c = read_result('rho_c')
321
322 x,y = np.meshgrid(xen,yen)
323 z = rho_c.transpose()
324
325 fig = plt.figure()
```

```
326     ax = fig.add_subplot(111)
327     plt.title(' c [J/( K m )]')
328     clev = np.arange(z.min()-1e2,z.max()+1e2,1e2)
329     CS3 = plt.contourf(x,y,z, clev, cmap='jet')
330     bar = plt.colorbar()
331     plt.xlabel('x[mm]')
332     plt.ylabel('y[mm]')
333     plt.grid()
334     plt.savefig(figures_dir+'rho_c.pdf')
335     plt.close()
336
337 ### Disipación
338
339 xen = read_result('xen')
340 yen = read_result('yen')
341 phi = read_result('phi')
342
343 x,y = np.meshgrid(xen,yen)
344 z = phi.transpose()
345
346 fig = plt.figure()
347 ax = fig.add_subplot(111)
348 plt.title('Disipación [W/m ]')
349 clev = np.arange(z.min()-1e3,z.max()+1e3,1e3)
350 CS3 = plt.contourf(x,y,z, clev, cmap='jet')
351 bar = plt.colorbar()
352 plt.xlabel('x[mm]')
353 plt.ylabel('y[mm]')
354 plt.grid()
355 plt.savefig(figures_dir+'phi.pdf')
356 plt.close()
357
358
359 ### Representación conjunta
360
361 if (apartado_a == 'yes' and apartado_b == 'yes' and apartado_c == 'yes' and apartado_d
362 == 'yes'):
363
```

```
363 # Analíticas
364 fig = plt.figure()
365 #
366 plt.plot(xa*1e3,Ta1)
367 plt.plot(xa*1e3,Ta2)
368 plt.plot(xb*1e3,Tb1)
369 plt.plot(xb*1e3,Tb2)
370 plt.plot(xc*1e3,Tc)
371 plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
372 h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
373 h.set_rotation(0)
374 plt.title('Distribución de temperatura')
375 plt.legend(['Potencia puntual','Potencia uniforme','k I C ','kIC dado','Radiaci
376 ón lineal'])
377 plt.grid()
378 plt.savefig(figures_dir+'all_analytic.pdf')
379 plt.close()
380
381 if numerico == 'yes':
382     # Numéricas
383     fig = plt.figure()
384     #
385     plt.plot(xan*1e3,Ta1n)
386     plt.plot(xan*1e3,Ta2n)
387     plt.plot(xbn*1e3,Tb1n)
388     plt.plot(xbn*1e3,Tb2n)
389     plt.plot(xcn*1e3,Tcn)
390     plt.plot(xdn*1e3,Tdn)
391     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)
392     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)
393     h.set_rotation(0)
394     plt.title('Distribución de temperatura')
395     plt.legend(['Potencia uniforme','k I C ','kIC dado','Radiación lineal','
396 Radiación & IC'])
397     plt.grid()
398     plt.savefig(figures_dir+'all_numeric.pdf')
399     plt.close()
```

```
399 if all == 'yes':  
400     print('*** all ***')  
401  
402     xa = read_result('xa')  
403     Ta1 = read_result('Ta1')  
404     Ta2 = read_result('Ta2')  
405  
406     xb = read_result('xb')  
407     Tb1 = read_result('Tb1')  
408     Tb2 = read_result('Tb2')  
409  
410     xc = read_result('xc')  
411     Tc = read_result('Tc')  
412  
413  
414     xdn = read_result('xdn')  
415     Tdn = read_result('Tdn')  
416  
417     fig = plt.figure()  
418     #  
419     plt.plot(xa*1e3,Ta1)  
420     plt.plot(xa*1e3,Ta2)  
421     plt.plot(xb*1e3,Tb1)  
422     plt.plot(xb*1e3,Tb2)  
423     plt.plot(xc*1e3,Tc)  
424     plt.plot(xdn*1e3,Tdn)  
425     plt.xlabel('x[mm]', horizontalalignment='right', x=1.0)  
426     h = plt.ylabel('T[K]', horizontalalignment='right', y=1.0)  
427     h.set_rotation(0)  
428     plt.title('Distribución de temperatura')  
429     plt.legend([('1) puntual','(1) uniforme','(2) k I C ','(2) KIC dado','(3)','(4)  
        '])  
430     plt.grid()  
431     plt.savefig(figures_dir+'all.pdf')  
432     plt.close()
```

Más detalles en <https://github.com/temisAP/TCCT>.