Estimación de la temperatura con la ecuacion del Bio-Calor usando DeepONet

Francisco Damián Escobar Candelaria
Yofre Hernán García Gómez

2025-04-21

Tabla de contenidos

Resumen

Aquí irá el resumen de la tesis.

1 Introducción

2 Aplicacion de Crank Nikolson

```
using DifferentialEquations, LinearAlgebra
using DataFrames, CSV

# --- PARÁMETROS FÍSICOS Y DIMENSIONALES ----------------------------------
 , c = 1050.0, 3639.0                          # densidad, calor específico
k_eff = 5.0                                    # conductividad
t_f, L = 1800.0, 0.05                          # tiempo final, longitud del dominio
c_b, Q = 3825.0, 0.0                           # coef. perfusión, fuente térmica
T_M, T_a = 45.0, 37.0                          # temperatura máxima, ambiente

# --- COEFICIENTES ADIMENSIONALES -----------------------------------------
  =   * c / k_eff
a  = t_f / (  * L^2)
a  = t_f * c_b / (  * c)
a  = (t_f * Q) / (  * c * (T_M - T_a))

# --- MALLA ESPACIAL ------------------------------------------------------
Nx, Ny = 51, 51
dx, dy = 1.0 / (Nx - 1), 1.0 / (Ny - 1)
x = range(0, 1, length=Nx)
y = range(0, 1, length=Ny)
N = Nx * Ny  # total de puntos

# --- CONDICIÓN INICIAL ---------------------------------------------------
u0 = zeros(N)

# --- SISTEMA DE EDOs DEL PDE ---------------------------------------------
function f!(du, u, _,  )
    U = reshape(u, Nx, Ny)
    D = similar(U)
    @inbounds for i in 1:Nx, j in 1:Ny
        # Derivadas segunda en x
        d2x = if i == 1
            (U[2, j] - 0) / dx^2
        elseif i == Nx
            U_ghost = U[Nx, j] +   * dx
            (U_ghost - 2U[Nx, j] + U[Nx-1, j]) / dx^2
```

```
        else
            (U[i+1, j] - 2U[i, j] + U[i-1, j]) / dx^2
        end

        # Derivadas segunda en y
        d2y = if j == 1
            (U[i, 2] - U[i, 1]) / dy^2
        elseif j == Ny
            (U[i, Ny-1] - U[i, Ny]) / dy^2
        else
            (U[i, j+1] - 2U[i, j] + U[i, j-1]) / dy^2
        end

        D[i, j] = (d2x + d2y - a  * U[i, j] + a ) / a
    end
    du .= vec(D)
end

# --- RESOLVER PDE ----------------------------------------------------------
 span = (0.0, 1.0)
prob = ODEProblem(f!, u0,  span)
taus = [0.0, 0.25, 0.5, 0.75, 1.0]
sol = solve(prob, TRBDF2(), dt=5e-4, saveat=taus)

# --- PROCESAR SOLUCIÓN EN GRILLA REDUCIDA ----------------------------------
idxs = 1:2:Nx  # índices para submuestreo
npts = length(idxs)^2 * length(taus)

# Preasignar vectores para crear el DataFrame
times = Float64[]
Xs = Float64[]
Ys = Float64[]
Thetas = Float64[]

for (k,  ) in enumerate(taus)
    θ = reshape(sol( ), Nx, Ny)
    for i in idxs, j in idxs
        push!(times,  )
        push!(Xs, x[i])
        push!(Ys, y[j])
        push!(Thetas, θ[i, j])
    end
end

df = DataFrame(time=times, X=Xs, Y=Ys, Theta=Thetas)
```

```
# --- GUARDAR CSV ------------------------------------------------------------
ruta = "data"
CSV.write(joinpath(ruta, "crank_nick.csv"), df)
```

```
using DataFrames, CSV, Plots, Statistics
pyplot()

# --- CARGAR DATOS DESDE EL CSV ----------------------------------------------
#ruta = "./Archivos"
#df = CSV.read(joinpath(ruta, "biocalor_2d.csv"), DataFrame)

# --- OBTENER VALORES ÚNICOS Y ORDENADOS DE X, Y, TIME -----------------------
x_vals = sort(unique(df.X))
y_vals = sort(unique(df.Y))
times = sort(unique(df.time))  # tiempos

Nx, Ny = length(x_vals), length(y_vals)

# --- RECONSTRUIR MATRICES 2D DE THETA PARA CADA TIEMPO ----------------------
solutions = []

for t in times
    dft = filter(:time => ==(t), df)

    # Crear matriz vacía
    Θ = fill(NaN, Nx, Ny)

    # Llenar la matriz con los valores correspondientes
    for row in eachrow(dft)
        ix = findfirst(==(row.X), x_vals)
        iy = findfirst(==(row.Y), y_vals)
        Θ[ix, iy] = row.Theta
    end

    push!(solutions, Θ)
end

# --- DETERMINAR ESCALA GLOBAL DE COLORES ------------------------------------
zmin = minimum([minimum(u) for u in solutions])
zmax = maximum([maximum(u) for u in solutions])

# --- GRAFICAR EN LAYOUT 3x2 -------------------------------------------------
p = plot(layout = (3, 2), size = (800, 900))
```

```
for (i, (t, θ)) in enumerate(zip(times, solutions))
    surface!(
        p, y_vals, x_vals, θ;  # transpuesta para que coincidan con x, y
        camera = (45,30),
        xlabel = "Y",
        ylabel = "X",
        zlabel = "T   ",
        title = "t = $(t)",
        subplot = i,
        c = :thermal,
        clim = (zmin, zmax),
        legend = false
    )
end

# Eliminar ejes y contenido del subplot 6
plot!(p[6], framestyle = :none, grid = false, xticks = false, yticks = false)
close("all")

display(p)
```

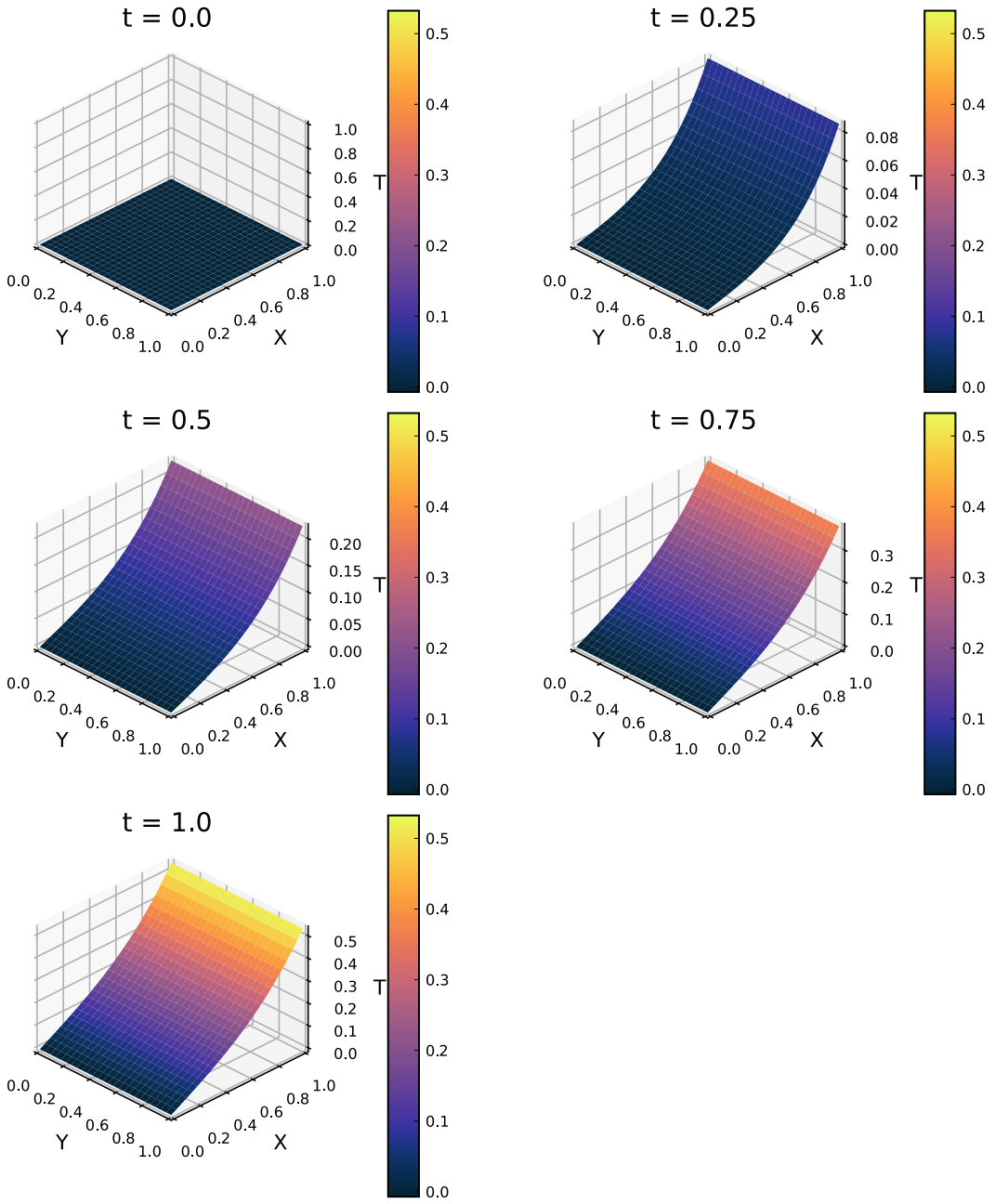

Figura 2.1: Resultados obtenidos mediante el método numérico de Crank Nickolson.

Part I

Redes neuronales

3 Physic Informed Neural Networks (PINNs)

Las Physics-Informed Neural Networks (PINNs) son un enfoque innovador que combina redes neuronales con ecuaciones diferenciales gobernantes para resolver problemas complejos de física (Blechschmidt y Ernst 2021). A diferencia de métodos tradicionales, las PINNs incorporan directamente las ecuaciones físicas en su función de pérdida mediante diferenciación automática, lo que permite minimizar simultáneamente el error en los datos y el residual de las PDEs (Karniadakis et al. 2021). Esta característica las hace particularmente valiosas en escenarios con datos limitados, donde el conocimiento físico actúa como un regularizador efectivo . La capacidad de aproximación de las PINNs se fundamenta en el teorema de aproximación universal de las redes neuronales, adaptado para incorporar restricciones físicas a través de términos de penalización en la función de optimización (Karniadakis et al. 2021).

Como ejemplo, se considera la **ecuación de Burgers para viscocidad**:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}$$

Con una condición inicial adecuada y condiciones de contorno de Dirichlet. En la figura Figura 3.1, la red izquierda (physics-uninformed) representa el sustituto de la solución de EDP $u(x,t)$, mientras que la red derecha (physics-informed) describe el residuo de EDP $\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2}$. La función de pérdida incluye una pérdida supervisada de las mediciones de datos de u de las condiciones iniciales y de contorno, y una pérdida no supervisada de EDP:

$$\mathcal{L} = w_{\text{data}}\mathcal{L}_{\text{data}} + w_{\text{PDE}}\mathcal{L}_{\text{PDE}} \tag{3.1}$$

donde:

$$\mathcal{L}_{\text{data}} = \frac{1}{N_{\text{data}}}\sum_{i=1}^{N_{\text{data}}} \left(u(x_i,t_i) - u_i\right)^2$$

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_{\text{PDE}}}\sum_{j=1}^{N_{\text{PDE}}} \left(\frac{\partial u}{\partial t}(x_j,t_j) + u\frac{\partial u}{\partial x}(x_j,t_j) - \nu\frac{\partial^2 u}{\partial x^2}(x_j,t_j)\right)^2$$

Aquí, (x_i,t_i) representan puntos donde se conocen valores de la solución y (x_j,t_j) son puntos interiores del dominio. Los pesos w_{data} y w_{PDE} equilibran la contribución de cada término. La

red se entrena minimizando $\mathcal{L}$ usando optimizadores como Adam o L-BFGS hasta alcanzar un umbral ε (Karniadakis et al. 2021).

Este enfoque permite resolver EDPs (clásicas, fraccionarias o estocásticas) sin mallas, en dominios complejos o con datos escasos y ruidosos, siendo una herramienta flexible y poderosa para la modelación científica.

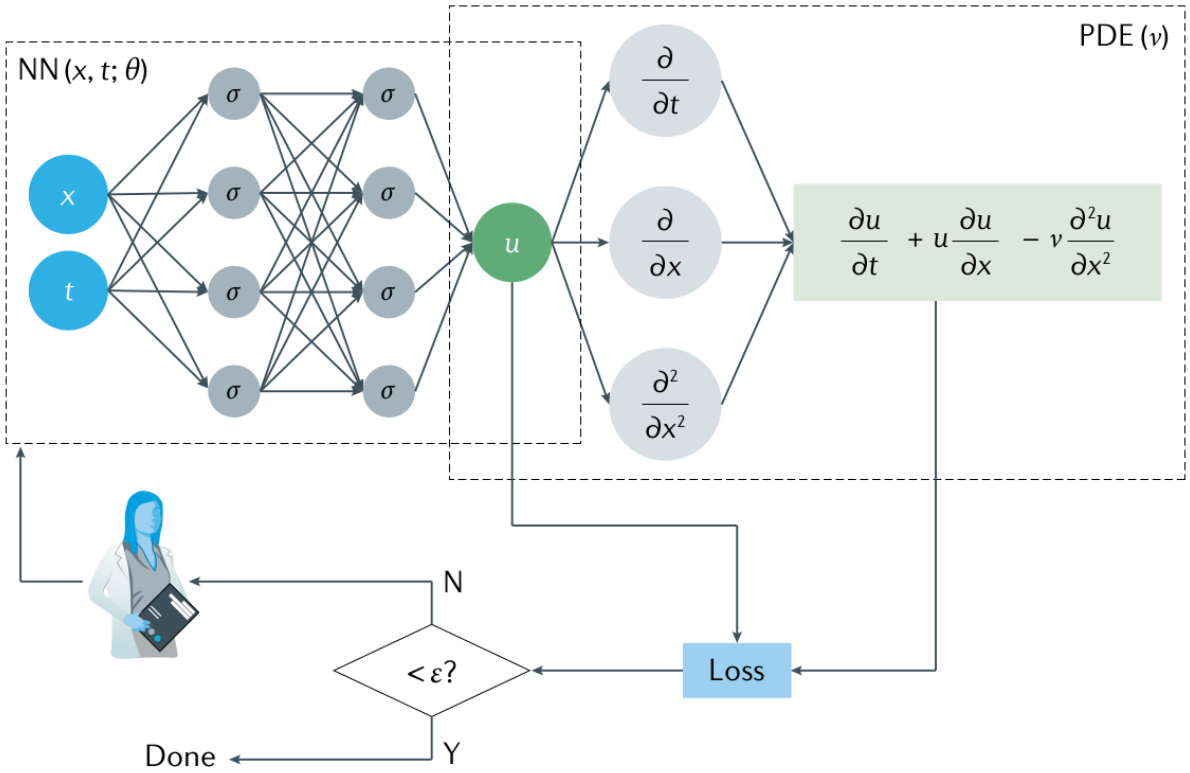

Figura 3.1: **El algoritmo de una PINN.** Construya una red neuronal (NN) $u(x,t;\theta)$ donde θ representa el conjunto de pesos entrenables w y sesgos b, y σ representa una función de activación no lineal. Especifique los datos de medición x_i, t_i, u_i para u y los puntos residuales x_j, t_j para la EDP. Especifique la pérdida $\mathcal{L}$ en la ecuación Ecuación 3.1 sumando las pérdidas ponderadas de los datos y la EDP. Entrene la NN para encontrar los mejores parámetros θ^* minimizando la pérdida $\mathcal{L}$ (Karniadakis et al. 2021).

3.1 Comparación con Redes Neuronales Tradicionales

Mientras que las redes neuronales tradicionales dependen exclusivamente de grandes volúmenes de datos etiquetados para su entrenamiento (Karniadakis et al. 2021), las PINNs integran el conocimiento físico como parte esencial de su arquitectura (Blechschmidt y Ernst 2021). Esta diferencia clave permite a las PINNs generar soluciones físicamente consistentes incluso con datos escasos, evitando el sobreajuste común en enfoques puramente basados en

datos. Otra ventaja significativa de las PINNs es su naturaleza mesh-free, que contrasta con los métodos numéricos tradicionales como FEM o FDM que requieren discretización espacial. Sin embargo, el entrenamiento de PINNs puede ser más desafiante debido a la necesidad de optimizar múltiples objetivos simultáneamente (ajuste a datos y cumplimiento de leyes físicas) (Blechschmidt y Ernst 2021; Karniadakis et al. 2021).

4 DeepONet

DeepONet (Deep Operator Network) es una arquitectura de red neuronal profunda diseñada para aprender operadores no lineales que mapean funciones de entrada a funciones de salida. A diferencia de las redes convencionales que aprenden funciones escalares, DeepONet se enfoca en representar operadores completos, como soluciones de ecuaciones diferenciales, a partir de datos observados o simulaciones numérica (Lu, Jin, et al. 2021).

4.1 Arquitectura

La arquitectura de DeepONet está compuesta por dos redes principales: la red de *branch* y la red de *trunk*. La red *branch* procesa las evaluaciones discretas de la función de entrada (por ejemplo, condiciones iniciales o de frontera), mientras que la red *trunk* recibe como entrada los puntos del dominio donde se desea evaluar la función de salida. La salida final se obtiene mediante el producto punto de los vectores generados por ambas redes, lo que permite representar operadores complejos con alta generalización a nuevos datos (Lu, Jin, et al. 2021).

4.2 Comparación con una PINN

En contraste con una red PINN convencional (Physics-Informed Neural Network), que resuelve una instancia específica de una ecuación diferencial para un conjunto dado de condiciones, DeepONet aprende el operador general que resuelve muchas instancias a la vez. Mientras que una PINN debe ser reentrenada para cada nuevo problema, DeepONet, una vez entrenado, puede predecir soluciones rápidamente para múltiples condiciones nuevas. Esto lo hace especialmente eficiente en aplicaciones donde se requiere realizar inferencias repetidas, como en control o diseño inverso (Kumar et al. 2024).

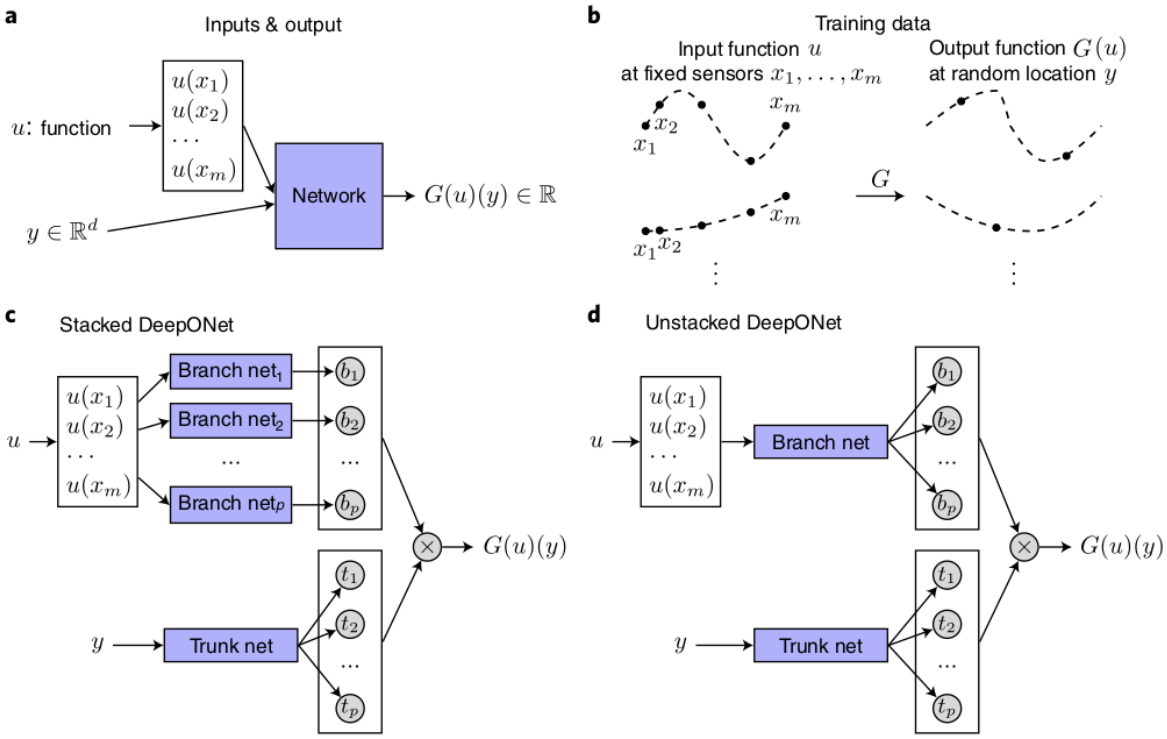

Figura 4.1: **Ilustraciones del planteamiento del problema y arquitectura DeepONet que conducen a una buena generalización. a)** Para que la red aprenda un operador $G : u \to G(u)$ se necesitan dos entradas $[u(x_1), u(x_2), ..., u(x_m)]$ e y. **b)** Ilustración de los datos de entrenamiento. Para cada función de entrada u, se requiere el mismo número de evaluaciones en los mismos sensores dispersos $x_1, x_2, ..., x_m$. Sin embargo, no se impone ninguna restricción sobre el número ni las ubicaciones para la evaluación de las funciones de salida. **c)** La DeepONet *stacked* se inspira en el **Teorema de aproximación universal para operadores** y consta de una red *Trunk* y p redes *Branch* apiladas. La red cuya construcción se inspira en el mismo teorema es una DeepONet *stacked* formada al elegir la red *Trunk* como una red de una capa de ancho p y cada red *Branch* como una red de una capa oculta de ancho n. **d)** La red DeepONet *unstacked* se inspira en el **Teorema general de aproximación universal para operadores** y consta de una red *Trunk* y una red *Branch*. Una red DeepONet *unstacked* puede considerarse como una red DeepONet *stacked*, en la que todas las redes *Branch* comparten el mismo conjunto de parámetros (Lu, Jin, et al. 2021).

Part II

Ecuación del Bio-Calor

La ecuación del bio-calor, formulada por Pennes (1948), surgió de su estudio pionero *"Analysis of Tissue and Arterial Blood Temperatures in the Resting Human Forearm"*. Publicado en el *Journal of Applied Physiology*, este trabajo fue el primero en cuantificar la interacción entre la temperatura arterial y tisular en humanos. Pennes combinó principios termodinámicos con mediciones experimentales en el antebrazo, estableciendo un modelo matemático que relacionaba el flujo sanguíneo, la producción metabólica de calor y la conducción térmica en tejidos.

Experimento

Durante su estudio, Pennes diseñó un experimento riguroso para medir la temperatura interna del antebrazo humano. Utilizó termopares tipo "Y" insertados transversalmente en la musculatura del antebrazo mediante una aguja estéril, como se ilustra en la Figura 4.2. Esta configuración permitía capturar un perfil térmico a lo largo del eje transversal, minimizando interferencias derivadas del contacto externo o la conducción axial no deseada.

La técnica experimental buscó máxima precisión geométrica y térmica: los termopares eran fijados con tensión controlada mediante un sistema mecánico que aseguraba trayectorias rectas y repetibles dentro del tejido. La inserción se realizaba con anestesia tópica mínima y bajo condiciones ambientales estables, lo cual garantizaba que los gradientes de temperatura registrados fueran atribuibles principalmente al metabolismo local y al efecto del flujo sanguíneo arterial.

Trascendencia

El modelo de Pennes simplificó la complejidad biológica al asumir un flujo sanguíneo uniforme y una transferencia de calor proporcional a la diferencia entre la temperatura arterial y la tisular. Aunque posteriores investigaciones refinaron sus supuestos, su ecuación sigue siendo un referente en bioingeniería térmica. Su trabajo no solo sentó las bases para aplicaciones clínicas, como la hipertermia oncológica, sino que también inspiró avances en el estudio de la termorregulación humana y el diseño de dispositivos médicos.

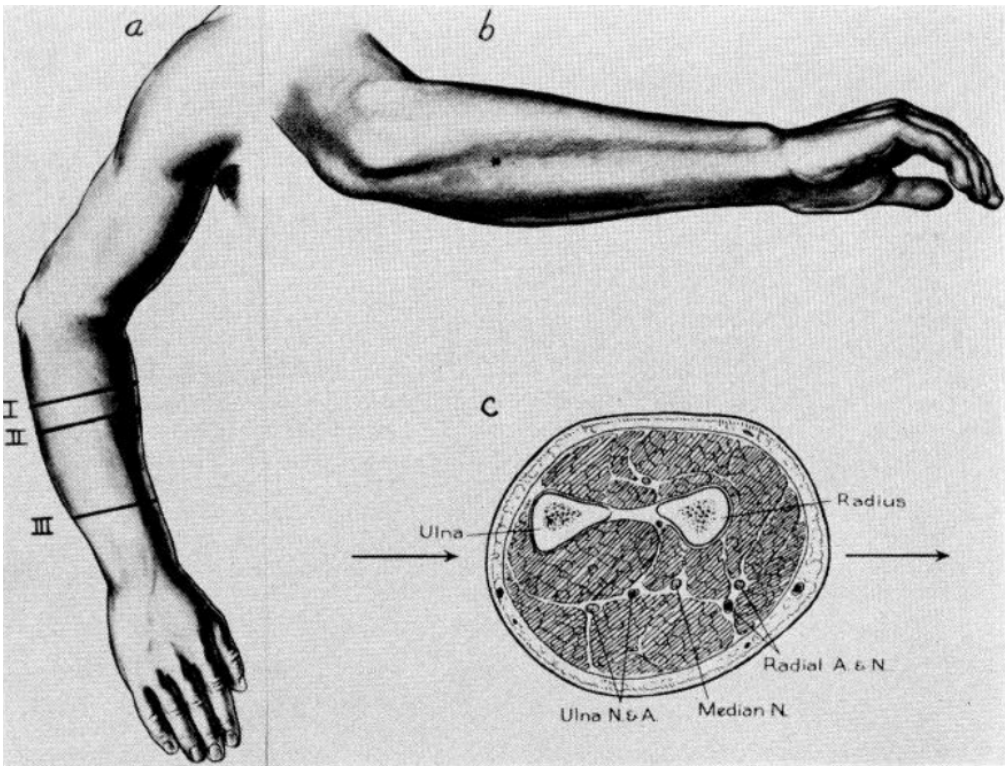

Figura 4.2: **a)** Posición del brazo derecho (vista superior). La linea horizontal II indica el nivel de la figura c). **b)** Posición del brazo derecho (vista lateral). **c)**Sección transversal anatómica del antebrazo en el nivel II (Pennes 1948).

5 Forma de la ecuación

La ecuación diferencial de bio-calor de Pennes (1948) modela la transferencia de calor en tejidos biológicos, integrando efectos de conducción, perfusión sanguínea y metabolismo. Su forma general es:

$$\rho c \frac{\partial T}{\partial t} = k_{\text{eff}} \frac{\partial^2 T}{\partial x^2} - \rho_b c_b \omega_b (T - T_a) + Q, \quad x \in \Omega,\, t \in [0, t_f] \tag{5.1}$$

Tabla 5.1: Tabla de nomenclaturas de la Ecuación 5.1.

Símbolo	Descripción	Unidades
T	Temperatura del tejido	°C
ρ	Densidad del tejido	$\frac{kg}{m^3}$
c	Calor específico del tejido	$\frac{J}{kg°C}$
k_{eff}	Conductividad térmica	$\frac{W}{m°C}$
ρ_b	Densidad de la sangre	$\frac{kg}{m^3}$
c_b	Calor específico de la sangre	$\frac{J}{kg°C}$
ω_b	Tasa de perfusión sanguínea	$1/s$
T_a	Temperatura arterial	°C
$Q = q_m + q_p$	Fuente de calor	$\frac{W}{m^3}$
q_m	Metabolismo	$\frac{W}{m^3}$
q_p	Externa	$\frac{W}{m^3}$

5.1 Versión reducida (adimensionalizada)

Mediante escalamiento:

$$T' = T - T_a \qquad \theta = \frac{T'}{T_M - T_a} \qquad X = \frac{x}{L_0} \qquad \tau = \frac{t}{t_f}$$

Tabla 5.2: Tabla de nomenclatura de las relaciones para escalamiento.

Símbolo	Descripción	Unidades
L_0	Longitud característica del dominio	m
t_f	Tiempo final de simulación	s

la Ecuación 5.1 se convierte en:

$$\partial_\tau \theta = a_1 \partial_{XX} \theta - a_2 W \theta + a_3 \tag{5.2}$$

Parámetros adimensionales:
- $a_1 = \frac{t_f}{\alpha L_0^2}$ (difusividad térmica $\alpha = \frac{k_{\text{eff}}}{\rho c}$).
- $a_2 = \frac{t_f c_b}{\rho c}$.
- $a_3 = \frac{t_f Q}{\rho c (T_M - T_a)}$.
- $W = \rho_b \omega_b$: Tasa volumétrica de perfusión ($\text{kg/m}^3 \cdot \text{s}$).

5.2 Condiciones de uso adecuadas

1. **Tejidos homogéneos**: Aproximación válida para regiones con propiedades térmicas uniformes.

2. **Perfusión sanguínea constante**: Supone flujo sanguíneo estable en el dominio.

3. **Aplicaciones clínicas**: Hipertermia, crioterapia y modelado térmico en terapias oncológicas.

6 Modelado del Bio-Calor en Hipertermia

La ecuación del bio-calor, formulada por Pennes (1948), permite modelar la distribución de temperatura en tejidos biológicos considerando el flujo sanguíneo, la conductividad térmica y fuentes internas o externas de calor. Su modelación es fundamental en la física médica para predecir el comportamiento térmico durante tratamientos como la hipertermia. Además, constituye una herramienta poderosa en el desarrollo de simulaciones computacionales aplicadas al diseño y control de terapias térmicas en tejidos vivos.

6.1 Aplicaciones recientes de la ecuación del bio-calor

Quintero et al. (2017) desarrollan un modelo basado en ecuaciones diferenciales parciales que integra la ecuación del bio-calor y la ley de Arrhenius para estimar el daño térmico en tratamientos de hipertermia superficial. Utilizan el método de líneas para resolver el sistema y plantean un problema de optimización que busca maximizar el daño al tejido tumoral minimizando el daño colateral. Su trabajo demuestra cómo la modelación matemática puede guiar estrategias terapéuticas más seguras y eficaces.

Dutta y Rangarajan (2018) presentan una solución analítica cerrada en dos dimensiones para la ecuación del bio-calor, considerando modelos de conducción tanto de tipo Fourier como no-Fourier. Mediante el uso de la transformada de Laplace, analizan la influencia de parámetros fisiológicos como la perfusión sanguínea y el tiempo de relajación térmica sobre la evolución de la temperatura. Su investigación aporta una base teórica sólida para comprender la propagación térmica en tejidos vivos durante la hipertermia terapéutica.

Yang et al. (2014) propone una estrategia numérica para resolver problemas inversos de conducción térmica en tejidos biológicos multicapa, utilizando un enfoque en diferencias finitas y el concepto de tiempo futuro. El estudio se enfoca en predecir las condiciones de frontera necesarias para generar distribuciones de temperatura deseadas. La implementación de este método permite estimar parámetros relevantes en tiempo real, lo cual resulta esencial para el control térmico preciso en procedimientos médicos no invasivos como la hipertermia localizada.

Part III

Estudio de caso

Hipertermia como opción terapéutica complementaria en el manejo de cáncer

La Organización Mundial de la Salud (2022) en su página web define Cáncer como:

«Cáncer» es un término genérico utilizado para designar un amplio grupo de enfermedades que pueden afectar a cualquier parte del organismo; también se habla de «tumores malignos» o «neoplasias malignas». Una característica definitoria del cáncer es la multiplicación rápida de células anormales que se extienden más allá de sus límites habituales y pueden invadir partes adyacentes del cuerpo o propagarse a otros órganos, en un proceso que se denomina «metástasis». La extensión de las metástasis es la principal causa de muerte por la enfermedad.

Por su parte Instituto Nacional del Cáncer (2021) aporta lo siguiente:

Es posible que el cáncer comience en cualquier parte del cuerpo humano, formado por billones de células. En condiciones normales, las células humanas se forman y se multiplican (mediante un proceso que se llama división celular) para formar células nuevas a medida que el cuerpo las necesita. Cuando las células envejecen o se dañan, mueren y las células nuevas las reemplazan. A veces el proceso no sigue este orden y las células anormales o células dañadas se forman y se multiplican cuando no deberían. Estas células tal vez formen tumores, que son bultos de tejido. Los tumores son cancerosos (malignos) o no cancerosos (benignos).

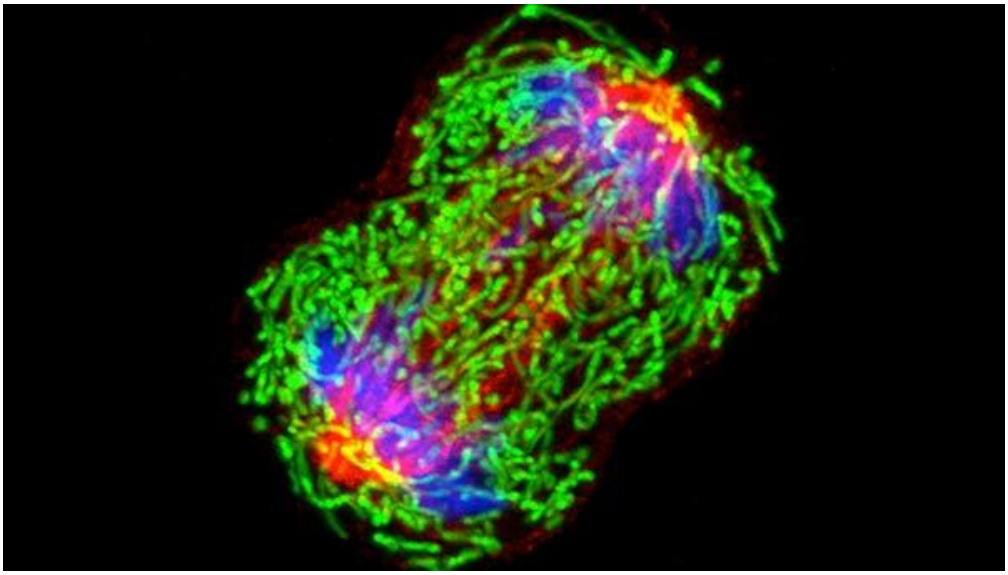

Figura 6.1: Una célula de cáncer de seno que se multiplica (Instituto Nacional del Cáncer 2021).

Ésta enfermedad es la principal causa de muerte a nivel mundial, solo en 2020 arrebató casi 10 millones de vidas y según datos de Organización Mundial de la Salud (2022) los cánceres más comunes en 2020 fueron:

- De mama (2.26 millones de casos)
- De pulmón (2.21 millones de casos)
- De colon (1.93 millones de casos)
- De próstata (1.41 millones de casos)
- De piel (distinto del melanoma) (1.20 millones de casos)
- Gástrico (1.09 millones de casos)

Es ante este panorama que distintos tratamientos surgen con el objetivo de erradicar la enfermedad siempre que se tenga una detección oportuna. Uno de dichos tratamientos es la hipertermia, según en el National Cancer Institute (2021) es un método que consiste en calentar el tejido corporal hasta los 39-45 °C para ayudar a erradicar células cancerígenas con pequeñas o nulas lesiones en el tejido sano. La hipertermia también es llamada terapia térmica o termoterapia.

Uno de los principales retos de este tratamiento es la creación de un modelo óptimo que se adecue al comportamiento de la transferencia de calor que se hace a los tejidos con el fin de dañar únicamente el área en el que se encuentran las célular cancerígenas, es por ello que los modelos de integencia artificial y más precisamente las PINN's (aqui irá una cita) surgen como posible solución a este reto.

El presente estudio utilizó como punto de partida el trabajo realizado por Alessio Borgi (2023) para modelar el calentamiento del tejido corporal usando la ecuación del Bio-Calor en dos dimensiones.

7 Metodología

En esta sección se describe el enfoque metodológico utilizado para evaluar la efectividad de una PINN utilizando una arquitectura DeepONet con el objetivo de resolver la ecuacion del Bio-Calor. El proceso metodológico se divide en las siguientes etapas:

7.1 Aportaciones del modelo

Ya que se parte del trabajo de Alessio Borgi (2023), se examinó que dos de los puntos a mejorar de la red neuronal que plantearon son:

1. Desarrollar nuevas arquitecturas para la red neuronal y explorar nuevas configuraciones
2. Combinar las fortalezas de los algoritmos de optimización Adam y L-BFGS para mejorar la velocidad de convergencia y la precisión

Tenindo los anteriores puntos en cuenta, se procedió a abordarlos e implementarlos dentro del diseño del modelo.

7.2 Diseño del modelo

El lenguaje seleccionado fué Python, a su vez el código se basa enteramente en la librería Deepxde creada por Lu, Meng, et al. (2021) la cual está directamente enfocada a resolver ecuaciones diferenciales, se usó además como backend *tensorflow_compat_v1* siendo su elección debida únicamente a la familiarización previa que se tenía con ella. Finalmente el entorno donde se programó y optimizó el código fué en *Google Colab* ya que la potencia de cómputo ofrecida por la plataforma era necesaria para ejecutar el modelo.

7.3 Implementación del modelo

Una vez creado el código que resuelve la ecuación del Bio-Calor, se ajustaron los hiperparámetros tales como cantidad de épocas de entrenamiento, el ratio de aprendizaje, la función de activación y el inicializador en base al trabajo de Alessio Borgi (2023).

7.4 Evaluación del modelo

Se llevó a cabo una evaluación del modelo al darle como entrada un conjunto de datos que no había visto y posteriormente obtener como salida sus predicciones, con ellas se elaboraron gráficas claras y detalladas de su pronóstico en el intervalo de tiempo y espacio especificados.

7.5 Comparación de resultados

Los resultados obtenidos de la evaluación del modelo fueron comparados con los del trabajo de Alessio Borgi (2023) para determinar su eficacia predictiva relativa. Se analizaron las fortalezas y debilidades del modelo en función de su desempeño en la predicción de las variables de interés.

7.6 Análisis y conclusión

Finalmente, se realizó un análisis detallado de los resultados obtenidos para extraer conclusiones significativas. Se proporcionaron recomendaciones basadas en los hallazgos del estudio, lo que permitió establecer un marco para interpretaciones analíticas profundas y recomendaciones bien fundamentadas en la sección de conclusiones del estudio.

Este enfoque metodológico proporcionó una base sólida para los resultados obtenidos, asegurando la integridad y la calidad del análisis realizado en el estudio.

8 Predicciones del modelo

Conforme se ha referido previamente, se creó el modelo utilizando Deepxde como base. Resulta relevante descatacar que se empleó la versión 1.10.1 de dicha librería. A continuación se presenta el código fuente de la red neuronal

```
import deepxde as dde
import numpy as np
import tensorflow as tf

# ------------------------------------------------------------------------------
# Constants and Parameters
# ------------------------------------------------------------------------------

# Backend and seed
dde.backend.set_default_backend("tensorflow.compat.v1")
dde.config.set_random_seed(123)

# Physical parameters
p = 1050
c = 3639
keff = 5
tf = 1800
L0 = 0.05
cb = 3825
Q = 0
TM = 45
Ta = 37
alpha = p * c / keff

# Dimensionless coefficients
a1 = tf / (alpha * L0**2)
a2 = tf * cb / (p * c)
a3 = (tf * Q) / (p * c * (TM - Ta))

# Domain boundaries
x_initial, x_boundary = 0.0, 1.0
y_initial, y_boundary = 0.0, 1.0
t_initial, t_final = 0.0, 1.0
```

```
# Dataset configuration
pts_dom = 10
pts_bc = 20
pts_ic = 60
num_test = 25

# Sensor grid and function space
num_sensors = 4
size_cov_matrix = 40

# Network architecture
width_net = 20
len_net = 3
AF = "elu"
k_initializer = "Glorot normal"

# Training parameters
num_iterations = 1000
learning_rate = 2e-3
decay_rate = 0.05
decay_steps = 1000

# ---------------------------------------------------------------------------
# Geometry and Time Domain
# ---------------------------------------------------------------------------

spatial_domain = dde.geometry.Rectangle([x_initial, y_initial],
                                        [x_boundary, y_boundary])
time_domain = dde.geometry.TimeDomain(t_initial, t_final)
geomtime = dde.geometry.GeometryXTime(spatial_domain, time_domain)

# ---------------------------------------------------------------------------
# PDE and Conditions
# ---------------------------------------------------------------------------

def initial_condition(X):
    return 0

def heat_equation(func, u, coords):
    u_t = dde.grad.jacobian(u, func, i=0, j=2)
    u_xx = dde.grad.hessian(u, func, i=0, j=0)
    u_yy = dde.grad.hessian(u, func, i=1, j=1)
    return a1 * u_t - (u_xx + u_yy) + a2 * u

def zero_value(X):
```

```
    return 0

def time_value(X):
    return X[:, 2]

def is_on_vertex(x):
    vertices = np.array([[x_initial, y_initial],
                         [x_boundary, y_initial],
                         [x_initial, y_boundary],
                         [x_boundary, y_boundary]])
    return any(np.allclose(x, v) for v in vertices)

def is_initial(X, on_initial):
    return on_initial and np.isclose(X[2], t_initial)

def left_boundary(X, on_boundary):
    spatial = X[0:2]
    t = X[2]
    return (
        on_boundary
        and np.isclose(spatial[0], x_initial)
        and not np.isclose(t, t_initial)
        and not is_on_vertex(spatial)
    )

def right_boundary(X, on_boundary):
    spatial = X[0:2]
    t = X[2]
    return (
        on_boundary
        and np.isclose(spatial[0], x_boundary)
        and not np.isclose(t, t_initial)
        and not is_on_vertex(spatial)
    )

def up_low_boundary(X, on_boundary):
    spatial = X[0:2]
    t = X[2]
    return (on_boundary
    and (np.isclose(spatial[1], y_initial)
    or np.isclose(spatial[1], y_boundary))
    and not np.isclose(t, t_initial)
    and not is_on_vertex(spatial)
    )
```

```
# Initial and boundary conditions
ic = dde.icbc.IC(geomtime, initial_condition, is_initial)
left_bc = dde.icbc.DirichletBC(geomtime,
                               zero_value, left_boundary)
right_bc = dde.icbc.NeumannBC(geomtime,
                               time_value, right_boundary)
up_low_bc = dde.icbc.NeumannBC(geomtime,
                               zero_value, up_low_boundary)

# ------------------------------------------------------------------------
# Dataset Construction
# ------------------------------------------------------------------------

pde_data = dde.data.TimePDE(
    geomtime,
    heat_equation,
    [ic, left_bc, right_bc, up_low_bc],
    num_domain=pts_dom,
    num_boundary=pts_bc,
    num_initial=pts_ic
)

# ------------------------------------------------------------------------
# Sensor Points and Function Space
# ------------------------------------------------------------------------

side = np.linspace(x_initial, x_boundary, num_sensors + 1)
x, y = np.meshgrid(side, side, indexing='xy')
sensor_pts = np.stack([x.ravel(), y.ravel()], axis=1)

fs = dde.data.function_spaces.GRF2D(N=size_cov_matrix,
                                    interp="linear")

data = dde.data.PDEOperatorCartesianProd(
    pde_data,
    fs,
    sensor_pts,
    num_function=(num_sensors + 1)**2,
    function_variables=[0, 1],
    num_test=num_test
)

# ------------------------------------------------------------------------
# Network Definition
# ------------------------------------------------------------------------
```

```
branch_layers = [(num_sensors + 1)**2] + len_net * [width_net]
trunk_layers = [3] + len_net * [width_net]

net = dde.nn.DeepONetCartesianProd(
    branch_layers,
    trunk_layers,
    activation=AF,
    kernel_initializer=k_initializer
)

# ------------------------------------------------------------------------------
# Model Compilation and Training
# ------------------------------------------------------------------------------

model = dde.Model(data, net)
model.compile("adam", lr=learning_rate, decay=("inverse time", decay_steps, decay_rate))
losshistory, train_state = model.train(iterations=num_iterations)

# Fine-tuning with LBFGS optimizer
model.compile("L-BFGS")
losshistory, train_state = model.train()
```

```
2025-05-29 23:27:39.973609: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda d
2025-05-29 23:27:40.054615: I tensorflow/tsl/cuda/cudart_stub.cc:28] Could not find cuda d
2025-05-29 23:27:40.056300: I tensorflow/core/platform/cpu_feature_guard.cc:182] This Tens
critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow wi
2025-05-29 23:27:41.442839: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-
TRT Warning: Could not find TensorRT
Using backend: tensorflow.compat.v1
Other supported backends: tensorflow, pytorch, jax, paddle.
paddle supports more examples now and is recommended.

WARNING:tensorflow:From /home/damian/.local/lib/python3.8/site-packages/tensorflow/python/
Instructions for updating:
non-resource variables are not supported in the long term
Setting the default backend to "tensorflow.compat.v1". You can change it in the ~/.deepxde
Compiling model...
Building DeepONetCartesianProd...
'build' took 0.116369 s

/home/damian/.local/lib/python3.8/site-packages/deepxde/nn/tensorflow_compat_v1/deeponet.p

`tf.layers.dense` is deprecated and will be removed in a future version. Please use `tf.ke
```

```
/home/damian/.local/lib/python3.8/site-packages/deepxde/nn/tensorflow_compat_v1/deeponet.p

`tf.layers.dense` is deprecated and will be removed in a future version. Please use `tf.ke

/home/damian/.local/lib/python3.8/site-packages/deepxde/nn/tensorflow_compat_v1/deeponet.p

`tf.layers.dense` is deprecated and will be removed in a future version. Please use `tf.ke

'compile' took 18.901549 s

2025-05-29 23:28:02.787394: I tensorflow/compiler/mlir/mlir_graph_optimization_pass.cc:375

Training model...

Step      Train loss                                              Test loss
0         [2.57e+00, 4.03e-01, 4.13e-01, 6.71e-01, 1.05e+00]    [1.57e+00, 1.57e-
01, 3.45e-01, 5.19e-01, 3.07e-01]    []
1000      [3.97e-03, 3.15e-03, 3.81e-04, 2.63e-02, 1.86e-04]    [6.39e-
03, 3.24e-03, 9.14e-04, 2.74e-02, 4.08e-04]    []

Best model at step 1000:
  train loss: 3.40e-02
  test loss: 3.84e-02
  test metric: []

'train' took 21.333825 s

Compiling model...
'compile' took 36.678331 s

Training model...

Step      Train loss                                              Test loss
1000      [3.97e-03, 3.15e-03, 3.81e-04, 2.63e-02, 1.86e-04]    [6.39e-
03, 3.24e-03, 9.14e-04, 2.74e-02, 4.08e-04]    []
INFO:tensorflow:Optimization terminated with:
  Message: CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
  Objective function value: 0.033171
  Number of iterations: 5
  Number of functions evaluations: 65
1065      [3.14e-03, 3.15e-03, 3.80e-04, 2.63e-02, 1.87e-04]    [5.62e-
03, 3.23e-03, 9.08e-04, 2.74e-02, 4.08e-04]    []

Best model at step 1065:
```

```
  train loss: 3.32e-02
  test loss: 3.76e-02
  test metric: []

'train' took 12.093573 s
```

El historial de perdida para el conjunto de entrenamiento es el siguiente:

```python
import plotly.graph_objects as go

# Nombres de las componentes del loss
loss_labels = [
    "PDE residual loss",
    "Initial-condition loss",
    "Left-boundary (Dirichlet) loss",
    "Right-boundary (Neumann) loss",
    "Top/Bottom-boundary (Neumann) loss"
]

# Extraer pasos y pérdida de entrenamiento
steps = losshistory.steps
train_loss = np.array(losshistory.loss_train)

# Crear figura
fig_train = go.Figure()

for i in range(train_loss.shape[1]):
    fig_train.add_trace(go.Scatter(
        x=steps,
        y=train_loss[:, i],
        mode='lines',
        name=loss_labels[i]
    ))

fig_train.update_layout(
    title="Training Loss history",
    xaxis=dict(title="Iteration", tickformat=".1e"),
    yaxis=dict(title="Loss", type="log", tickformat=".1e"),
    template="plotly_white",
    legend=dict(x=0.99, y=0.99),
    font=dict(size=14)
)
fig_train.show()
```

El historial de perdida para el conjunto de prueba es el siguiente:

Figura 8.1: Gráfica de la perdida en el entrenamiento.

```
import plotly.graph_objects as go

# Nombres de las componentes del loss
loss_labels = [
    "PDE residual loss",
    "Initial-condition loss",
    "Left-boundary (Dirichlet) loss",
    "Right-boundary (Neumann) loss",
    "Top/Bottom-boundary (Neumann) loss"
]

# Extraer pasos y pérdida de entrenamiento
steps = losshistory.steps
test_loss = np.array(losshistory.loss_test)

# Crear figura
fig_test = go.Figure()

for i in range(test_loss.shape[1]):
    fig_test.add_trace(go.Scatter(
        x=steps,
        y=test_loss[:, i],
        mode='lines',
        name=loss_labels[i]
    ))

fig_test.update_layout(
    title="Test Loss history",
    xaxis=dict(title="Iteration", tickformat=".1e"),
    yaxis=dict(title="Loss", type="log", tickformat=".1e"),
    template="plotly_white",
    legend=dict(x=0.99, y=0.99),
    font=dict(size=14)
)
```

Para guardar los datos y porteriormente graficar, se uso el modelo para predecir los valores en el cuadrado de $[0,1] \times [0,1]$ dividiendo sus lados en 26 segementos equiespaciados a los tiempos $t = [0.0, 0.25, 0.50, 0.75, 1.0]$.

```
import pandas as pd
# Lista de tiempos
times = [0.0, 0.25, 0.5, 0.75, 1.0]

# Crear la malla (x, y)
num_points = 26
```

Figura 8.2: Gráfica de la perdida en el conjunto de prueba.

```
x = np.linspace(0, 1, num_points)
y = np.linspace(0, 1, num_points)
X, Y = np.meshgrid(x, y)

# Lista para almacenar resultados
results = []

for t_val in times:
    # Crear entrada trunk: (num_points^2, 3)
    points = np.vstack((X.flatten(), Y.flatten(), t_val * np.ones_like(X.flatten()))).T

    # Crear entrada branch: condición inicial constante cero
    branch_input = np.zeros((1, sensor_pts.shape[0]))

    # Predecir
    predicted = model.predict((branch_input, points)).flatten()

    # Agregar los datos al resultado
    for xi, yi, thetai in zip(points[:, 0], points[:, 1], predicted):
        results.append([t_val, xi, yi, thetai])

# Crear el DataFrame
df = pd.DataFrame(results, columns=["time", "X", "Y", "Theta"])

# Obtener la ruta del script actual y guardar el archivo CSV
ruta = r"data/model_DoN.csv"
df.to_csv(ruta, index=False)
```

A continuación, los valores predichos por la red neuronal a tiempos antes mencionados.

```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.gridspec as gridspec

# Cargar el DataFrame desde CSV
#df = pd.read_csv("predicciones_biocalor.csv")

# Lista de tiempos
#times = [0.0, 0.25, 0.5, 0.75, 1.0]

# Crear una figura y layout con GridSpec
ncols = len(times)
fig = plt.figure(figsize=((5 * ncols) + 1, 6))
gs = gridspec.GridSpec(2, ncols, height_ratios=[10, 1], hspace=0.3)
```

```
# Lista para almacenar los objetos surface
surf_list = []

# Asumimos que el grid es regular, así que podemos inferir la forma
num_points = int(np.sqrt(df[df["time"] == times[0]].shape[0]))

# Reordenar para graficar
for i, t_val in enumerate(times):
    # Filtrar por tiempo actual
    df_t = df[df["time"] == t_val]

    # Obtener los valores de X, Y, Theta
    X_vals = df_t["X"].values.reshape((num_points, num_points))
    Y_vals = df_t["Y"].values.reshape((num_points, num_points))
    Z_vals = df_t["Theta"].values.reshape((num_points, num_points))

    # Subgráfico 3D
    ax = fig.add_subplot(gs[0, i], projection="3d")

    # Dibujar la superficie
    surf = ax.plot_surface(
        Y_vals, X_vals, Z_vals,
        rstride=1, cstride=1,
        cmap="viridis",
        edgecolor="none",
        antialiased=True
    )
    surf_list.append(surf)

    ax.set_title(f"Time = {t_val:.2f} s")
    ax.set_xlabel("Y")
    ax.set_ylabel("X")
    ax.set_zlabel("T[K]", labelpad=5, rotation=90)

# Barra de color común
cbar_ax = fig.add_subplot(gs[1, :])
fig.colorbar(surf_list[-1], cax=cbar_ax, orientation="horizontal")

plt.show()
```

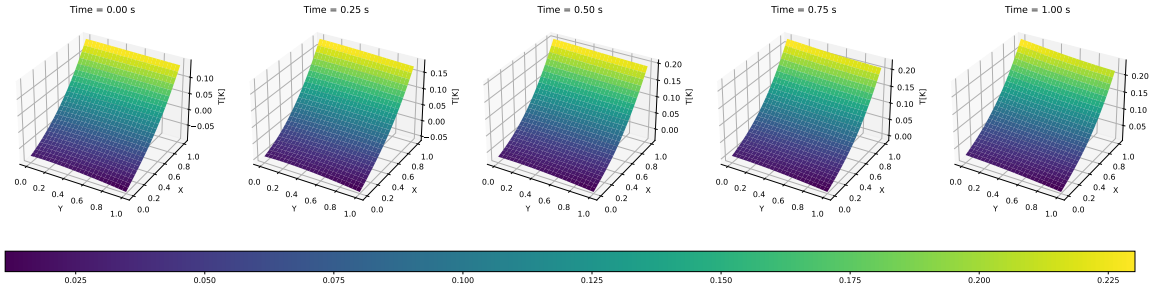

Figura 8.3: Predicciones de la red neuronal a distintos tiempos.

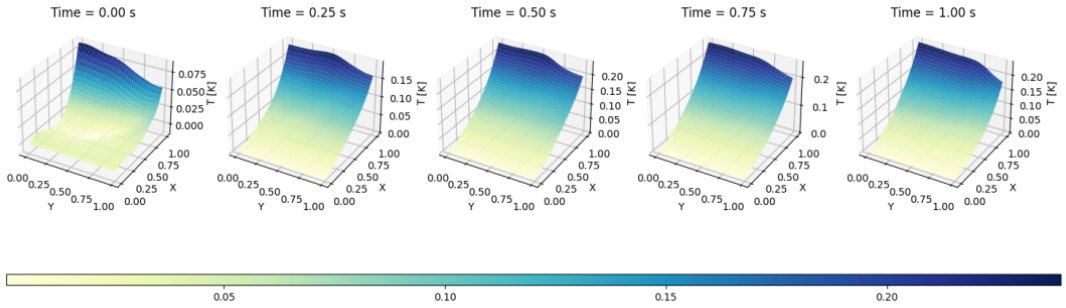

Figura 8.4: Resultados reportados por Alessio Borgi (2023) en el caso 2D.

8.1 Comparación con el método de Crank Nickolson

Para tener un valor numérico con el cual conocer al veracidad del modelo se comparó con las predicciones del método de Crank Nickolson

```
import os

# Definir los tiempos a evaluar
times = [0.0, 0.25, 0.5, 0.75, 1.0]

# Rutas de los archivos (modificar según sea necesario)
crank_nick_path = "data/crank_nick.csv"
model_don_path = "data/model_DoN.csv"
output_path = "data/error_comparison.csv"

# Cargar los datos
def load_data(file_path):
    try:
        return pd.read_csv(file_path)
    except FileNotFoundError:
```

```
        print(f"Error: No se encontró el archivo {file_path}")
        return None
    except Exception as e:
        print(f"Error al cargar {file_path}: {str(e)}")
        return None

crank_nick_data = load_data(crank_nick_path)
model_don_data = load_data(model_don_path)

if crank_nick_data is None or model_don_data is None:
    exit()

# Función para calcular errores
def calculate_errors(true_data, pred_data, times):
    results = []

    for time in times:
        # Filtrar datos por tiempo
        true_subset = true_data[true_data['time'] == time]
        pred_subset = pred_data[pred_data['time'] == time]

        if len(true_subset) == 0 or len(pred_subset) == 0:
            print(f"Advertencia: No hay datos para tiempo t={time}")
            continue

        # Verificar que las dimensiones coincidan
        if len(true_subset) != len(pred_subset):
            print(f"Advertencia: Número de puntos no coincide para t={time}")
            min_len = min(len(true_subset), len(pred_subset))
            true_subset = true_subset.iloc[:min_len]
            pred_subset = pred_subset.iloc[:min_len]

        # Calcular errores para Theta
        theta_true = true_subset['Theta'].values
        theta_pred = pred_subset['Theta'].values

        absolute_error = np.abs(theta_true - theta_pred)
        l2_error = np.sqrt(np.sum((theta_true - theta_pred)**2))

        results.append({
            'time': time,
            'mean_absolute_error': np.mean(absolute_error),
            'max_absolute_error': np.max(absolute_error),
            'l2_error': l2_error,
            'n_points': len(true_subset)
```

Tabla 8.1: Errores de la red neuronal.

time	mean_absolute_error	max_absolute_error	l2_error	n_points
0	0.0574679	0.147267	1.77741	676
0.25	0.0602457	0.182125	1.93128	676
0.5	0.0769529	0.25433	2.50529	676
0.75	0.114009	0.385995	3.78112	676
1	0.161816	0.523789	5.3985	676

```
        })

    return pd.DataFrame(results)

# Calcular errores
error_results = calculate_errors(crank_nick_data, model_don_data, times)

# Guardar resultados
error_results.to_csv(output_path, index=False)
```

Referencias

Alessio Borgi, Alessandro De Luca, Eugenio Bugli. 2023. «BioHeat PINNs: Temperature Estimation with Bio-Heat Equation using Physics-Informed Neural Networks». https://github.com/alessioborgi/BioHeat_PINNs/tree/main?tab=readme-ov-file#bioheat-pinns-temperature-estimation-with-bio-heat-equation-using-physics-informed-neural-networks.

Blechschmidt, Jan, y Oliver G. Ernst. 2021. «Three ways to solve partial differential equations with neural networks—A review». *GAMM-Mitteilungen* 44 (2): e202100006. https://doi.org/10.1002/gamm.202100006.

Dutta, Abhijit, y Gopal Rangarajan. 2018. «Diffusion in pharmaceutical systems: modelling and applications». *Journal of Pharmacy and Pharmacology* 70 (5): 581-98. https://doi.org/10.1111/jphp.12885.

Instituto Nacional del Cáncer. 2021. «¿Qué es el cáncer?» https://www.cancer.gov/espanol/cancer/naturaleza/que-es.

Karniadakis, George Em, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, y Liu Yang. 2021. «Physics-informed machine learning». *Nature Reviews Physics* 3 (6): 422-40. https://doi.org/10.1038/s42254-021-00314-5.

Kumar, Varun, Somdatta Goswami, Katiana Kontolati, Michael D. Shields, y George Em Karniadakis. 2024. «Synergistic Learning with Multi-Task DeepONet for Efficient PDE Problem Solving». *arXiv preprint arXiv:2408.02198*. https://arxiv.org/abs/2408.02198.

Lu, Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, y George Em Karniadakis. 2021. «Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators». *Nature Machine Intelligence* 3 (3): 218-29. https://doi.org/10.1038/s42256-021-00302-5.

Lu, Lu, Xuhui Meng, Zhiping Mao, y George Em Karniadakis. 2021. «DeepXDE: A deep learning library for solving differential equations». *SIAM Review* 63 (1): 208-28. https://doi.org/10.1137/19M1274067.

National Cancer Institute. 2021. «Hyperthermia to Treat Cancer». https://www.cancer.gov/about-cancer/treatment/types/hyperthermia.

Organización Mundial de la Salud. 2022. «Cáncer». https://www.who.int/es/news-room/fact-sheets/detail/cancer.

Pennes, H. H. 1948. «Analysis of Tissue and Arterial Blood Temperatures in the Resting Human Forearm». *Journal of Applied Physiology* 1 (2): 93-122. https://doi.org/10.1152/jappl.1948.1.2.93.

Quintero, Luis A., Mauricio Peñuela, Armando Zambrano, y Edwin Rodríguez. 2017. «Optimización del proceso de preparación de soluciones madre de antibióticos en un servicio farmacéutico hospitalario». *Revista Cubana de Farmacia* 50 (2): 448-65. https://www.medigraphic.com/cgi-bin/new/resumen.cgi?IDARTICULO=75483.

Yang, Lihong, Xin Wu, Qian Wan, Jian Kong, Rui Liu, y Xiaoxi Liu. 2014. «Pharmaceutical preparation of antibiotics: a review on formulation and technique». *Asian Journal of Pharmaceutical Sciences* 9 (3): 145-53. https://doi.org/10.1016/j.ajps.2014.04.001.