UNIVERSIDAD TECNOLÓGICA NACIONAL

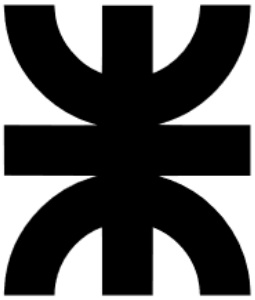
Facultad Regional Santa Fe

---

# Matemática Superior

*Ingeniería en Sistemas de Información*

## Trabajo Práctico 4




| Alumno | Correo electrónico |
|---|---|
| Izaguirre, Ezequiel | ezequielmizaguirre@gmail.com |
| Pacheco Pilan, Federico Ignacio | fedepacheco2112@gmail.com |
| Rodríguez, Alejandro | rodriguezalejandro_@hotmail.com |


ISI A • 2021

# 1. Herramientas utilizadas

Las librerías más relevantes del lenguaje de programación Python utilizadas para la realización del trabajo fueron:

- *numpy*
- *sympy*
- *matplotlib*
- *math*

# 2. Integración númerica

## 2.1. Formulación de los métodos:

Dado que solo se poseen tres puntos, a saber, $(t_0; x(t_0))$, $(t_0 + h; x(t_0 + h))$ y $(t_0 + 2h, x(t_0 + 2h))$, es posible interpolar a la función mediante dos rectas o una parábola, similarmente a lo que realizan la regla del trapecio y Simpson, respectivamente. Como los extremos de integración no son los mismos que los aplicados para las fórmulas cerradas y abiertas de cada uno (con los cuales se podrían emplear las fórmulas originales), un enfoque sencillo para extremos de integración arbitrarios consiste en realizar interpolaciones con los tres puntos dados, hallar sus antiderivadas (factible tratándose de polinomios) y emplear el teorema fundamental del cálculo para aproximar las integrales originales.

Si:

$$d_1 = \frac{x(t_0 + h) - x(t_0)}{h} \; ; \; d_2 = \frac{x(t_0 + 2h) - x(t_0 + h)}{h}$$

para las rectas de $(t_0; x(t_0))$ *a* $(t_0 + h; x(t_0 + h))$ y de $(t_0 + h; x(t_0 + h))$ a $(t_0 + 2h, x(t_0 + 2h))$ se tiene:

$$x_{r1}(t) = x(t_0) + d_1(t - t_0)$$

$$X_{r1}(t) = x(t_0)\, t + d_1\left(\frac{t^2}{2} - t_0 t\right)$$

$$x_{r2}(t) = x(t_0 + h) + d_2(t - (t_0 + h))$$

$$X_{r2}(t) = x(t_0 + h)\, t + d_2\left(\frac{t^2}{2} - (t_0 + h)\, t\right)$$

Para la parábola se tiene:

$$x_p(t) = x(t_0) + d_1(t - t_0) + \frac{d_2 - d_1}{2h}(t - t_0)(t - (t_0 + h))$$

$$X_p(t) = c_1 t^3 + c_2 t^2 + c_3 t$$

donde:

$$c_1 = \frac{d_2 - d_1}{6h} \; ; \; c_2 = \frac{1}{2}\left(d_1 + \frac{(d_1 - d_2)(2t_0 + h)}{2h}\right) ; c_3 = x(t_0) + t_0\left(\frac{(t_0 + h)(d_2 - d_1)}{2h} - d_1\right)$$

Finalmente, el cálculo de las integrales resulta:

| Método | General *(a; b)* | Semiabierta | Supracerrada |
|---|---|---|---|
| "Trapecios" | $X_{r1}(t_0 + h) - X_{r1}(a) + X_{r2}(b) - X_{r2}(t_0 + h)$ | $X_{r2}(t_0 + 2.5h) - X_{r2}(t_0 + h)$ | $X_{r1}(t_0 + h) - X_{r1}(t_0 + 0.5h) + X_{r2}(t_0 + 1.5h) - X_{r2}(t_0 + h)$ |
| "Simpson" | $X_p(b) - X_p(a)$ | $X_p(t_0 + 2.5h) - X_p(t_0 + h)$ | $X_p(t_0 + 1.5h) - X_p(t_0 + 0.5h)$ |

**2.2. Utilidad y orden del error:**

Ambos métodos pueden usarse de manera análoga a las reglas del trapecio y Simpson originales, cerradas y abiertas (dependiendo de dónde se coloquen los límites de integración), aunque el costo del cálculo es un poco mayor. Por otra parte, se pueden hacer extrapolaciones arbitrariamente lejanas a ambos lados de los tres puntos originales. Esto puede resultar en una aproximación buena o mala, dependiendo de si la función original tiene un forma aproximadamente recta o parabólica en las cercanías de los puntos originales o no.

Respecto del orden de error, como en la regla del trapecio y Simpson originales, es de $O(h^3)$ y $O(h^5)$, respectivamente.

# 3. Resolución de EDOs

## 3.1. Método predictor-corrector-supracorrector:

Se hizo uso de una versión ligeramente modificada del método de Milne tratado en las clases de práctica de la materia, que integra empleando el método de Simpson modificado discutido en *Integración numérica*; es vectorial, esto es, aplicable a sistemas de EDOs (para el caso escalar, basta considerar vectores 1D) y a nivel de implementación corrige las veces que se requieran (por defecto, 2 veces), estimando el error y sumando el estimador en cada paso. Para los primeros 3 pasos se hace uso de Runge-Kutta 4 vectorial con estimador del error incorporado.

Más específicamente, sea *t* una variable independiente y escalar,

$$\mathbf{x}(t) = < x_1(t), x_2(t), \dots , x_n(t) >$$

el vector de las variables dependientes,

$$\mathbf{x'}(t) = < x_1'(t), x_2'(t), \dots , x_n'(t) >$$

el vector de las derivadas con respecto a *t* de las variables dependientes,

$$\mathbf{f}(t, \mathbf{x}(t)) = < \mathbf{f_1}(t, \mathbf{x}(t)), \mathbf{f_2}(t, \mathbf{x}(t)), \dots , \mathbf{f_n}(t, \mathbf{x}(t)) > , \ \mathbf{f_i}(t, \mathbf{x}(t)): (\Re, \Re^{n}) \rightarrow \Re$$

un vector de funciones escalares. El método resuelve sistemas de EDOs de la forma:

$$\mathbf{x'}(t) = \mathbf{f}(t, \mathbf{x}(t))$$

Para ello, si se denota con *simpson3(a, b, t0, h,* $\boldsymbol{x'_1}$, $\boldsymbol{x'_2}$, $\boldsymbol{x'_3}$*)* el método vectorial que aplica componente a componente la integración escalar desarrollada anteriormente, se aplican iterativamente las siguientes fórmulas, siendo *h* el paso:

$$t_{i+1} = t_i + h$$

Predicción:

$$\mathbf{x}^{P}_{i+1} = x_{i-3} + simpson3(t_{i-3}, t_{i+1}, t_{i-2}, h, \mathbf{x'}_{i-2}, \mathbf{x'}_{i-1}, \mathbf{x'}_{i})$$

$$\mathbf{x'}^{P}_{i+1} = \mathbf{f}(t_{i+1}, \mathbf{x}^{P}_{i+1})$$

Corrección, supracorrección, … (iterar al menos 2 veces):

$$\mathbf{x}^{C}_{i+1} = x_{i-1} + simpson3(t_{i-1}, t_{i+1}, t_{i-1}, h, \mathbf{x'}_{i-1}, \mathbf{x'}_{i}, \mathbf{x'}^{P}_{i+1})$$

$$est(E(\mathbf{x}^{C}_{i+1})) = -29^{-1} (\mathbf{x}^{C}_{i+1} - \mathbf{x}^{P}_{i+1})$$

$$\mathbf{x}^{C}_{i+1} \leftarrow \mathbf{x}^{C}_{i+1} + est(E(\mathbf{x}^{C}_{i+1}))$$

$$\mathbf{x'}^{C}_{i+1} = \mathbf{f}(t_{i+1}, \mathbf{x}^{C}_{i+1})$$
$$\mathbf{x}^{P}_{i+1} \leftarrow \mathbf{x}^{C}_{i+1}$$
$$\mathbf{x'}^{P}_{i+1} \leftarrow \mathbf{x'}^{C}_{i+1}$$

Para Runge-Kutta 4, empleado en los primeras 3 iteraciones, se usan las siguientes expresiones de cálculo:

$$\mathbf{k_1} = \mathbf{f}(t_i, \mathbf{x}(t_i))$$
$$\mathbf{k_2} = \mathbf{f}(t_i + 2^{-1} h, \mathbf{x}(t_i) + 2^{-1} h\, \mathbf{k_1})$$
$$\mathbf{k_3} = \mathbf{f}(t_i + 2^{-1} h, \mathbf{x}(t_i) + 2^{-1} h\, \mathbf{k_2})$$
$$\mathbf{k_4} = \mathbf{f}(t_i + h, \mathbf{x}(t_i) + h\, \mathbf{k_3})$$

$$t_{i+1} = t_i + h$$
$$\mathbf{x}_{i+1} = \mathbf{x}_i + 6^{-1} h\, (\mathbf{k_1} + 2\, \mathbf{k_2} + 2\, \mathbf{k_3} + \mathbf{k_4})$$
$$\mathbf{x'}_{i+1} = \mathbf{f}(t_{i+1}, \mathbf{x}_{i+1})$$

$$est(\mathrm{E}(\mathbf{x}_{h/2})) = 15^{-1} (\mathbf{x}_{h/2} - \mathbf{x}_h)$$

## 3.2. Aplicación a ECG:

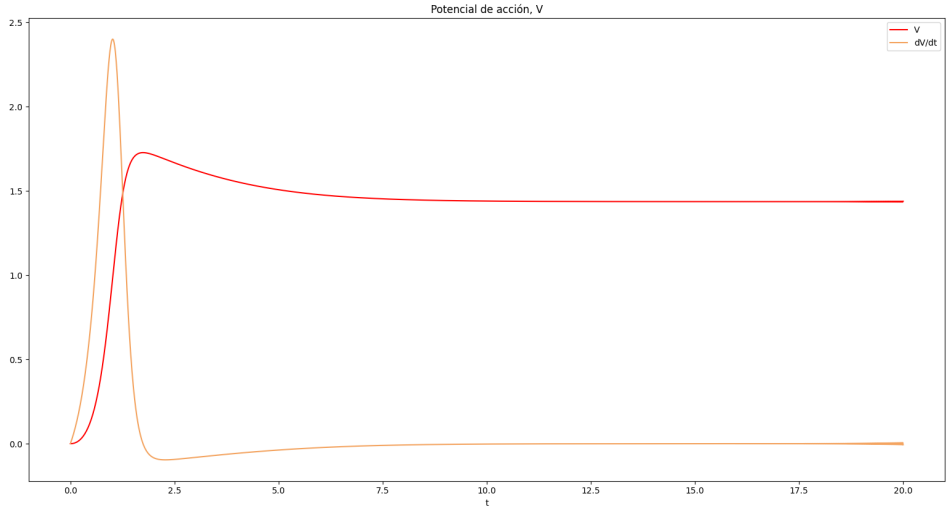
Se tiene:

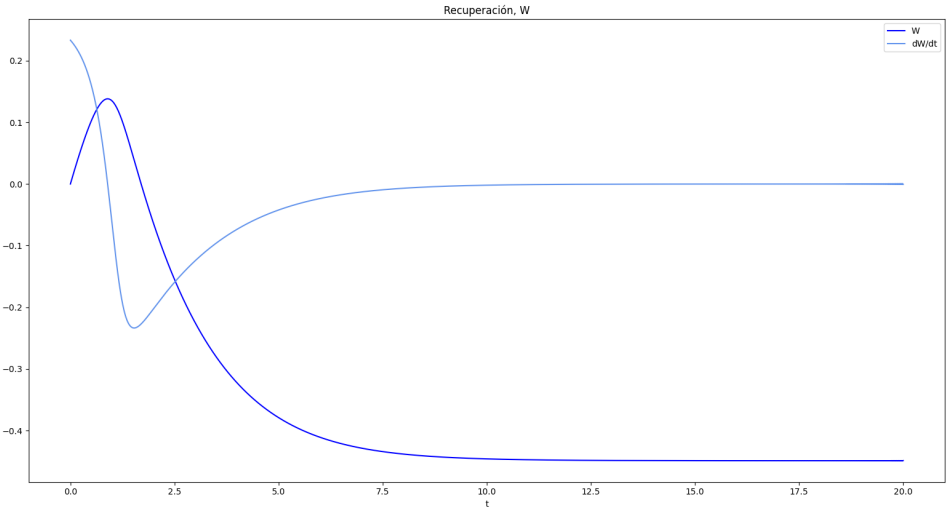
$$\mathbf{x}(t) = < V(t), W(t) >$$
$$\mathbf{x'}(t) = < V'(t), W'(t) >$$
$$\mathbf{f}(t, \mathbf{x}(t)) = < 3\,(V(t) + W(t) - 3^{-1} V^3(t) + Z), -3^{-1}\,(W(t) - 0.7 + 0.8\, V(t)) >$$

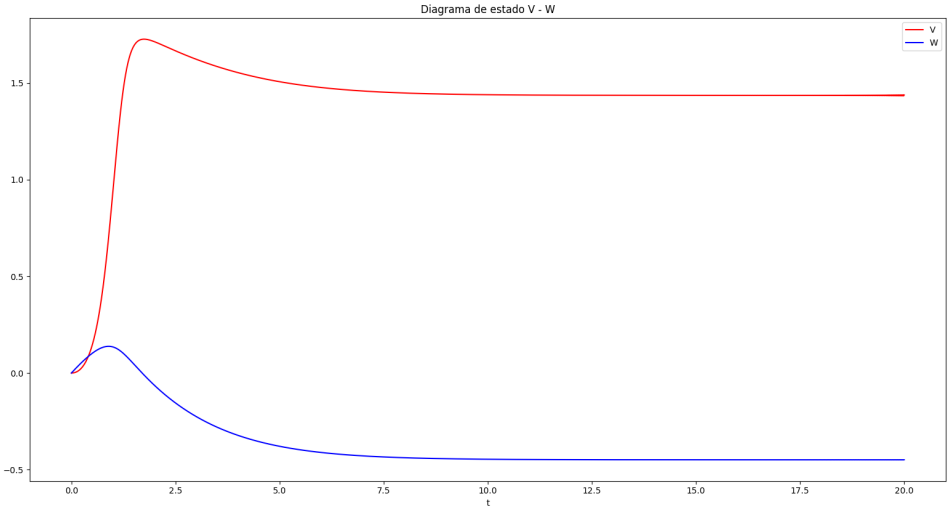
Con las condiciones

$$t_0 = 0$$
$$t_f = 20$$
$$h = 0.01$$
$$Z = 0$$
$$\mathbf{x}_0 = < V(t_0) = 0, W(t_0) = 0 >$$

se obtiene


Potencial de acción, V
V
dV/dt
2.5
2.0
1.5
1.0
0.5
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t


Recuperación, W
W
dW/dt
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
t




Se sabe que el orden del error para el método de Milne para el caso escalar es $O(h^5)$. Se sigue que para el caso vectorial el orden del error es equivalente componente a componente. Como en cada corrección se suma el estimador del error de Richardson, se incrementa en uno el orden del error. Así, para $n \in Z^+$ correcciones, $O(h^{5+n})$. Para el caso de la consigna original del trabajo práctico, $n = 2$. Luego, $O(h^7)$.

# 4. Modelo cardíaco 2D

### 4.1.1. Formulación del problema:

Dado que el enunciado trata un modelo bidimensional del corazón a emplearse para estudiar su conservación en bajas temperaturas, se parte de la ecuación del calor para coordenadas cartesianas en $\Re^2$:

$$k\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2}\right) = \frac{\partial T}{\partial t}$$

Si se aproximan a las derivadas parciales espaciales mediante diferencias centradas, se tiene:

$$\frac{\partial^2 T}{\partial x^2} \approx \frac{T(x+h_x, y, t) - 2\,T(x, y, t) + T(x-h_x, y, t)}{h_x^2}$$

$$\frac{\partial^2 T}{\partial y^2} \approx \frac{T(x+h_y, y, t) - 2\,T(x, y, t) + T(x-h_y, y, t)}{h_y^2}$$

Para la derivada parcial temporal, empleando una diferencia atrasada, se tiene:

$$\frac{\partial T}{\partial t} \approx \frac{T(x, y, t) - T(x, y, t-h_t)}{h_t}$$

Así, reemplazando en la ecuación del calor y despejando,

$$T(x, y, t - h_t) = (1 + 2\,k(x, y)\,h_t\left(\frac{1}{h_x^2} + \frac{1}{h_y^2}\right))\,T(x, y, t)$$

$$- h_t\left(\frac{k(x+h_x, y)}{h_x^2}\,T(x + h_x, y, t) + \frac{k(x-h_x, y)}{h_x^2}\,T(x - h_x, y, t)\right.$$

$$\left. + \frac{k(x, y+h_y)}{h_y^2}\,T(x, y + h_y, t) + \frac{k(x, y-h_y)}{h_y^2}\,T(x, y - h_y, t)\right)$$

Si se discretiza haciendo

$$x = x_0 + j\,h_x$$

$$y = y_0 + i\,h_y$$

$$t = t_0 + l\,h_t$$

se obtiene finalmente,

$$T[i,\ j,\ l-1]\ =\ (1\ +\ 2\,k[i,\ j]\,h_t(\frac{1}{h_x^{\,2}}+\frac{1}{h_y^{\,2}}))\,T[i,\ j,\ l]$$

$$-\ h_t(\frac{k[i,\,j+1]}{h_x^{\,2}}\,T\,[i\,,\ j\ +\ 1,\ l]\ +\frac{k[i,\,j-1]}{h_x^{\,2}}\,T\,[i,\ j\ -\ 1,\ l]$$

$$+\ \frac{k[i+1,\,j]}{h_y^{\,2}}\,T\,[i\ +\ 1,\ j,\ l]\ +\ \frac{k[i-1,\,j]}{h_y^{\,2}}\,T\,[i\ -\ 1,\ j,\ l])$$

Resolviendo el sistema de ecuaciones lineales resultante se obtiene una aproximación para cada *T[i, j, l]* dentro de la malla.

Para *k(x, y)* se considera

$k_{rojo} = 0.5\ mm^2 / s$ (paredes del corazón)

$k_{naranja} = 0.8\ mm^2 / s$ (interior del corazón)

$k_{celeste} = 0.5\ mm^2 / s$ (líquido refrigerante)

Como condiciones iniciales se consideran:

$T_{paredes} = 35\ °C$

$T_{interior} = 37\ °C$

$T_{líquido} = -150\ °C$

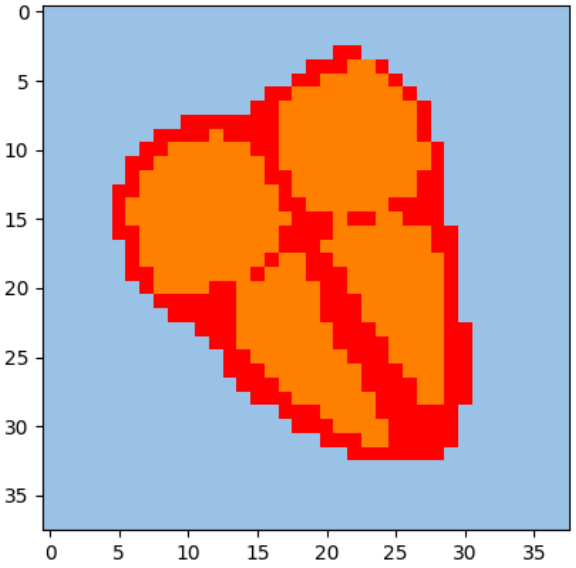
Como condiciones de borde se considera que T = -150 °C para todo el perímetro exterior.

## 4.2. Estudio de los resultados obtenidos:

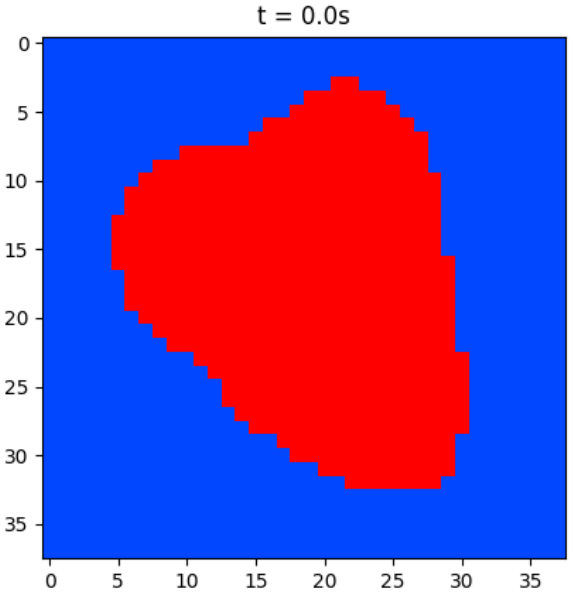
Cada uno de los siguientes ejemplos fue ejecutado bajo las mismas condiciones con un procesador Intel Core i7 10750H y 16 GB de RAM, permitiendo así usar el tiempo de cálculo como otro parámetro más para comparar.

### 4.2.1. Con variación en el paso del tiempo:

Para todos los casos, se considera un espaciado entre nodos de la malla de $h_x = h_y =$ 5mm, obteniéndose la siguiente discretización:

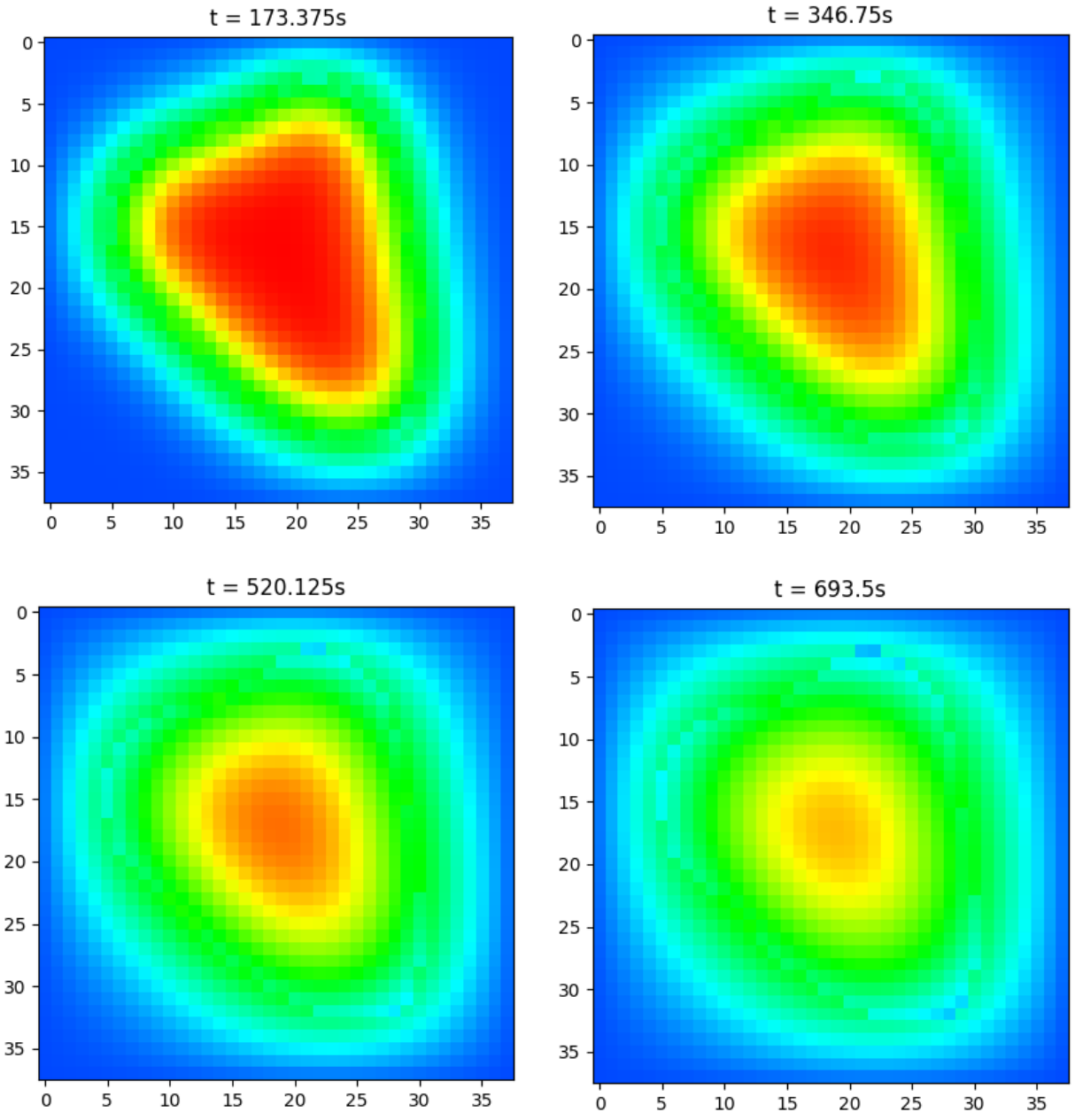



De este modo, en t = 0, se parte desde lo siguiente:



4.2.1.1. Ejemplo 1[1]:

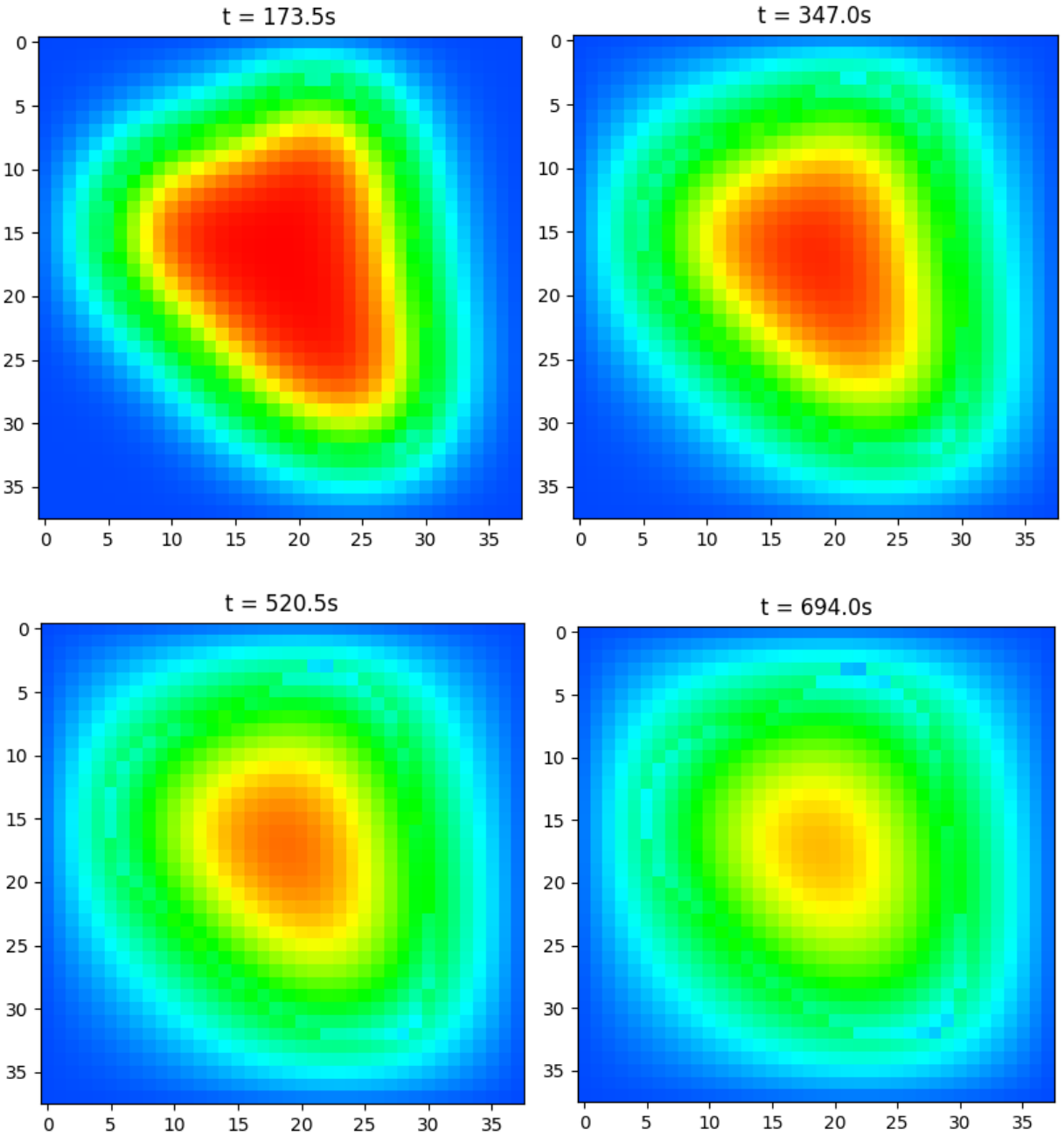
- Paso en el tiempo: $h_t = 0.1s$
- Tiempo de cálculo: 257.6945s
- Tiempo de enfriamiento para llegar por debajo de *0* ºC: 693.5s




[1] Nota: la gama de colores representa las temperaturas desde 37 ºC (rojo intenso) hasta -150ºC (azul intenso). Un naranja tenue representa alrededor de 0 °C.

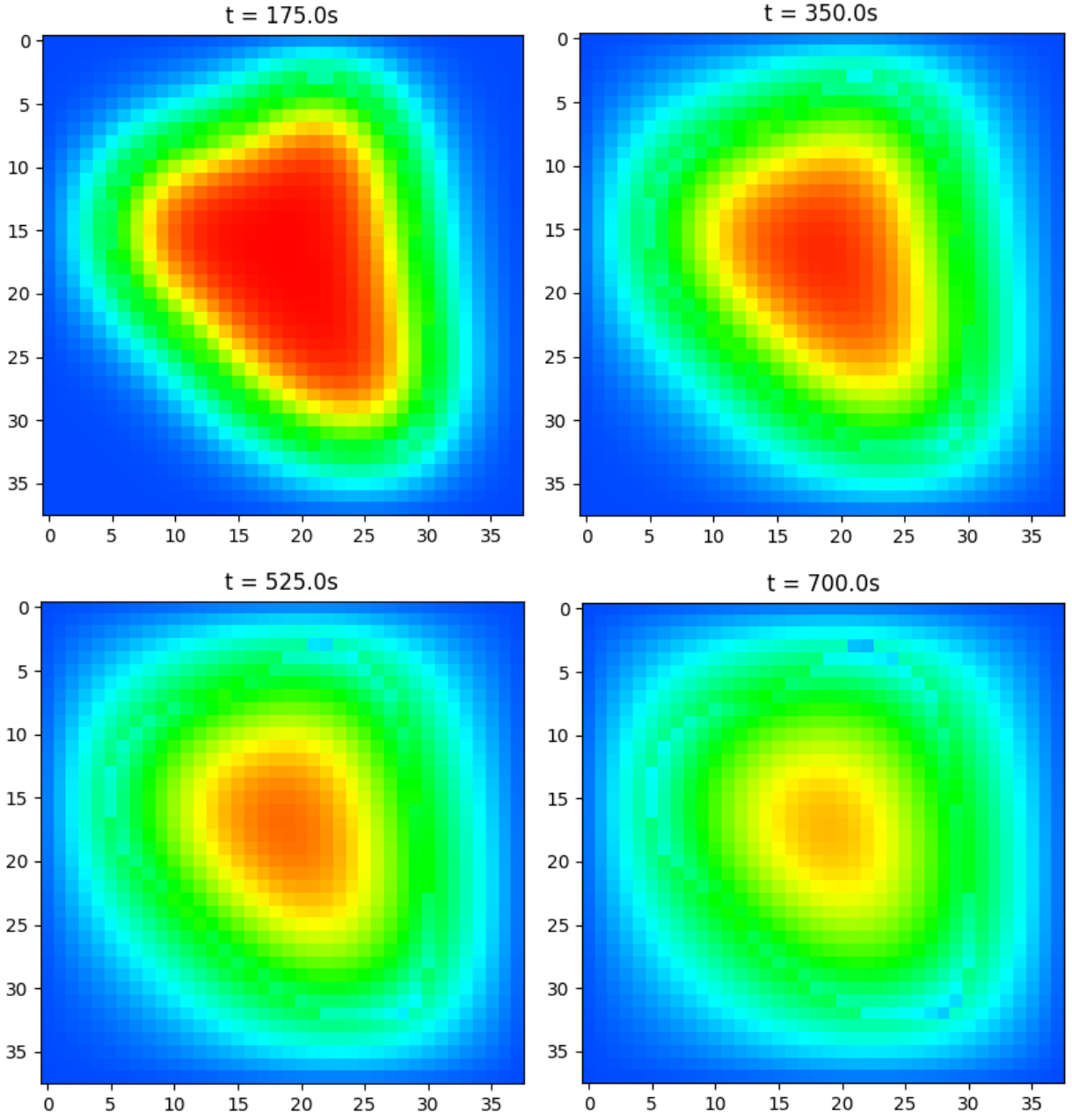
4.2.1.2. Ejemplo 2:

- Paso en el tiempo: $h_t = 1s$
- Tiempo de cálculo: 27.9489s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 694s



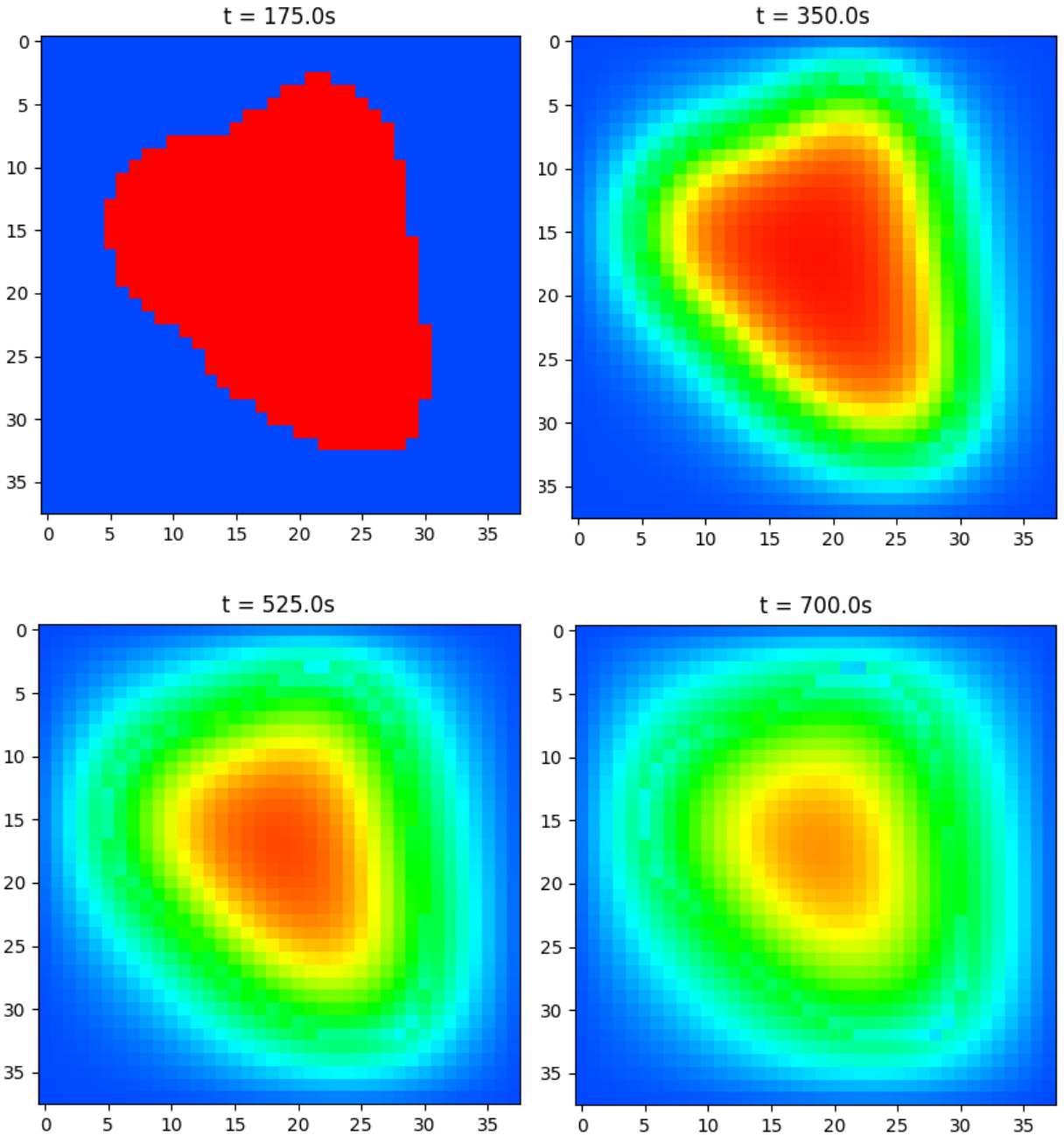
4.2.1.3. Ejemplo 3:

- Paso en el tiempo: $h_t = 10s$
- Tiempo de cálculo: 2.937s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 700s



4.2.1.4. Ejemplo 4:

- Paso en el tiempo: $h_t = 100s$
- Tiempo de cálculo: 0.5328s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 700s

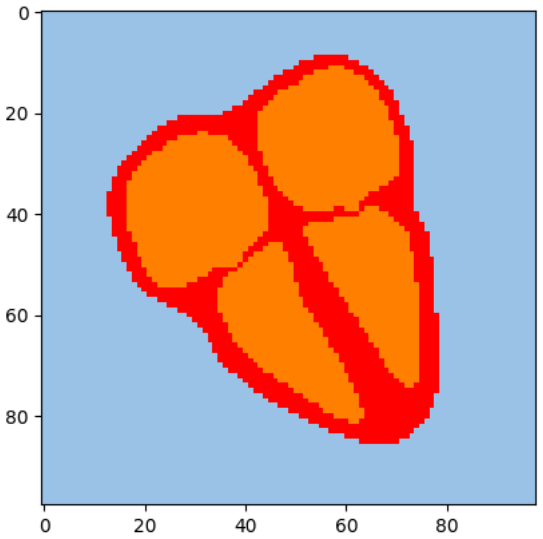


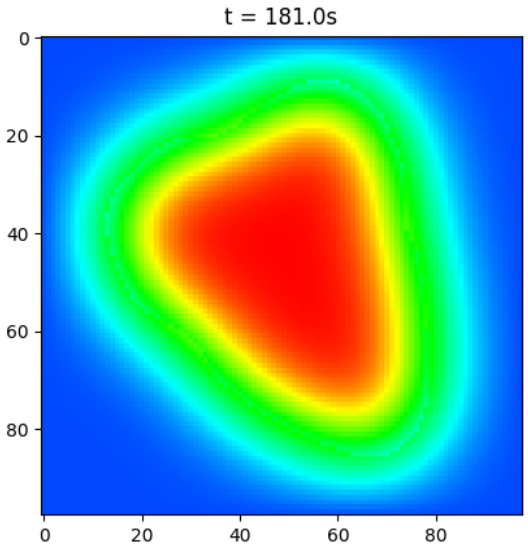
4.2.2. Con variación de la calidad de la malla:

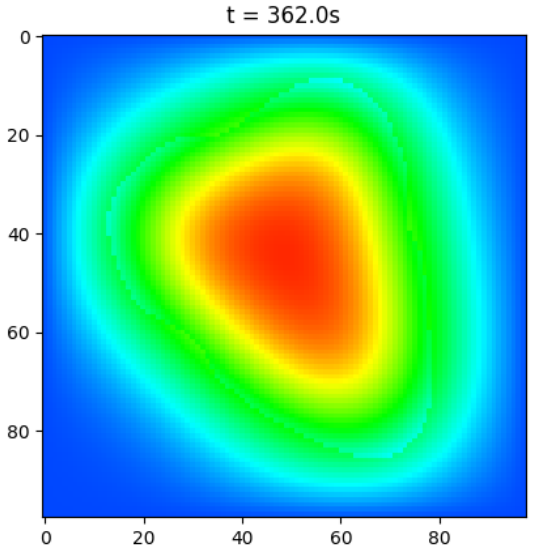
Para todos los casos, se considera un paso en el tiempo de $h_t = 1s$ y para el espacio $h_x = h_y = h$.

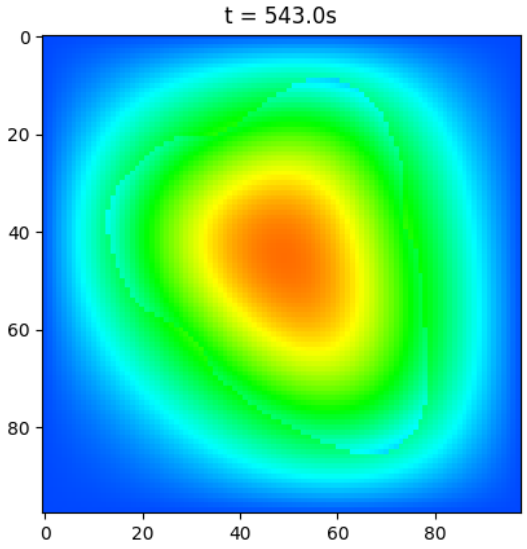
4.2.2.1. Ejemplo 1:

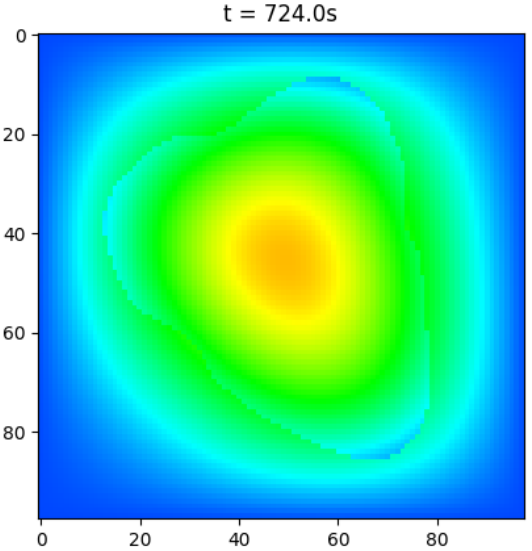
- Distancia entre nodos: h = 2mm
- Tiempo de cálculo: 2729.08s ≈ 45min 29s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 724s



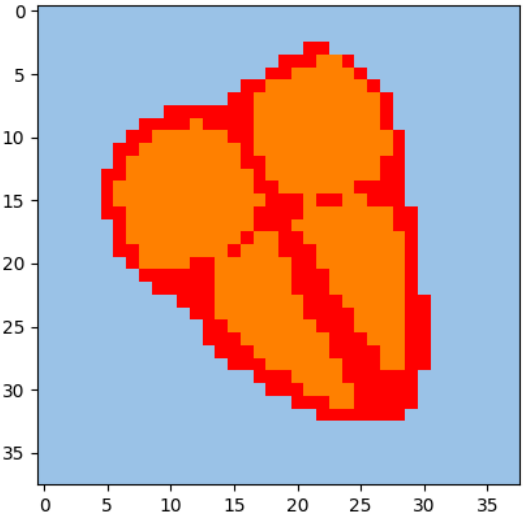



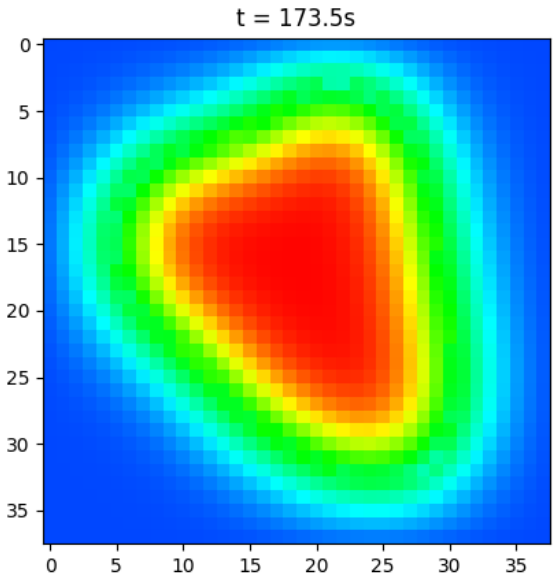



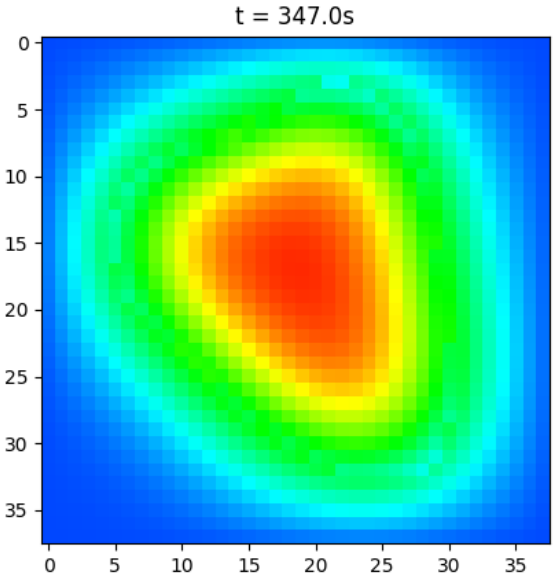



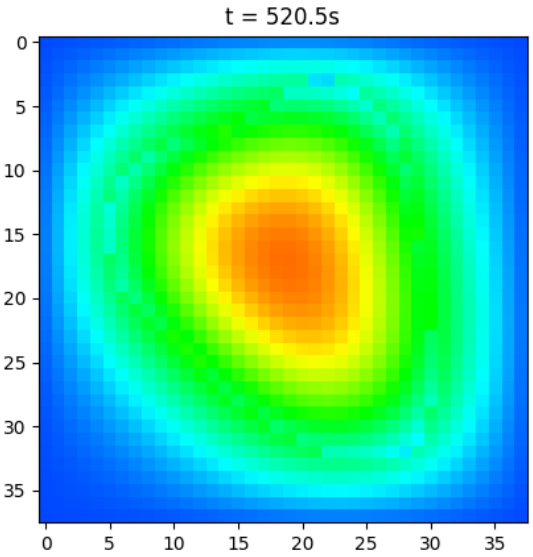
4.2.2.2. Ejemplo 2:

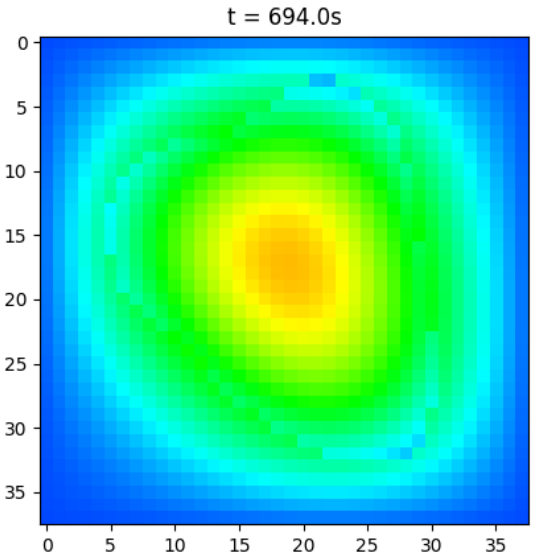
- Distancia entre nodos: h = 5mm
- Tiempo de cálculo: 27.9489s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 694s





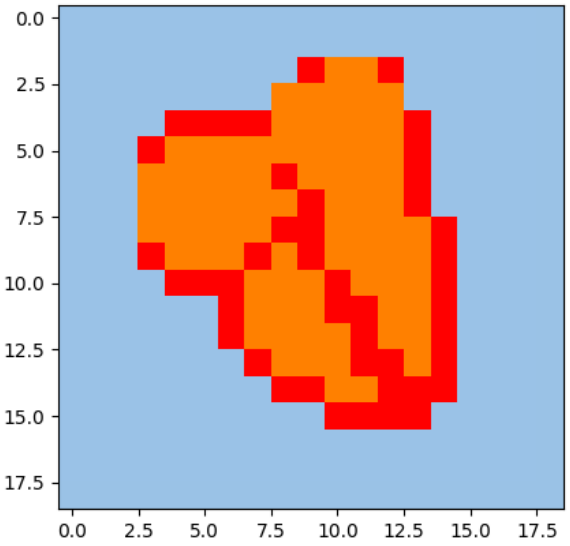



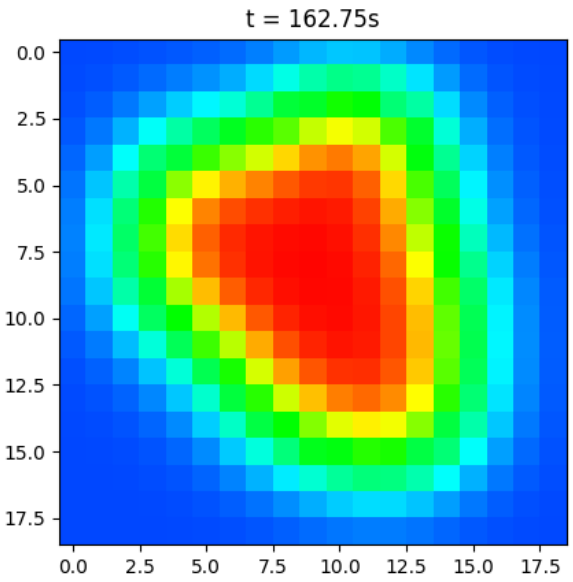



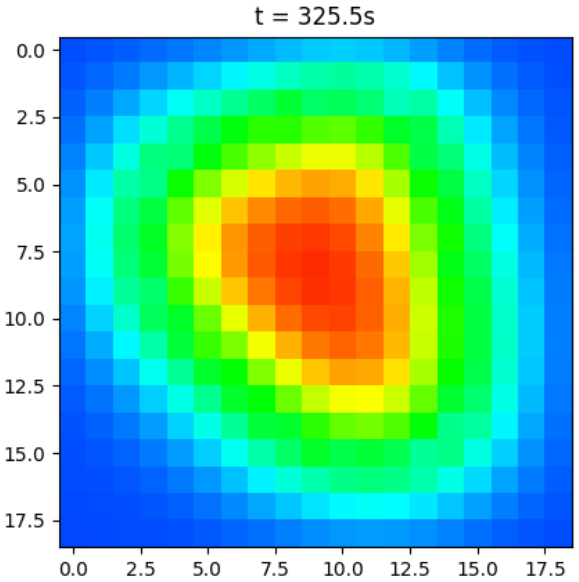



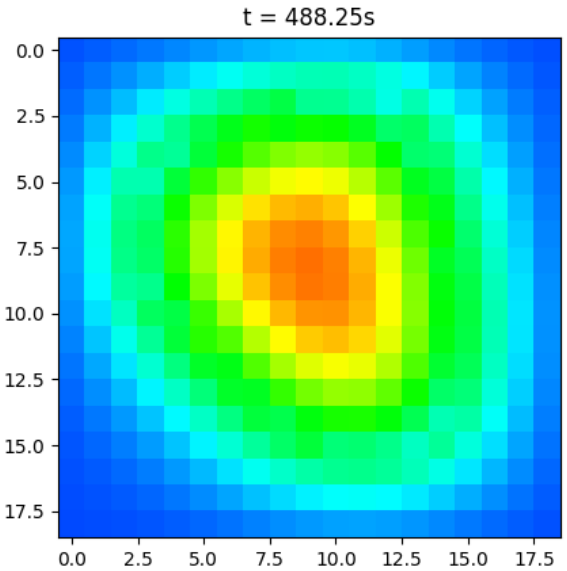
4.2.2.3. Ejemplo 3:

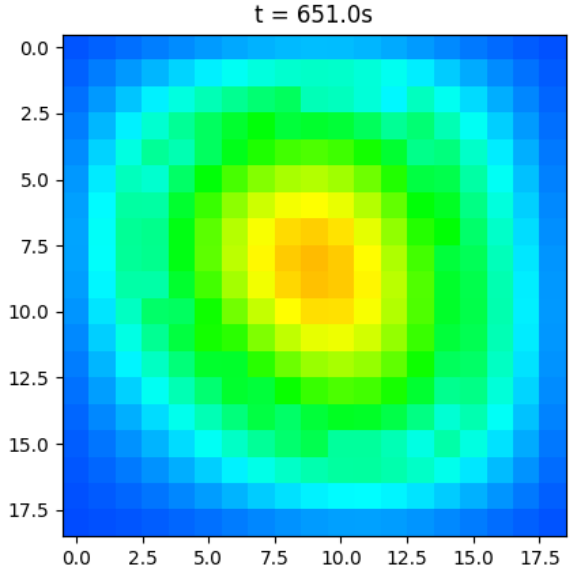
- Distancia entre nodos: h = 10mm
- Tiempo de cálculo: 5.8889s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 651s



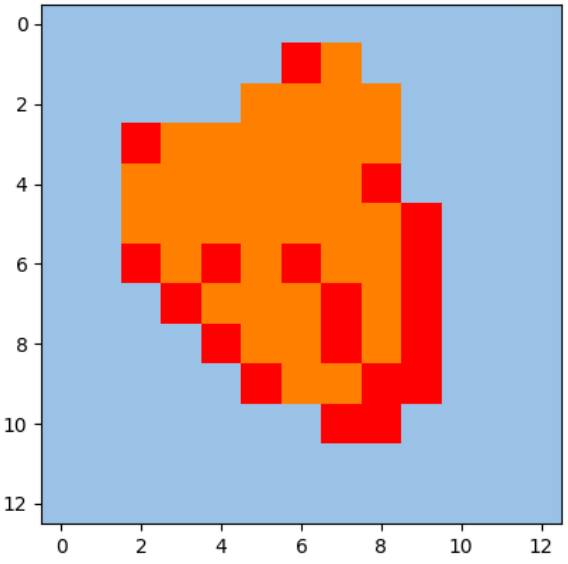



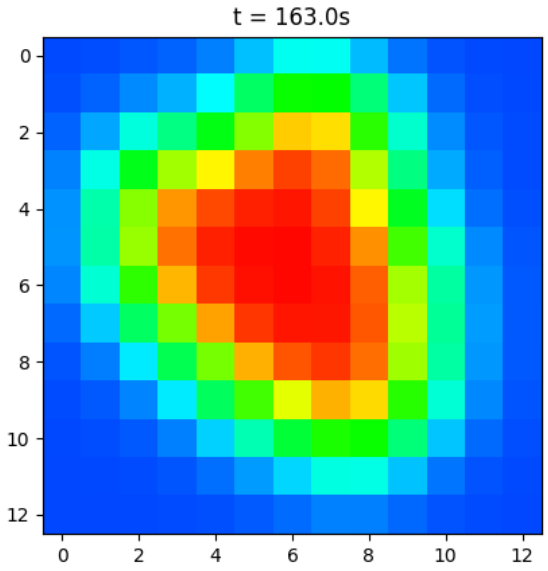



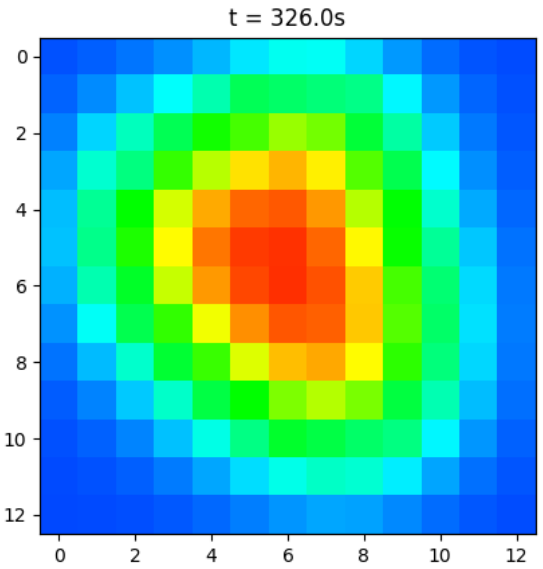



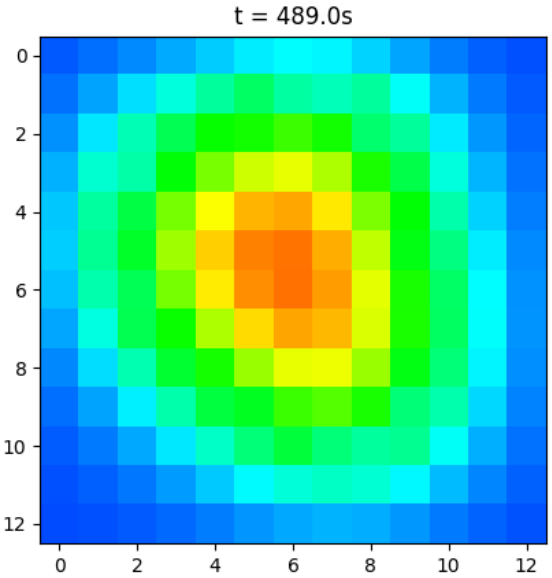
4.2.2.4. Ejemplo 4:

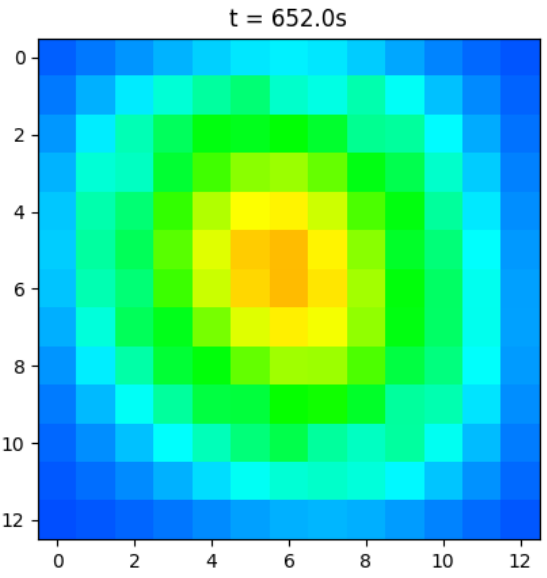
- Distancia entre nodos: h = 15mm
- Tiempo de cálculo: 1.6175s
- Tiempo de enfriamiento para llegar por debajo de 0 ºC: 652s





**4.2.3. Conclusiones:**

Se observa que más allá del tamaño de la malla o el paso en el tiempo el corazón logra refrigerarse (alcanzando 0 °C en todo su interior) en, cuánto mucho, 724s = 12min 4s. Una breve investigación en la web[2] para el caso de corazones refrigerados con nitrógeno líquido lleva a pensar que este valor puede ser razonable, puesto que en las fuentes consultadas el tiempo que se sumerge el órgano a temperaturas más bajas para congelarlo completamente varía entre 10 y 20 minutos.

[2] https://youtu.be/zLWEemhtdbE?t=48
https://youtu.be/NshDkbyQyNs?t=155

## 5. Webgrafía


- https://stackoverflow.com/questions/4455076/how-to-access-the-ith-column-of-a-numpy-multidimensional-array
- https://numpy.org/doc/stable/reference/generated/numpy.append.html
- https://stackoverflow.com/questions/18688948/numpy-how-do-i-find-total-rows-in-a-2d-array-and-total-column-in-a-1d-array
- https://matplotlib.org/stable/gallery/color/named_colors.html
- https://stackoverflow.com/questions/8885663/how-to-format-a-floating-number-to-fixed-width-in-python
- https://mortoray.com/2016/01/05/the-trouble-with-floor-and-ceil/
- https://matplotlib.org/stable/tutorials/introductory/images.html
- https://stackoverflow.com/questions/22981845/3-dimensional-array-in-numpy?rq=1
- https://opensourceoptions.com/blog/numpy-array-shapes-and-reshaping-arrays/#:%7E:text=3D%20arrays,order%20layers%2C%20rows%2C%20columns.
- https://stackoverflow.com/questions/183853/what-is-the-difference-between-and-when-used-for-division
- https://en.wikipedia.org/wiki/Heart#Structure
- https://matplotlib.org/3.1.1/api/_as_gen/matplotlib.pyplot.imshow.html
- https://stackoverflow.com/questions/13384653/imshow-extent-and-aspect
- https://numpy.org/doc/stable/reference/generated/numpy.any.html#numpy.any
- https://www.geeksforgeeks.org/division-operator-in-python/
- https://en.wikipedia.org/wiki/Thermal_diffusivity
- https://en.wikipedia.org/wiki/Heart#Structure
- https://www.youtube.com/watch?v=zLWEemhtdbE
- https://www.youtube.com/watch?v=NshDkbyQyNs
- http://www.netlib.org/lapack/explore-html/d7/d3b/group__double_g_esolve_ga5ee879032a8365897c3ba91e3dc8d512.html
- https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html