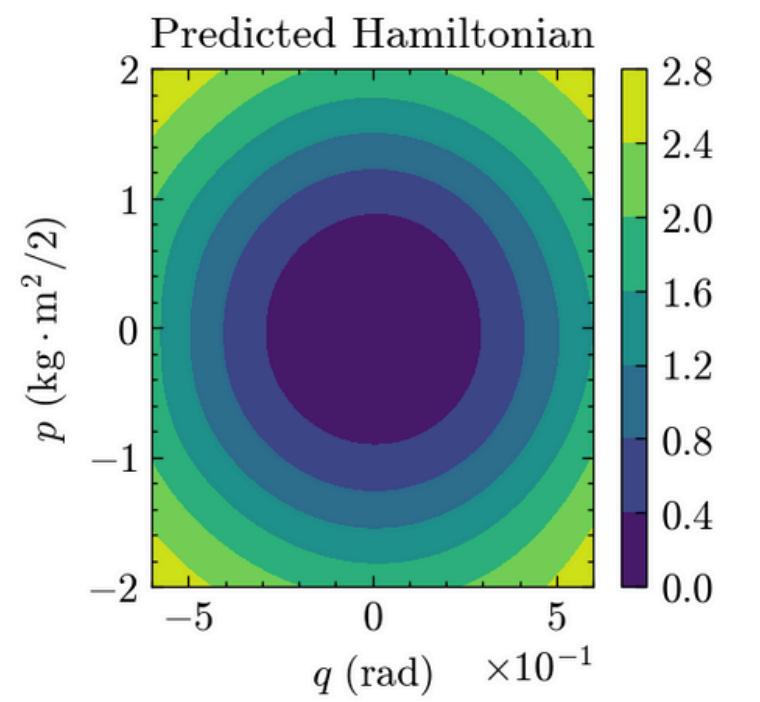
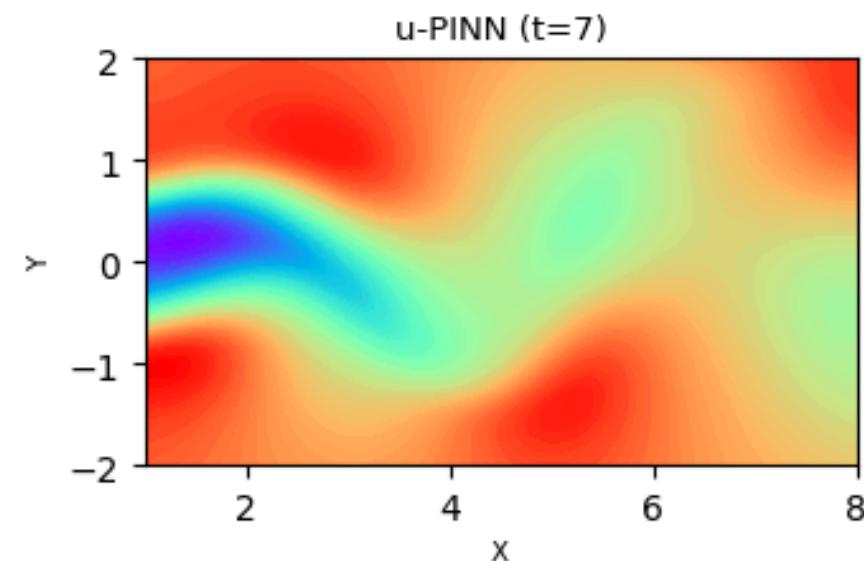
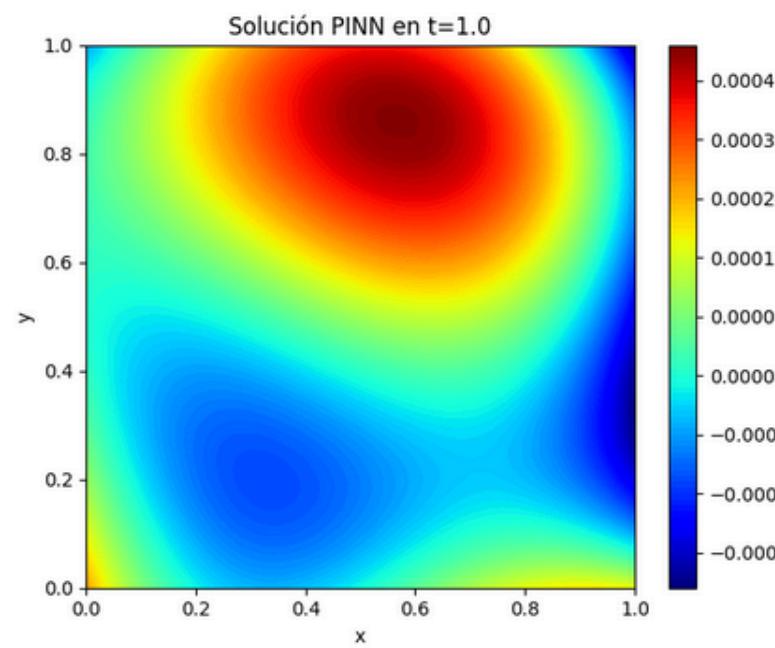


Redes Neuronales Informadas por la Física (PINNs): Una Nueva Frontera en el Modelamiento Computacional



Modelamiento físico computacional

Indice

1. El Escenario del Modelamiento Físico.

a. El Objetivo Universal.

b. El Arsenal Clásico.

c. Los Muros Computacionales.

2. La Arquitectura Matemática de las PINNs.

a. El fundamento Teórico: Aproximadores Universales.

b. La función de Pérdida: El Corazón Informada por la Fisica.

c. El Proceso de Optimización.

3. Implementación Práctica con DeepXDE.

a. ¿Qué es DeepXDE?.

b. Instalación y Dependencias.

c. Los Bloques de Construcción en DeepXDE.

Indice

4. La Escalera de Complejidad: Casos de Estudio.

- a. Ejemplo 1: Keppler, movimiento planetario.
- b. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.
- c. Ejemplo 3: El Desafío Geométrico - Ecuacion de Poisson en un dominio no trivial.
- d. Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.
- e. Ejemplo 5: Ecuación de Klein-Gordon

5. Patologías de las PINN's

- a. El Problema del "Sesgo Espectral" (Spectral Bias).
- b. Inestabilidad en el Entrenamiento y Gradientes Desbalanceados.
- c. Manejo de Discontinuidades y Frentes de Choque.
- d. Falta de Garantías Teóricas

Indice

6. ¿Medicina para las patologías?

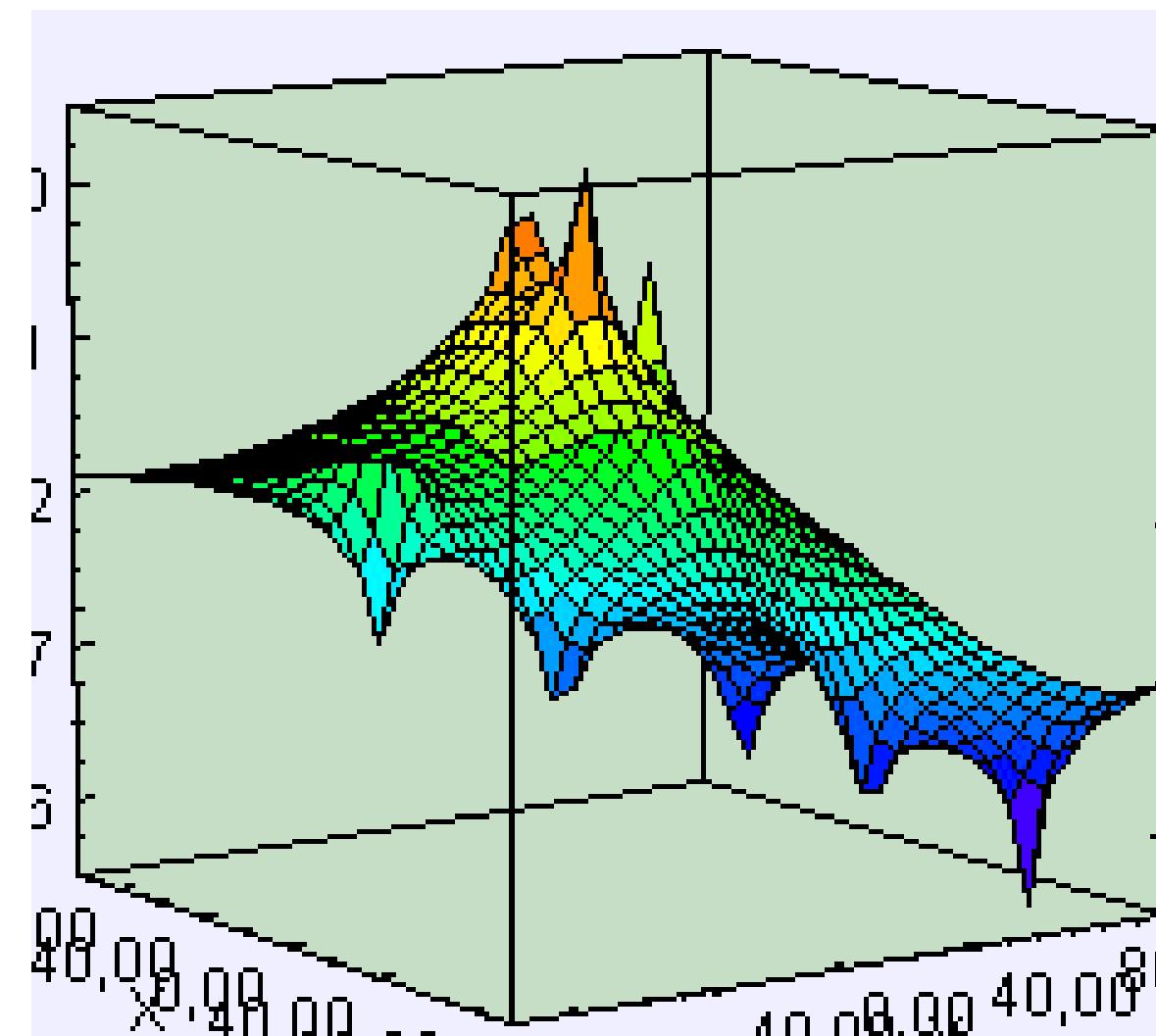
- a. Mitigación del Sesgo Espectral
- b. Estrategias de Ponderación Adaptativa de la Pérdida (Adaptive Loss Weighting)
- c. Extensiones Arquitectónicas y Enfoques Híbridos
- d. Incorporación de Simetrías y Múltiples Soluciones

7. Discusión Crítica y Conclusiones.

- a. Resumen de Poderes.
- b. Evaluación Crítica del Estado Actual.
- c. Nichos de Aplicación con Ventajas Claras.
- d. Direcciones Futuras

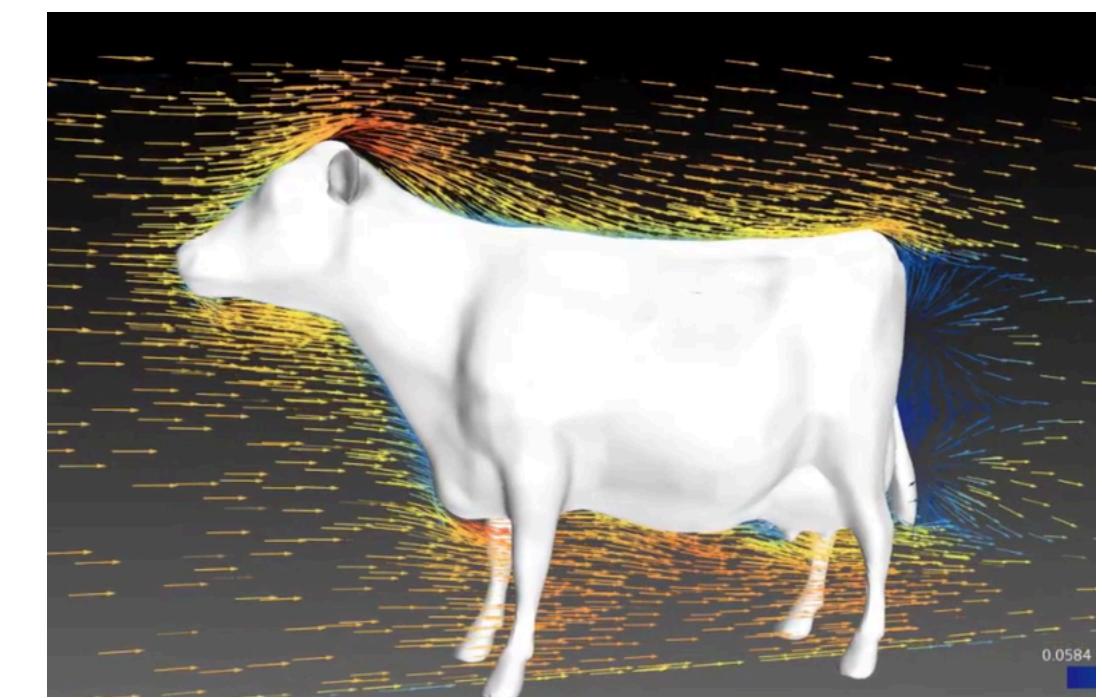
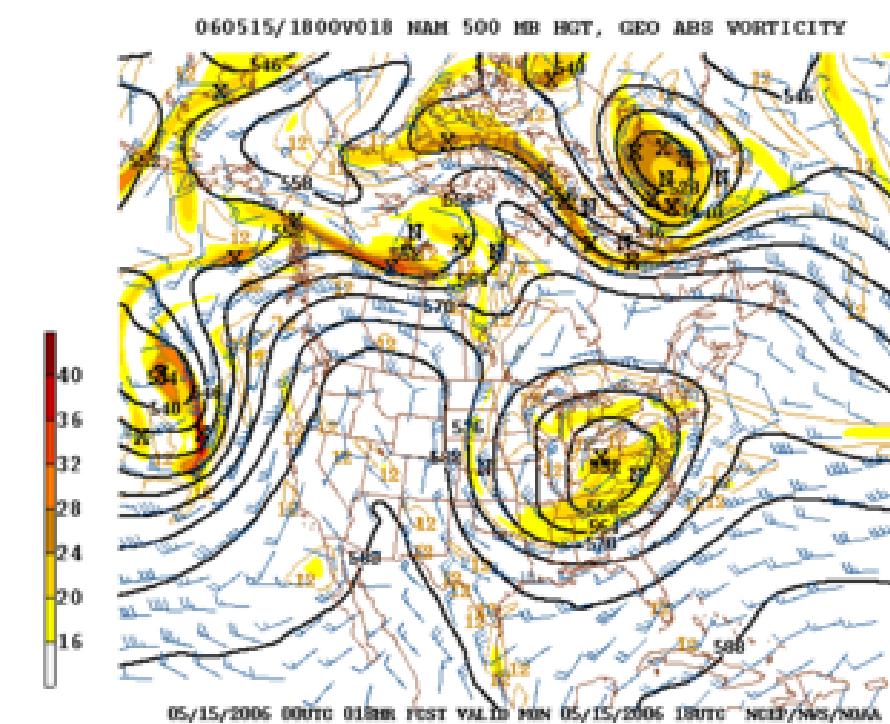
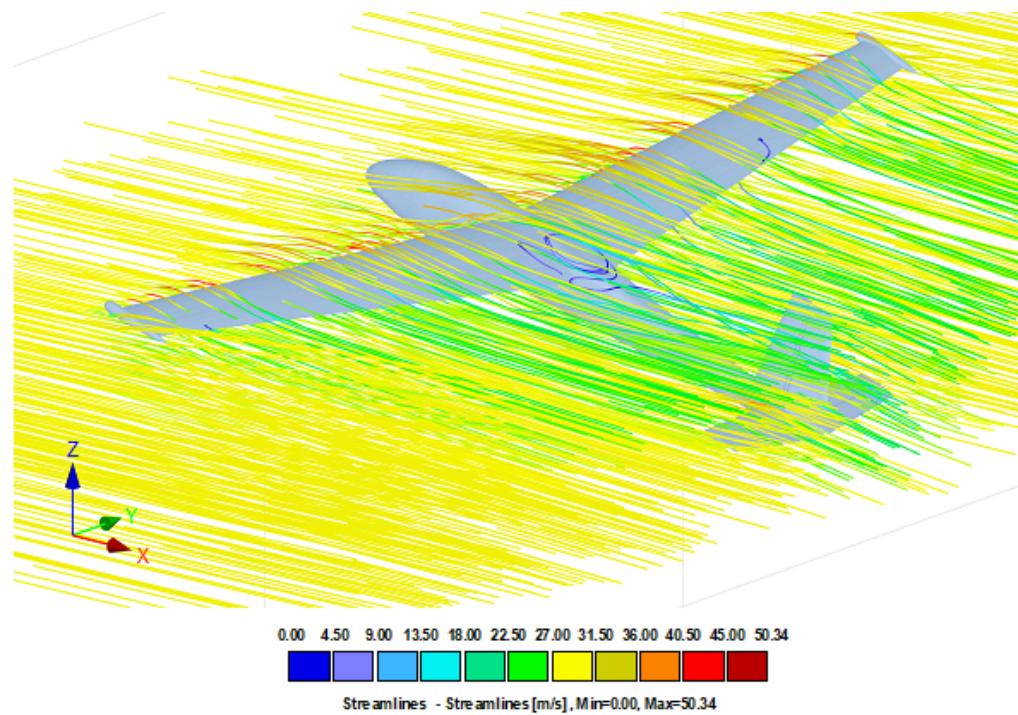
1. El Escenario del Modelamiento Físico.

De las Leyes Físicas a los Límites de la Simulación Clásica



El Mundo Oculto que Nos Rodea

¿Cómo se diseñan aviones más eficientes y seguros? ¿Cómo se predicen los patrones del clima o el impacto de un tsunami? o ¿Cómo se simula el flujo sanguíneo en una arteria para prevenir enfermedades?



La respuesta a todas estas preguntas es la misma: el modelamiento físico computacional. Usamos supercomputadoras para simular las leyes de la física, permitiéndonos ver lo invisible y predecir el futuro.

El Lenguaje de la Naturaleza: Ecuaciones Diferenciales Parciales (EDPs)

- Ecuaciones de Navier-Stokes: (Mecánica de Fluidos)

$$\rho \left(\frac{\partial \vec{v}}{\partial t} + \vec{v} \cdot \nabla \vec{v} \right) = -\nabla p + \mu \nabla^2 \vec{v} + \vec{f}$$

- Ecuación del Calor: (Transferencia de Energía)

$$\frac{\partial u}{\partial t} - \alpha \nabla^2 u = 0$$

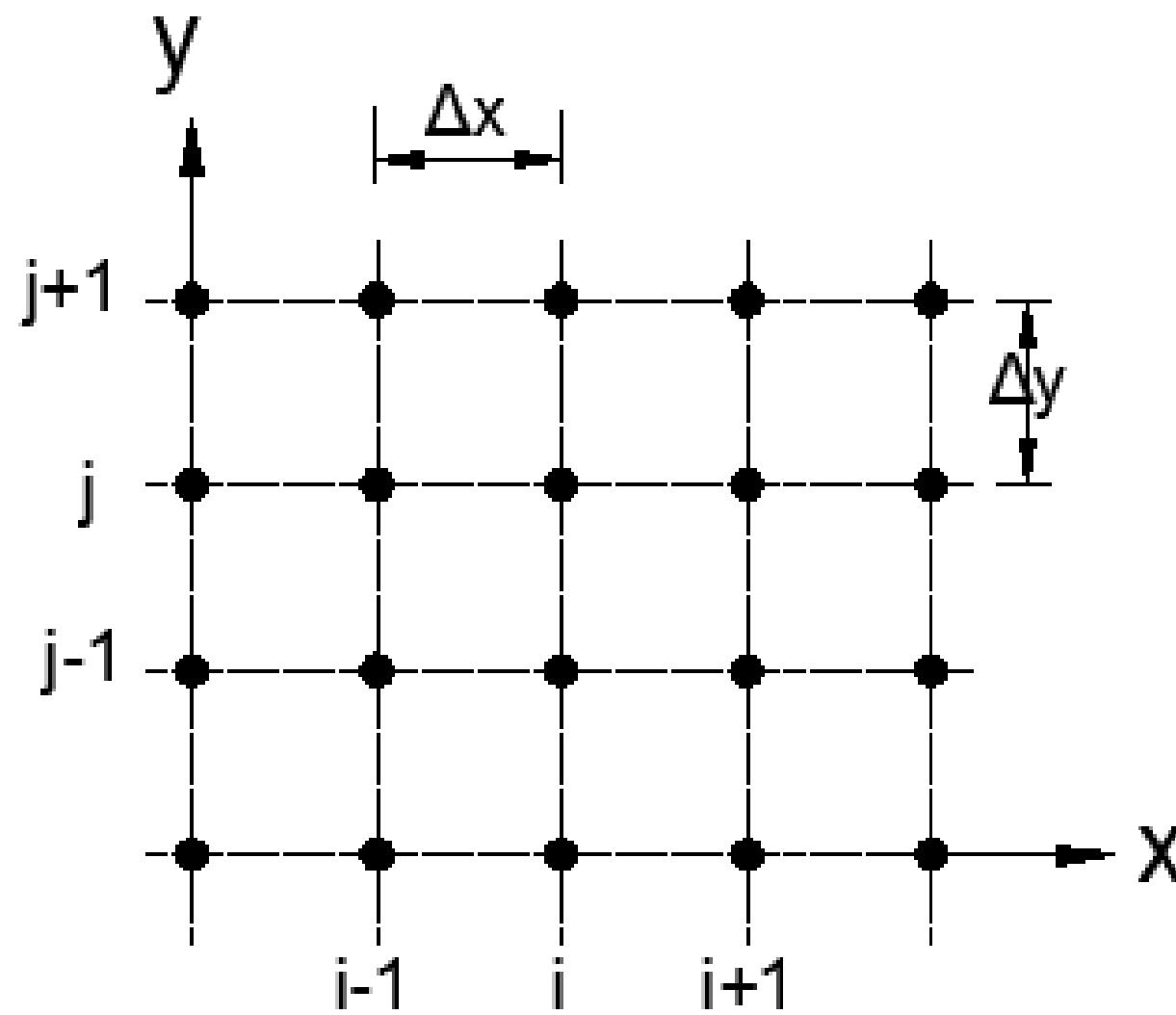
- Ecuación de Onda: (Vibraciones, Acústica, Electromagnetismo)

$$\frac{\partial^2 u}{\partial t^2} - c^2 \nabla^2 u = 0$$

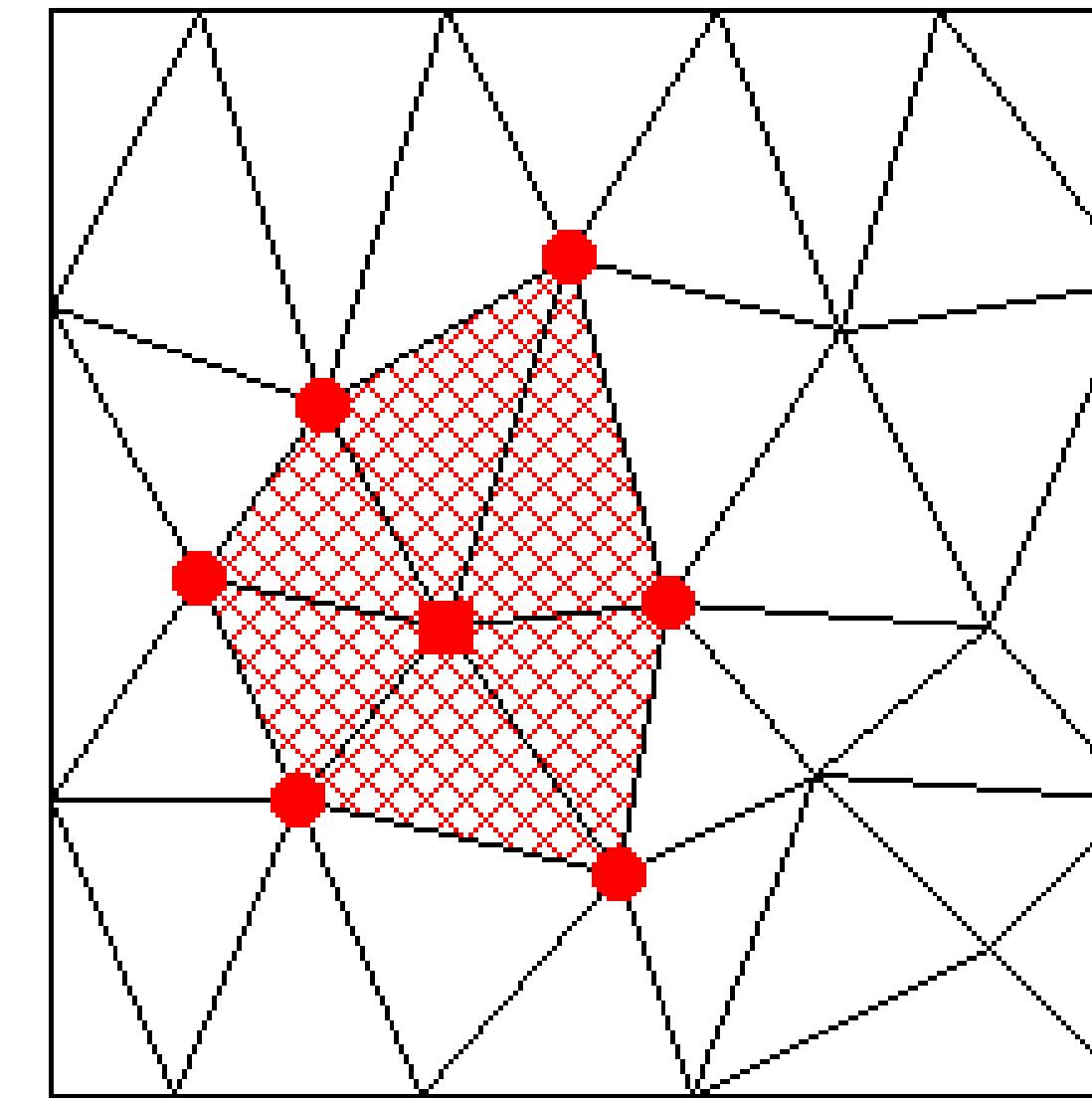
Resolver estas ecuaciones es la llave para desbloquear los secretos de estos fenómenos. El problema es que, para casi cualquier caso real, son imposibles de resolver a mano.

El Arsenal Clásico: ¿Cómo Resolvíamos Esto?

- Método de Diferencias Finitas (FDM):

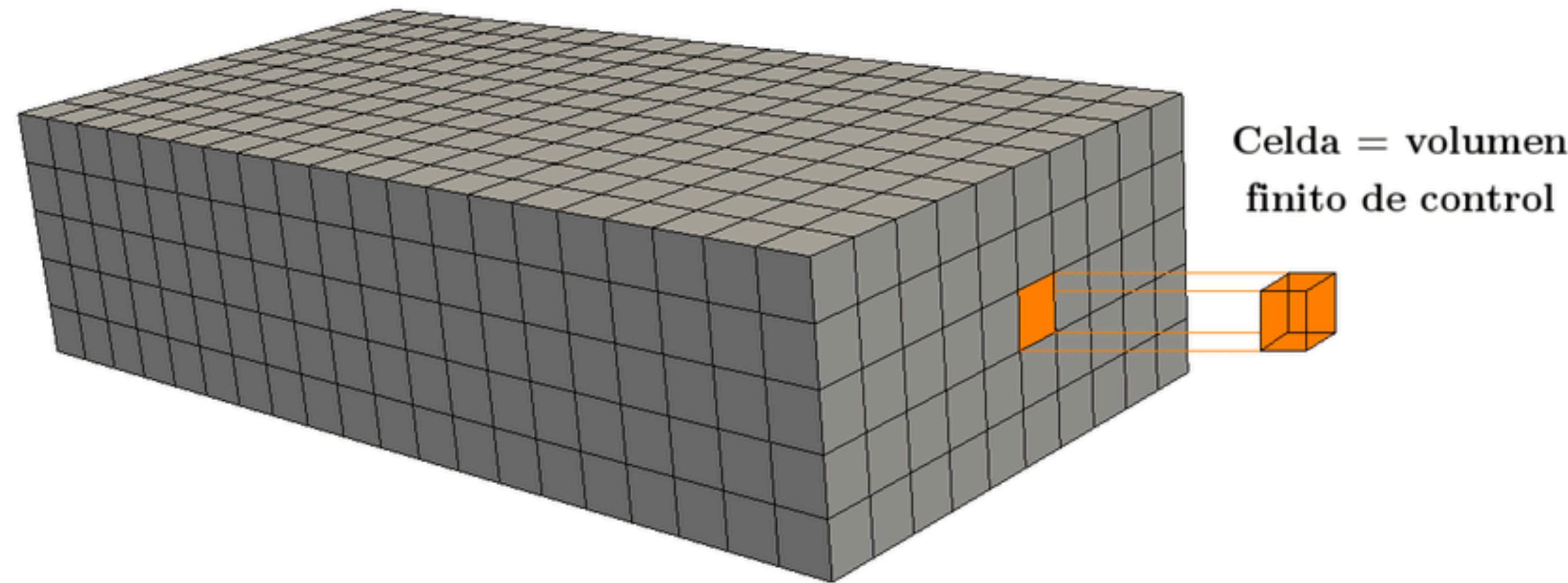


- Método de Elementos Finitos (FEM):



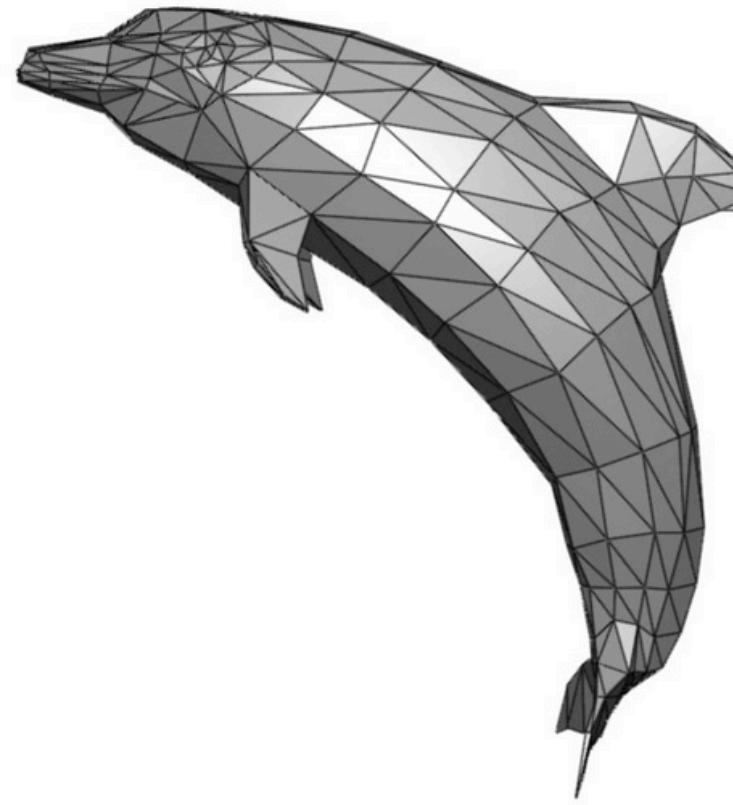
El Arsenal Clásico: ¿Cómo Resolvíamos Esto?

- Método de Volúmenes Finitos (FVM):



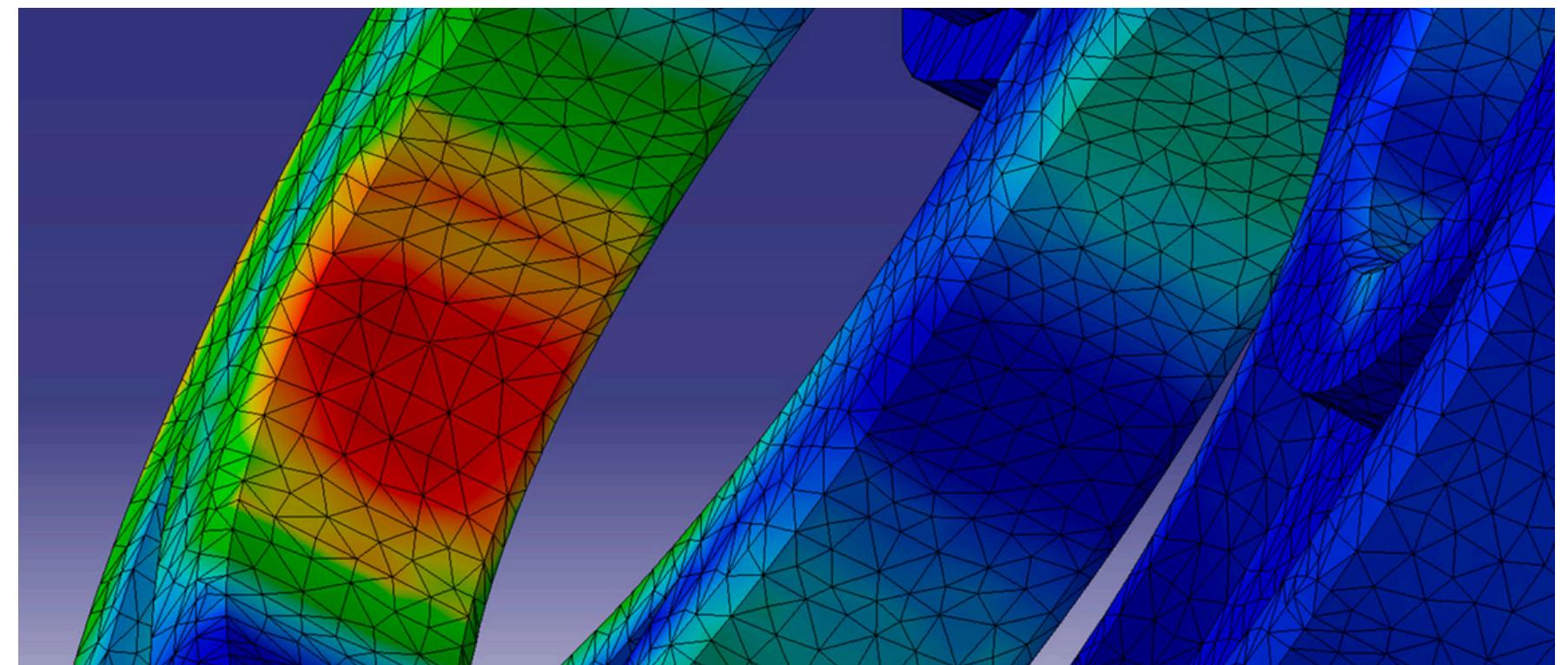
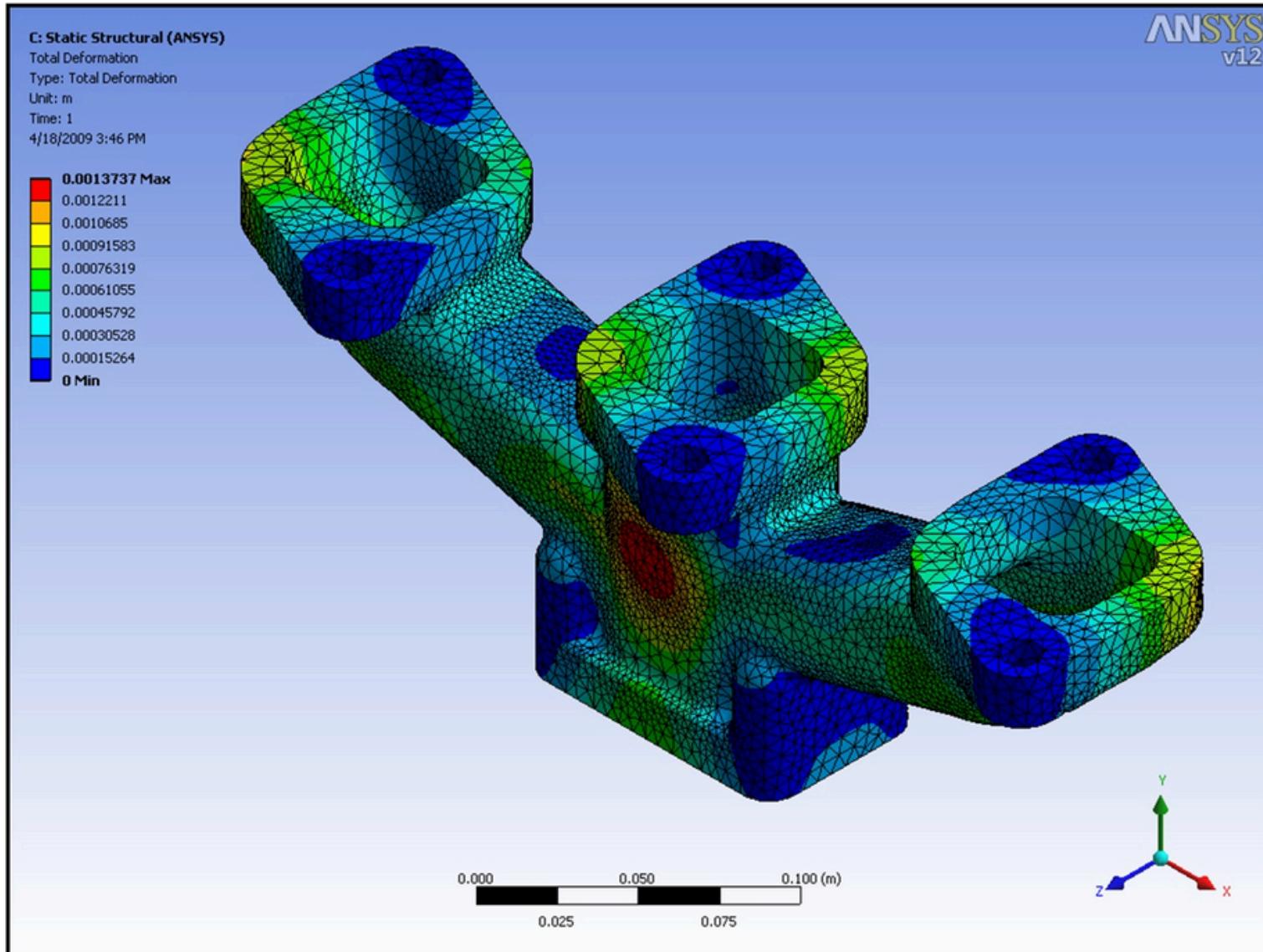
Estos métodos son los pilares de la ingeniería y la ciencia moderna. Y todos, sin excepción, se basan en una única idea fundamental...

La Idea Central Clásica: Discretización



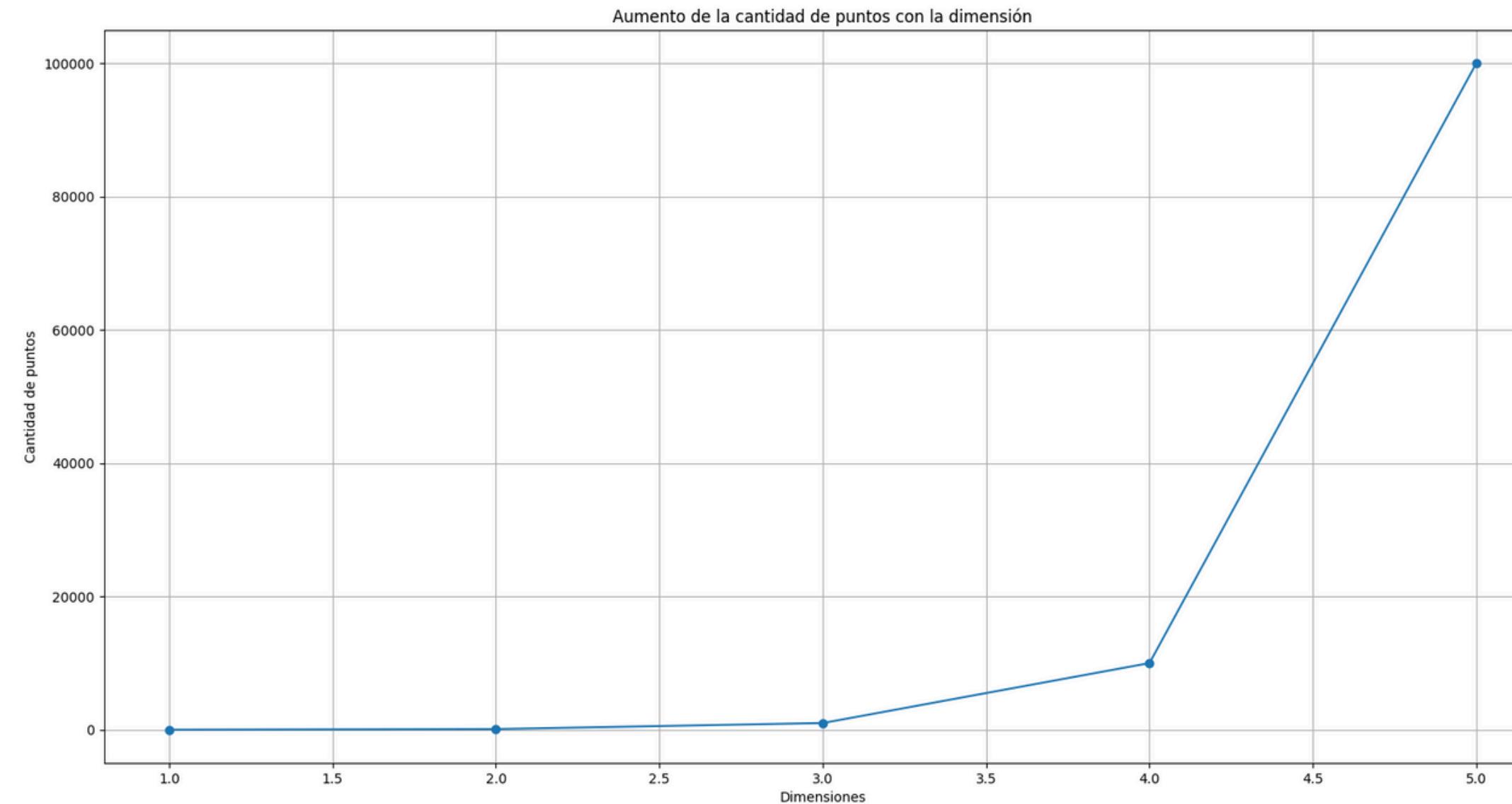
Convertimos un problema de cálculo (EDPs) en un problema de álgebra (un sistema gigante de ecuaciones lineales) que una computadora puede resolver. Esta idea es la fuente del poder de los métodos clásicos, pero también, como veremos, de sus mayores debilidades.

Primer Muro: El "Infierno de la Malla" (Meshing Hell)



A veces, el esfuerzo para generar la malla supera por mucho el esfuerzo de resolver la física del problema.

Segundo Muro: La Maldición de la Dimensionalidad

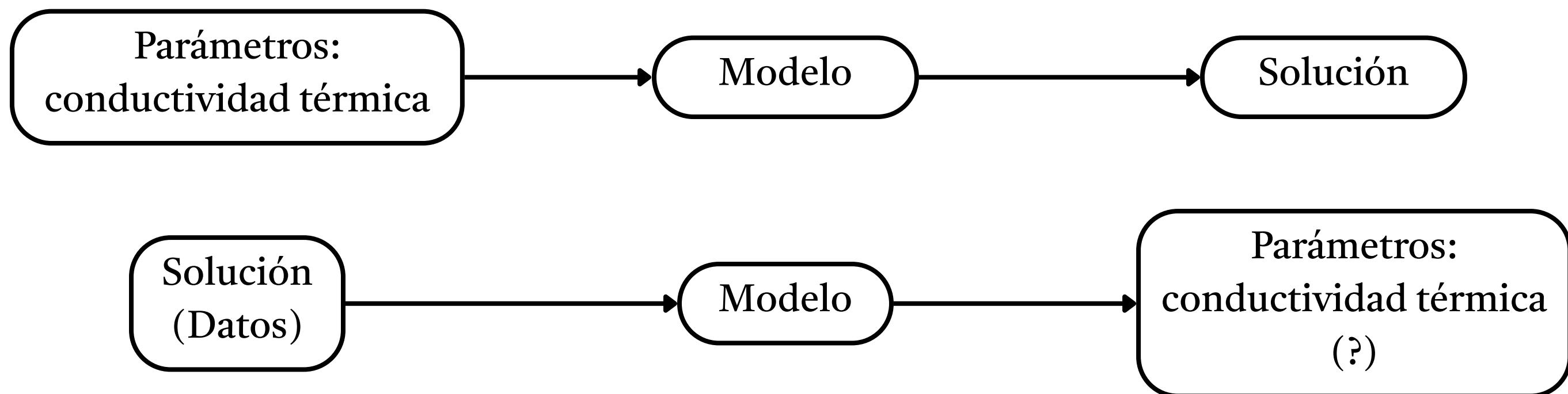


$$\text{cantidad puntos} = N^d$$

Los métodos basados en mallas son prácticamente inútiles para problemas de alta dimensión, como los que encontramos en finanzas (modelando carteras con muchos activos) o en química cuántica.

Tercer Muro: El Dilema del Problema Inverso

Imaginemos dos escenarios:



Este segundo tipo de problema, el problema inverso, es fundamental para la ciencia y la ingeniería, pero es extremadamente difícil y costoso de resolver con el arsenal clásico.

La Necesidad de un Nuevo Paradigma

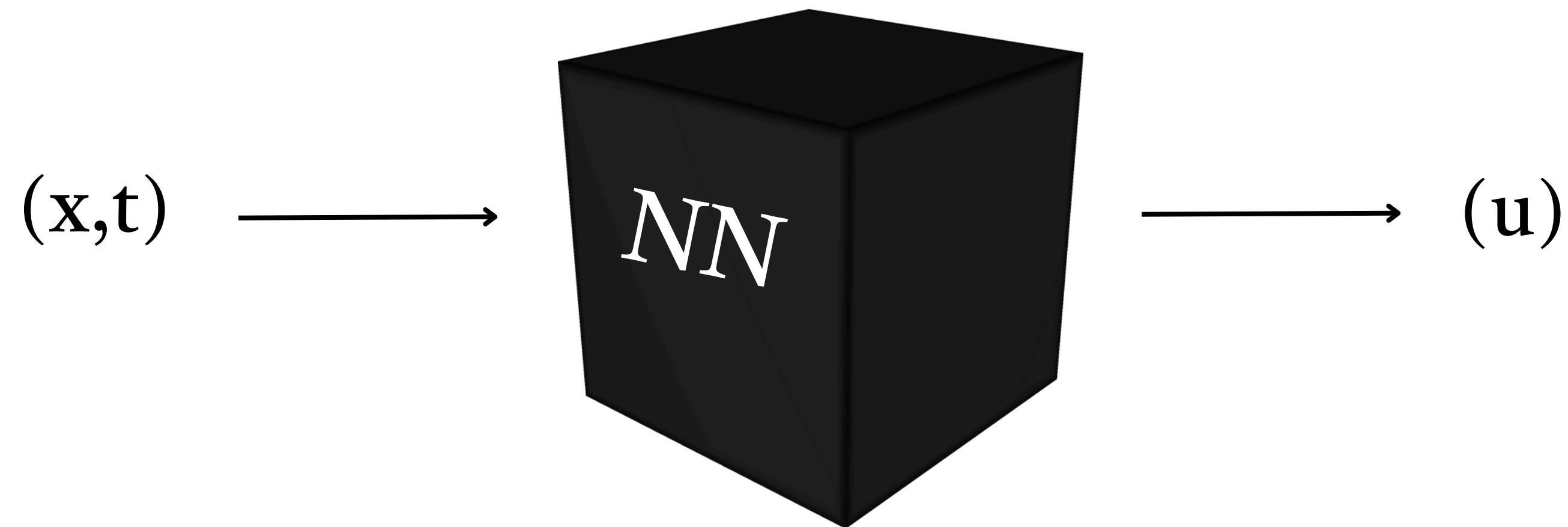
¿Podríamos concebir un método de solución que...

- sea libre de mallas por naturaleza?
- no sufra la maldición de la dimensionalidad?
- pueda fusionar datos experimentales y leyes físicas de forma nativa para resolver problemas inversos?

La búsqueda de una respuesta a esta pregunta nos lleva a la emocionante intersección de dos campos que antes estaban separados: la simulación física y el machine learning. Nos lleva a las Redes Neuronales Informadas por la Física.

2. La Arquitectura Matemática de las PINNs

Del Teorema de Aproximación Universal a la Optimización por Gradiente.



¿Por qué usar una red Neuronal?

Antes de ver cómo funcionan, debemos responder por qué podemos usarlas. Si una PINN es una red neuronal, ¿qué garantía teórica tenemos de que esta estructura puede si quiera representar la solución a una compleja ley física?

La solución a una EDP, $u(x,t)$, es una función. Necesitamos una herramienta que pueda aproximar funciones arbitrariamente complejas.

Afortunadamente, la teoría matemática nos da una base sólida.

Fundamento: El Teorema de Aproximación Universal

"Una red neuronal con una sola capa oculta, suficientemente ancha, puede aproximar cualquier función continua de interés con la precisión que queramos."

Este teorema nos da la confianza teórica para proponer una red neuronal, $u\theta(x,t)$, como una "plantilla" para la solución de nuestra EDP. La red es un **aproximador de funciones universal**, y la solución que buscamos es, en efecto, una función.

Nota: El teorema es una declaración de existencia: nos dice que un conjunto de pesos y sesgos (θ) existe, pero no nos dice cómo encontrarlo. La tarea de encontrar los pesos óptimos es el trabajo del entrenamiento.

El Motor: La función de Pérdida Híbrida

El objetivo del entrenamiento es minimizar una función de pérdida (o de coste). En las PINNs, esta función es una combinación inteligente que une el mundo de los datos y el de la física.

$$L_{total}(\theta) = \omega_{data}L_{data} + \omega_{physics}L_{physics}$$

θ : Los pesos y sesgos de la red neuronal que queremos optimizar.

L_{data} : Mide el error en los datos conocidos (condiciones de frontera/iniciales).

$L_{physics}$: Mide cómo la solución de la red viola la ley física (la EDP).

$\omega_{physics}, \omega_{data}$: Hiperparámetros que balancean la importancia de cada término.

Pérdida de Datos (L_{data}): Anclaje a la Realidad

Esta es la parte de "Machine Learning supervisado" tradicional. Forzamos a que la red neuronal coincida con las mediciones o condiciones que ya conocemos.

Se calcula como el Error Cuadrático Medio (MSE) sobre todos los puntos de datos disponibles (condiciones iniciales N_{ic} y de frontera N_{bc}).

$$L_{data} = \frac{1}{N_{ic}} \sum_{i=1}^{N_{ic}} |u_\theta(x_i, t_0) - u_{ic}(x_i)|^2 + \frac{1}{N_{bc}} \sum_{j=1}^{N_{bc}} |u_\theta(x_j, t_j) - u_{bc}(x_j, t_j)|^2$$

Pérdida de Física ($L_{physics}$): El Vínculo con las Leyes Naturales

Esta es la innovación clave. No solo usamos datos, usamos el conocimiento de la ecuación misma para entrenar la red en puntos donde no tenemos datos.

Ejemplo Concreto: Ecuación de Burgers:

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

Pérdida de Física ($L_{physics}$): El Vínculo con las Leyes Naturales

1. Definir el Residuo (f): El residuo es lo que la red neuronal u_θ produce al ser insertada en la EDP.

$$f(x, t) := \frac{\partial u_\theta}{\partial t} + u_\theta \frac{\partial u_\theta}{\partial x} - \nu \frac{\partial^2 u_\theta}{\partial x^2}$$

Pérdida de Física ($L_{physics}$): El Vínculo con las Leyes Naturales

2.Calcular la Pérdida: Penalizamos cualquier desviación de cero en un gran número de puntos aleatorios (N_c , puntos de colocación) dentro del dominio.

$$L_{physics} = \frac{1}{N_c} \sum_{j=1}^{N_c} |f(x_j, t_j)|^2$$

"La red debe aprender una función u_θ que, al derivarla e insertarla en la ecuación, el resultado sea lo más cercano a cero posible en todo el dominio espacio-temporal".

El Mecanismo: ¿Cómo Calculamos las Derivadas?

¿Cómo se calculan términos como $\frac{\partial u_\theta}{\partial x}$ o $\frac{\partial^2 u_\theta}{\partial x^2}$ de forma eficiente y precisa?

¿Qué no usar?

- Diferenciación Simbólica: Ineficiente. Genera expresiones gigantes y lentas de evaluar.
- Diferenciación Numérica (ej. Diferencias Finitas): Imprecisa. Introduce errores de truncamiento y es numéricamente inestable.

La Solución: Diferenciación Automática - AD

- Funciona aplicando la regla de la cadena sistemáticamente a nivel de operaciones elementales (suma, producto, tanh, etc.) en el grafo computacional de la red.

AD: Modo Forward vs. Modo Reverso

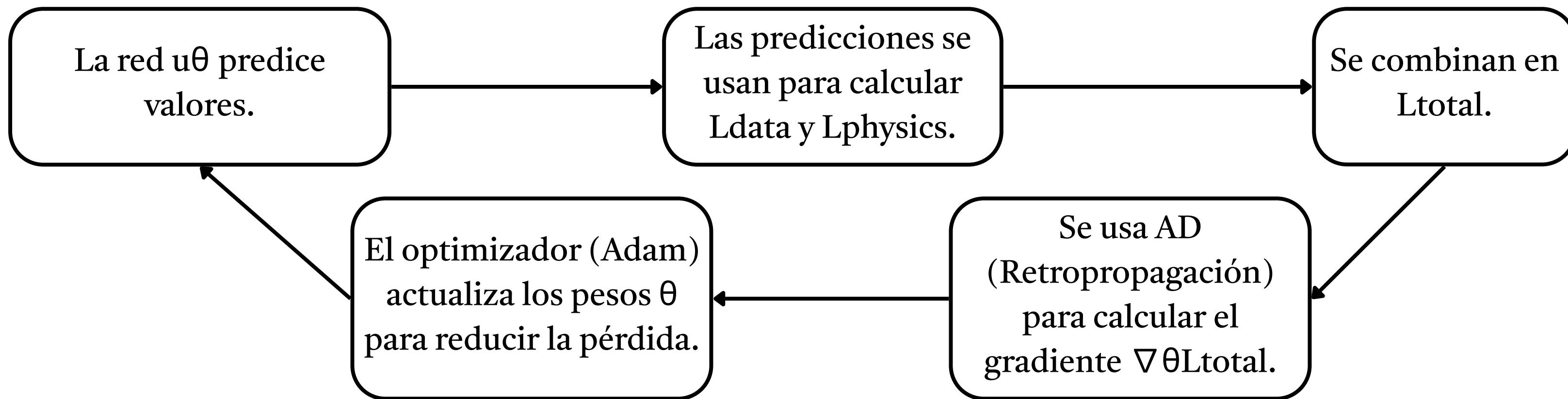
Modo Forward:

- Calcula cómo un único input afecta a todas las salidas.
- Recorre el grafo computacional "hacia adelante", desde las entradas hasta las salidas.
- Eficiente si el número de entradas es mucho menor que el de salidas.

Modo Reverso (Backpropagation):

- Calcula cómo todos los inputs afectan a una única salida.
- Recorre el grafo "hacia atrás", desde la salida final (la pérdida) hasta las entradas (los pesos θ).
- Esta es la clave: Para entrenar una red, tenemos una única salida (el valor escalar de la pérdida $L_{total}(\theta)$) y millones de entradas (los pesos y sesgos θ).

Juntándolo Todo: El Proceso de Optimización



La actualización iterativa:

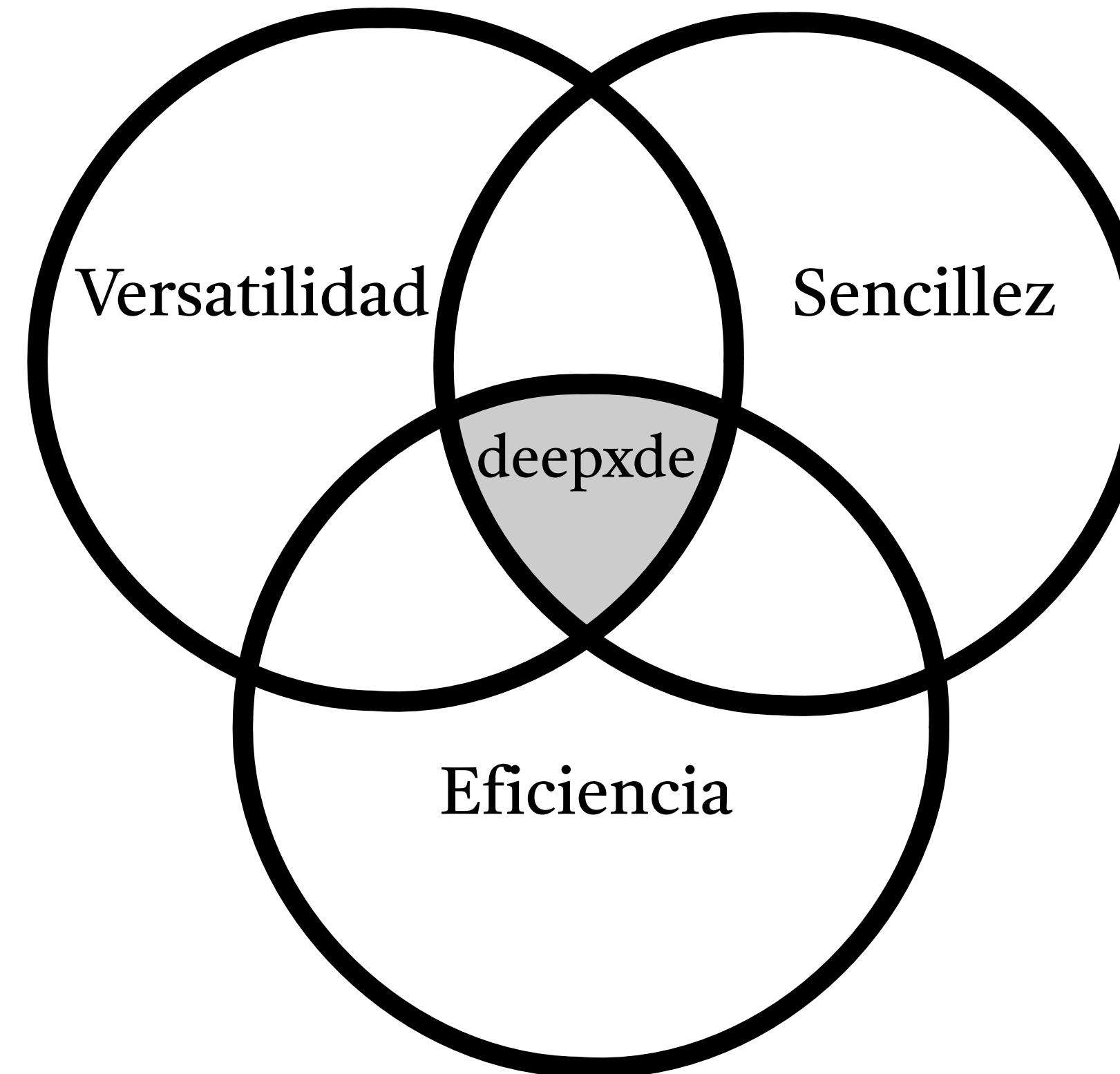
$$\theta_{k+1} = \theta_k - \eta \nabla_\theta L_{total}(\theta_k)$$

3. Implementación Práctica con DeepXDE

Del Papel al Código: Traduciendo la Física con Abstracciones de Alto Nivel

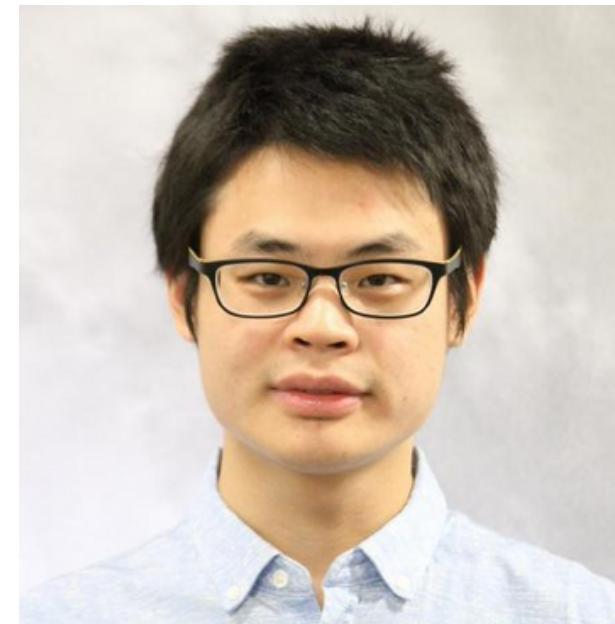
```
1 import deepxde as dde  
2 from deepxde.backend import tf  
~
```

¿Qué es deepxde?



Línea de Tiempo


BROWN
SciCoNet
2019

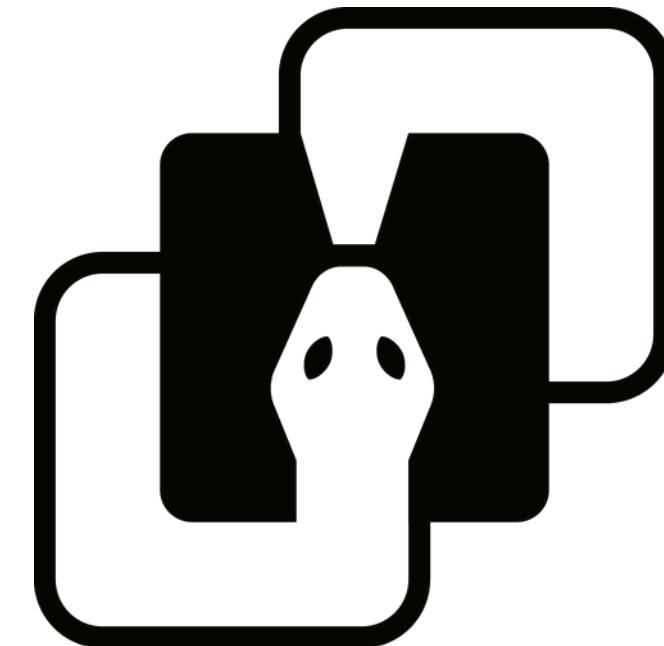


Lu Lu

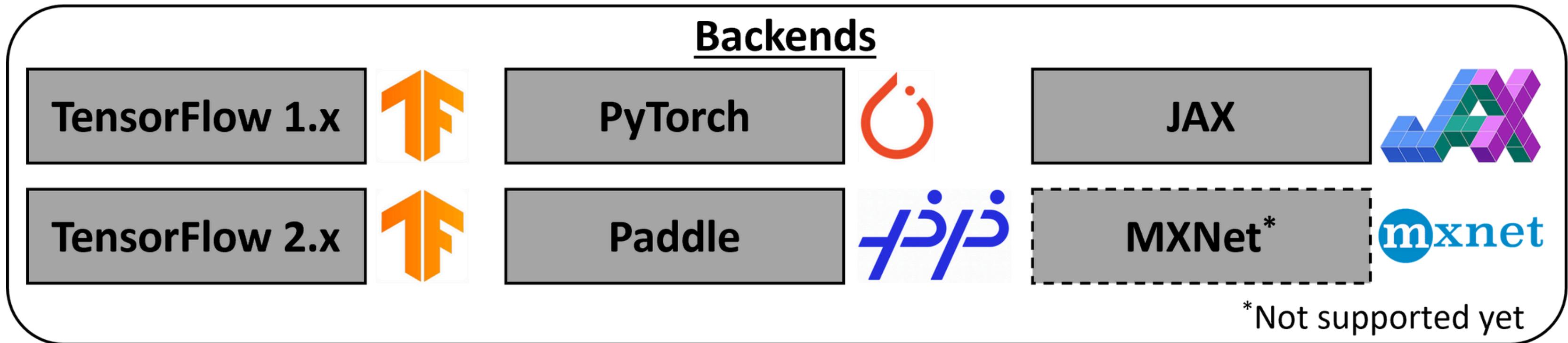



Yale University
DeepXDE
2020

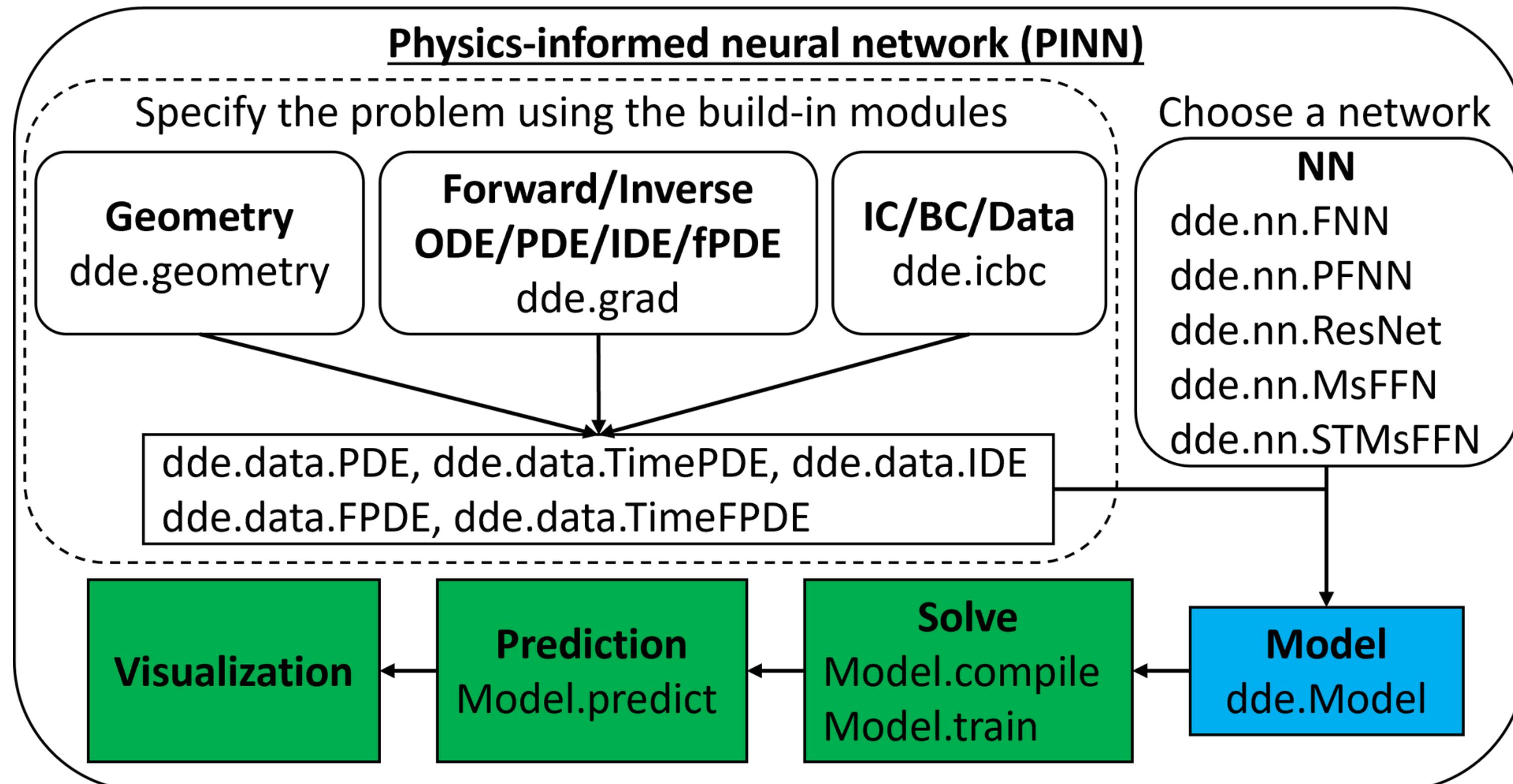
Dependencias



Backends



Bloques de construcción en dde



Geometries

```
deepxde.geometry.geometry_1d.Interval(l, r)
```

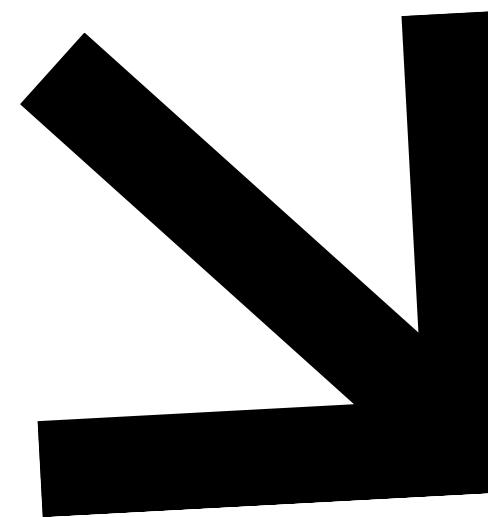
```
deepxde.geometry.geometry_2d.StarShaped(center, radius, coeffs_cos, coeffs_sin)
```

```
class deepxde.geometry.geometry_3d.Sphere(center, radius)
```

```
deepxde.geometry.timedomain.TimeDomain(t0, t1)
```

dde.data.DataSet

```
fname_train = ".../dataset/dataset.train"  
fname_test = ".../dataset/dataset.test"
```



```
data = dde.data.DataSet(  
    fname_train=fname_train,  
    fname_test=fname_test,  
    col_x=(0,),  
    col_y=(1,),  
    standardize=True,  
)
```

Condiciones iniciales y de frontera

```
deepxde.icbc.boundary_conditions.BC(geom, on_boundary, component)
```

```
deepxde.icbc.boundary_conditions.DirichletBC(geom, func, on_boundary, component=0)
```

```
deepxde.icbc.boundary_conditions.NeumannBC(geom, func, on_boundary, component=0)
```

```
deepxde.icbc.initial_conditions.IC(geom, func, on_initial, component=0)
```

dde.gradients

```
deepxde.gradients.gradients.hessian(ys, xs, component=0, i=0, j=0)
```

```
deepxde.gradients.gradients.jacobian(ys, xs, i=None, j=None)
```

Tipos de problemas

`dde.data.PDE`, `dde.data.TimePDE`, `dde.data.IDE`

`dde.data.FPDE`, `dde.data.TimeFPDE`

$$\vec{\nabla}^2 \psi = \alpha \frac{\partial^2 \psi}{\partial t^2}$$

$$\left(\frac{\hbar^2}{2m} \vec{\nabla}^2 + V(r) \right) \psi = E \psi$$

$$\nabla^\alpha U = 0$$

$$\frac{d(u(x))}{dx} = \int_0^x e^t u(t) dt$$

Modelos

NN

dde.nn.FNN

dde.nn.PFNN

dde.nn.ResNet

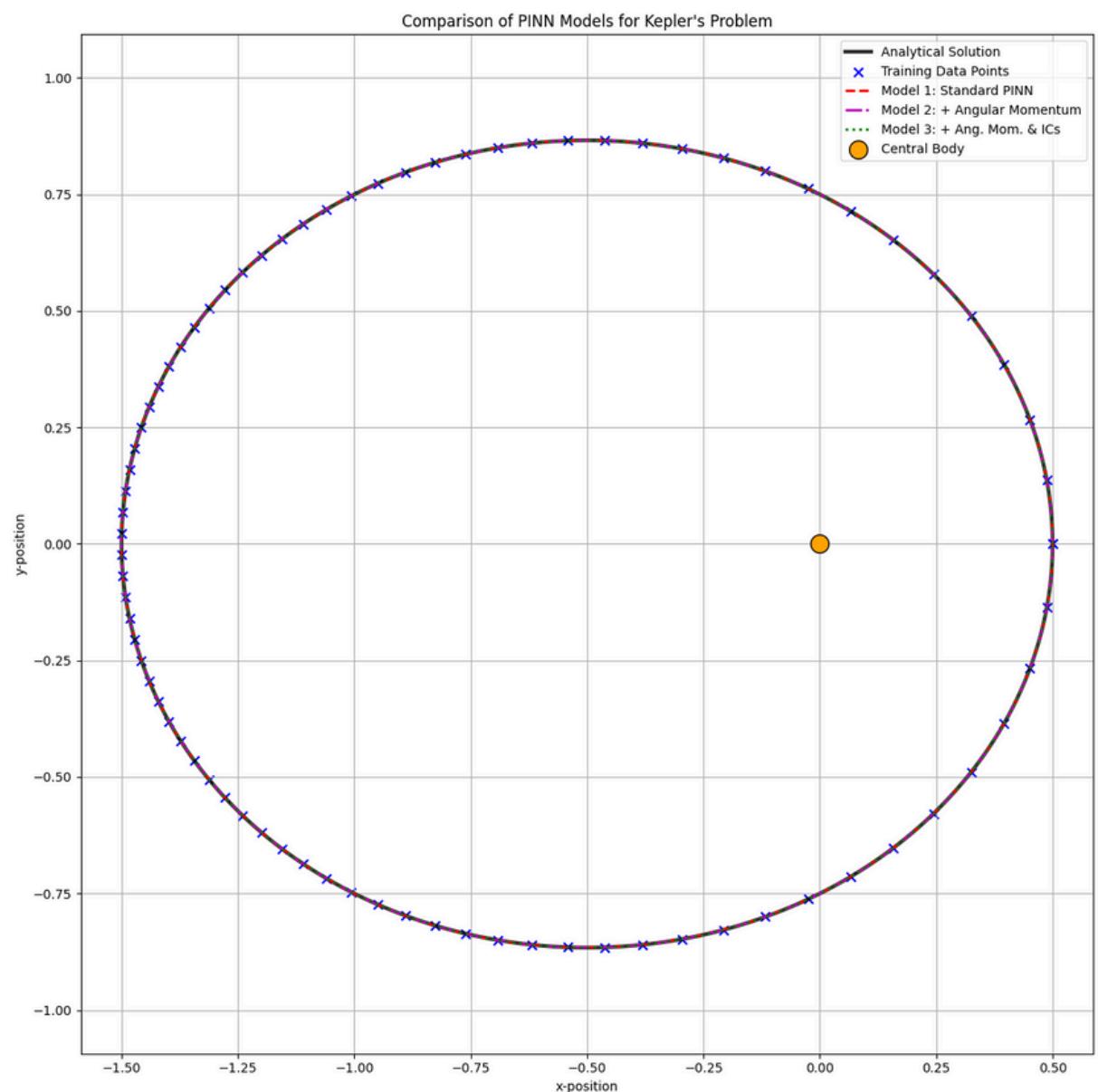
dde.nn.MsFFN

dde.nn.STMsFFN

4. Casos de Estudio: PINNs en Acción

Más Allá de la Malla: De la Geometría Compleja al Problema Inverso

Keppler, movimiento planetario.



$$\frac{d^2 \mathbf{x}}{dt^2} = -\frac{K}{x^2} \hat{\mathbf{x}}$$

Modelos:

- ODE.
- Conservación del momentum angular.
- Condiciones iniciales.

Implementación

```
# Physical constants for the Kepler problem
MU = 1.0 # Standard gravitational parameter (G*M)
a = 1.0 # Semi-major axis
e = 0.5 # Eccentricity
T = 2 * np.pi * np.sqrt(a**3 / MU) # Orbital period from Kepler's Third Law

def solve_kepler(M, e, tol=1e-9):
    """
    Solves Kepler's equation M = E - e*sin(E) for the eccentric anomaly E,
    given the mean anomaly M and eccentricity e.
    Uses the Newton-Raphson iterative method.
    """
    E = M
    for _ in range(100):
        f = E - e * np.sin(E) - M
        f_prime = 1 - e * np.cos(E)
        E_new = E - f / f_prime
        if np.all(np.abs(E_new - E) < tol):
            return E_new
        E = E_new
    return E
```

4.1. Ejemplo 1: Keppler, movimiento planetario.

4. Casos de Estudio: PINNs en Acción

```
def pde_system(t, u):
    """
    Defines the governing differential equations for Kepler's problem (Newton's law of gravitation).
    This function must use backend-compatible operations.
    """

    x, y = u[:, 0:1], u[:, 1:2]
    ax = dde.grad.hessian(u, t, component=0, i=0, j=0)
    ay = dde.grad.hessian(u, t, component=1, i=0, j=0)

    r_sq = x**2 + y**2

    epsilon = torch.tensor(1e-8, dtype=t.dtype, device=t.device)
    mu_tensor = torch.tensor(MU, dtype=t.dtype, device=t.device)

    r = (r_sq + epsilon)**0.5

    res_x = ax + mu_tensor * x / (r**3)
    res_y = ay + mu_tensor * y / (r**3)

    return [res_x, res_y]
```

```
def angular_momentum_residual(t, u):
    """
    Defines the residual for the conservation of specific angular momentum (h).
    """

    x, y = u[:, 0:1], u[:, 1:2]
    vx = dde.grad.jacobian(u, t, i=0, j=0)
    vy = dde.grad.jacobian(u, t, i=1, j=0)
    h = x * vy - y * vx
    h0_tensor = torch.tensor(h0, dtype=t.dtype, device=t.device)
    return h - h0_tensor
```

```
def create_fnn():
    """Creates a standard Feedforward Neural Network for the models."""
    return dde.nn.FNN([1] + [64] * 4 + [2], "tanh", "Glorot normal")

def combined_physics_model(t, u):
    pde_res = pde_system(t, u)
    ang_mom_res = angular_momentum_residual(t, u)
    return pde_res + [ang_mom_res]

solution_fn = lambda t: get_analytical_solution(t, a, e, MU)

# Define the metric to be used, which is the L2 relative error.
# This serves as a normalized Root Mean Squared Error (RMSE).
metrics = ["l2 relative error"]
```

4.1. Ejemplo 1: Keppler, movimiento planetario.

4. Casos de Estudio: PINNs en Acción

```
# --- Model 1: Standard PINN (PDE + Data) ---
print("\n--- Setting up Model 1: Standard PINN (PDE + Data) ---")
data1 = dde.data.PDE(geom, pde_system, bcs_data_only, num_domain=500, num_boundary=2, solution=solution_fn, num_collocation_points=100)
model1 = dde.Model(data1, create_fnn())
loss_weights1 = [1, 1, 100, 100]
model1.compile("adam", lr=1e-3, loss_weights=loss_weights1, metrics=metrics)

# --- Model 2: PINN with Angular Momentum (PDE + Data + Ang. Mom.) ---
print("\n--- Setting up Model 2: PINN with Angular Momentum ---")
data2 = dde.data.PDE(geom, combined_physics_model, bcs_data_only, num_domain=500, num_boundary=2, solution=solution_fn)
model2 = dde.Model(data2, create_fnn())
loss_weights2 = [1, 1, 10, 100, 100]
model2.compile("adam", lr=1e-3, loss_weights=loss_weights2, metrics=metrics)

# --- Model 3: PINN with Ang. Mom. and Full Initial Conditions ---
print("\n--- Setting up Model 3: PINN with Ang. Mom. and Full ICs ---")
data3 = dde.data.PDE(geom, combined_physics_model, bcs_full, num_domain=500, num_boundary=2, solution=solution_fn)
model3 = dde.Model(data3, create_fnn())
loss_weights3 = [1, 1, 10, 100, 100, 100, 100, 100]
model3.compile("adam", lr=1e-3, loss_weights=loss_weights3, metrics=metrics)
```

4.1. Ejemplo 1: Kepler, movimiento planetario.

4. Casos de Estudio: PINNs en Acción

```
# --- Training Loop ---
models_to_train = {
    "Standard_PINN": (model1, loss_weights1),
    "PINN_with_AngMom": (model2, loss_weights2),
    "PINN_with_AngMom_IC": (model3, loss_weights3)
}
predictions = {}

for name, (model, weights) in models_to_train.items():
    print(f"\n--- Training {name} ---")
    if not os.path.exists(name):
        os.makedirs(name)

    losshistory, train_state = model.train(iterations=40000, display_every=2000)

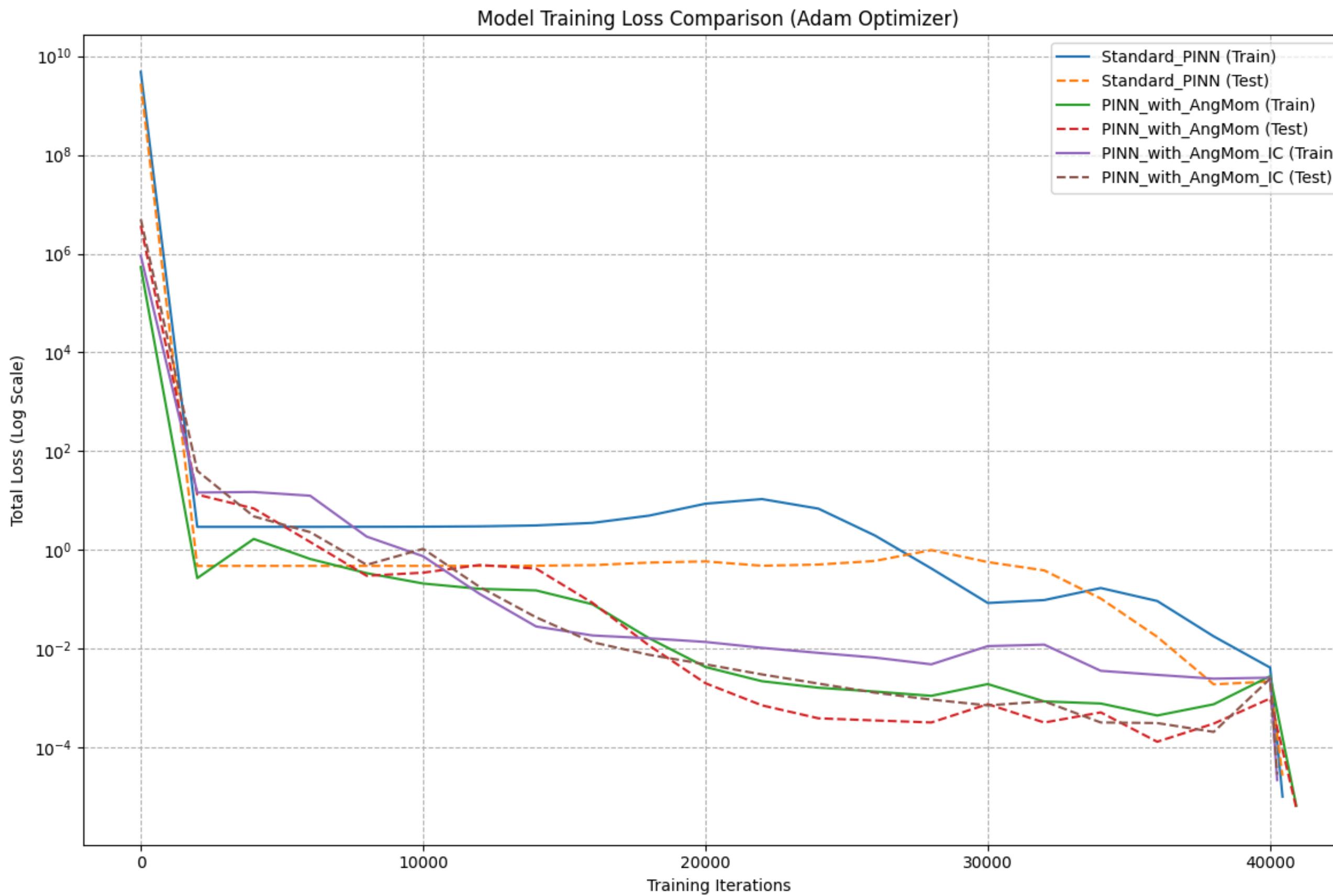
    print(f"--- Refining {name} with L-BFGS ---")
    model.compile("L-BFGS", loss_weights=weights, metrics=metrics) # Re-compile for L-BFGS
    losshistory, train_state = model.train()

    dde.saveplot(losshistory, train_state, issave=True, isplot=False, output_dir=name)

    t_pred_domain = np.linspace(t_start, t_end, 500).reshape(-1, 1)
    predictions[name] = model.predict(t_pred_domain)
```

4.1. Ejemplo 1: Kepler, movimiento planetario.

4. Casos de Estudio: PINNs en Acción



Clasica:

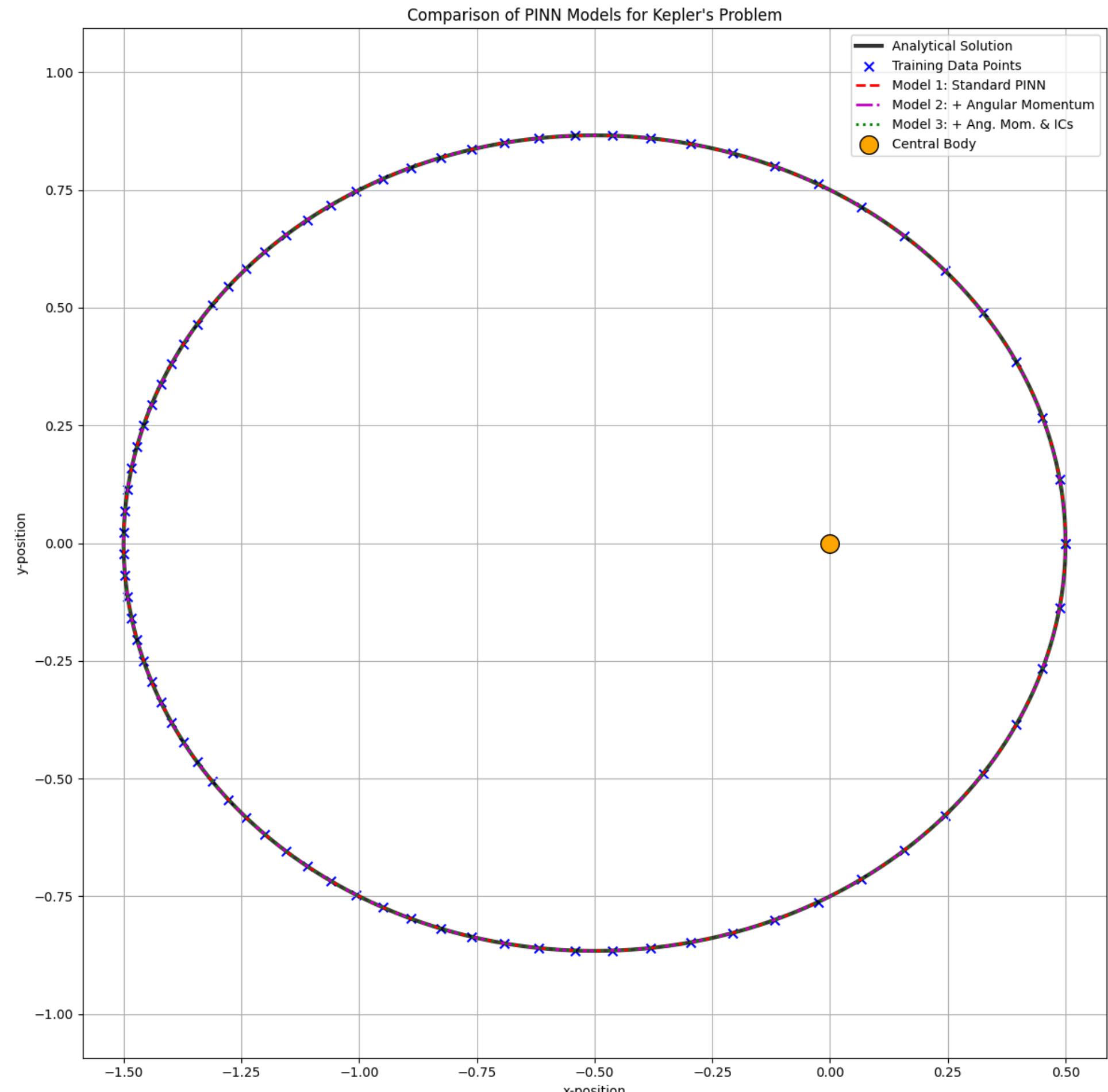
- Epocas: 40438
- Train Loss: 3.98e-05
- Test Loss: 3.77e-05
- Test Metric: 1.73
- Tiempo: 628.451687 s

Angular:

- Epocas: 40926
- Train Loss: 1.47e-05
- Test Loss: 1.35e-05
- Test Metric: 1.73
- Tiempo: 676.347726 s

CI:

- Epocas: 40247
- Train Loss: 6.21e-05
- Test Loss: 5.71e-05
- Test Metric: 1.73
- Tiempo: 729.631399 s



Ecuación del Calor 2D

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

- $T(x, y, t)$ es la temperatura en la posición (x, y) en el tiempo t .
- α es la difusividad térmica del material.

Condiciones de Frontera (Dirichlet):

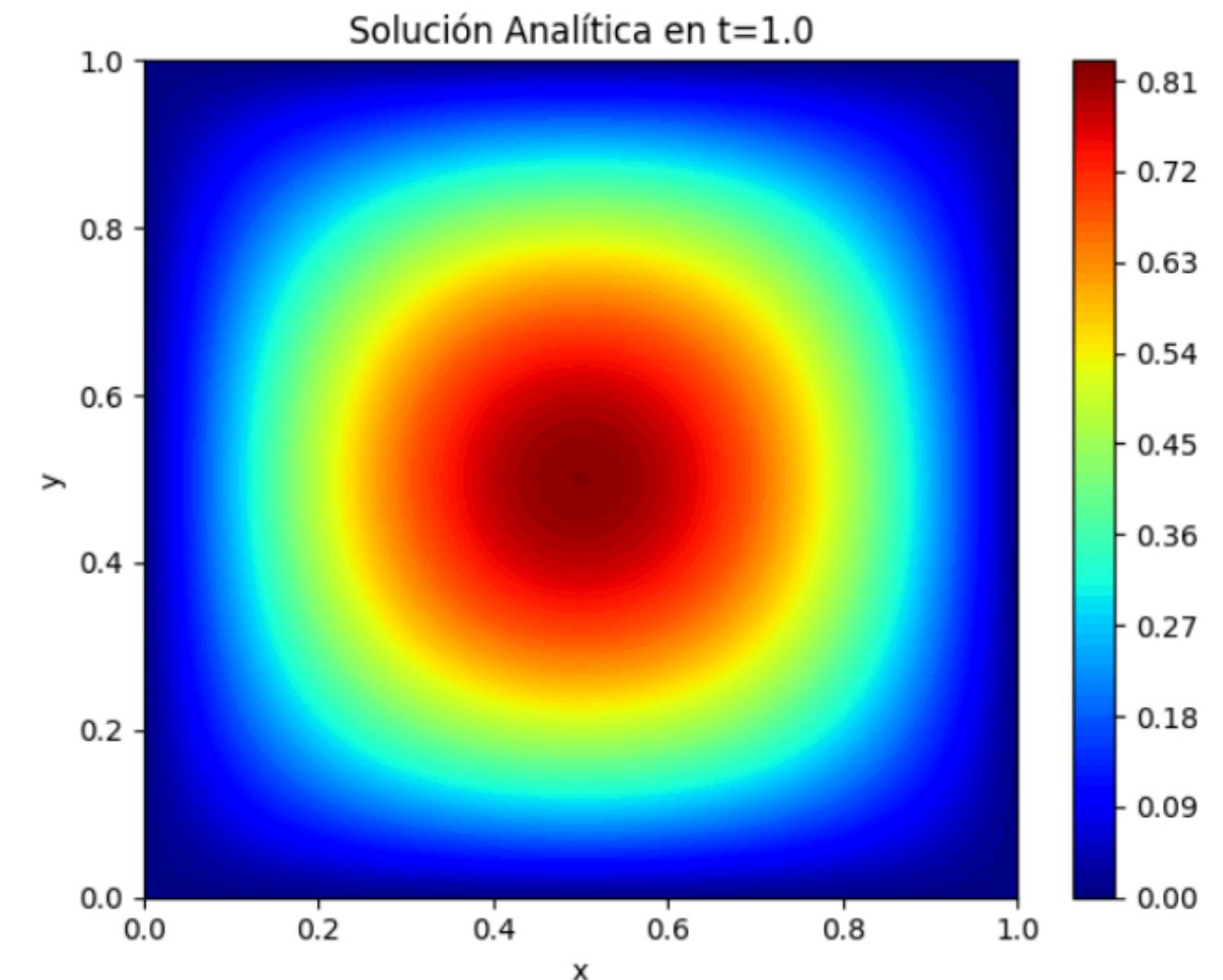
- $T(0, y, t) = 0$
- $T(1, y, t) = 0$
- $T(x, 0, t) = 0$
- $T(x, 1, t) = 0$

Dominio:

- Espacial: $x \in [0, 1], y \in [0, 1]$
- Temporal: $t \in [0, 1]$

Condición Inicial:

- $T(x, y, 0) = \sin(\pi x) \sin(\pi y)$



4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

Implementación:

Parámetros Globales

```
# Parámetros físicos y de simulación
ALPHA = 0.01      # Difusividad térmica
X_MIN, X_MAX = 0.0, 1.0
Y_MIN, Y_MAX = 0.0, 1.0
T_MIN, T_MAX = 0.0, 1.0 # Tiempo inicial y final

# Parámetros de la malla (para FD y predicción PINN)
NX = 51          # Número de puntos en x
NY = 51          # Número de puntos en y
NT_FD = 2000     # Número de pasos de tiempo para FD (estimado para estabilidad)

# Parámetros para la PINN
PINN_EPOCHS = 5000 # Idealmente >50000
PINN_LR = 1e-3
PINN_NUM_DOMAIN = 2500
PINN_NUM_BOUNDARY = 100
PINN_NUM_INITIAL = 100
PINN_LAYERS = [3] + [32] * 4 + [1] # [t, x, y] -> [T]
```

Sección 1: Método con Red Neuronal Informada por la Física (PINN)

```
: # 1. Definición de la EDP y Geometría para PINN
def pde(x, T):
    """Define la Ecuación Diferencial Parcial (EDP) para la ecuación de calor."""
    dT_dt = dde.grad.jacobian(T, x, i=0, j=0)
    dT_dxx = dde.grad.hessian(T, x, i=0, j=1)
    dT_dyy = dde.grad.hessian(T, x, i=0, j=2)
    return dT_dt - ALPHA * (dT_dxx + dT_dyy)

# Dominio geométrico y temporal
geom = dde.geometry.Rectangle([X_MIN, Y_MIN], [X_MAX, Y_MAX])
timedomain = dde.geometry.TimeDomain(T_MIN, T_MAX)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)
```

```
# Condiciones de frontera (BC)
def boundary_func(x, on_boundary):
    return on_boundary and not (np.isclose(x[0], T_MAX))

bc_zeros = dde.icbc.DirichletBC(
    geomtime,
    lambda x: 0.0, # T = 0 en las fronteras
    boundary_func
)
```

4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

```
# Condición inicial (IC)
def initial_condition_func(x):
    return np.sin(np.pi * x[:, 1:2]) * np.sin(np.pi * x[:, 2:3])

ic = dde.icbc.IC(
    geomtime,
    initial_condition_func,
    lambda _, on_initial: on_initial
)

# Definición del problema de datos para DeepXDE
data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc_zeros, ic],
    num_domain=PINN_NUM_DOMAIN,
    num_boundary=PINN_NUM_BOUNDARY,
    num_initial=PINN_NUM_INITIAL
)

# 2. Construcción y entrenamiento de la PINN
net = dde.nn.FNN(PINN_LAYERS, "tanh", "Glorot normal")
model = dde.Model(data, net)

# Compilar el modelo especificando el optimizador y la tasa de aprendizaje
model.compile("adam", lr=PINN_LR)
```

4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

```
# Entrenamiento
print("Iniciando entrenamiento de la PINN...")
start_time_pinn = time.time()
losshistory, train_state = model.train(epochs=PINN_EPOCHS)
end_time_pinn = time.time()
tiempo_pinn = end_time_pinn - start_time_pinn
print(f"Entrenamiento de la PINN completado en {tiempo_pinn:.2f} segundos.")

# --- INICIO DE LA CORRECCIÓN ---

# Guardar historial de pérdida
# En lugar de .epoch, se usa .steps
loss_pinn_epochs = losshistory.steps
# Usamos np.sum() para sumar las componentes de la pérdida (pde, ic, bc)
loss_pinn_values = np.sum(losshistory.loss_train, axis=1)

# --- FIN DE LA CORRECCIÓN ---
```

4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

```
Compiling model...
Building feed-forward neural network...
'build' took 0.054139 s

'compile' took 0.492432 s
```

Iniciando entrenamiento de la PINN...

Step	Train loss	Test loss	Test metric
0	[1.99e-03, 3.52e-02, 5.82e-02]	[1.99e-03, 3.52e-02, 5.82e-02]	[]
1000	[1.82e-06, 1.18e-06, 6.09e-07]	[1.82e-06, 1.18e-06, 6.09e-07]	[]
2000	[3.73e-07, 2.09e-07, 1.11e-07]	[3.73e-07, 2.09e-07, 1.11e-07]	[]
3000	[2.30e-07, 1.75e-07, 1.31e-07]	[2.30e-07, 1.75e-07, 1.31e-07]	[]
4000	[7.39e-08, 3.37e-08, 1.51e-08]	[7.39e-08, 3.37e-08, 1.51e-08]	[]
5000	[5.35e-08, 1.34e-08, 7.36e-09]	[5.35e-08, 1.34e-08, 7.36e-09]	[]

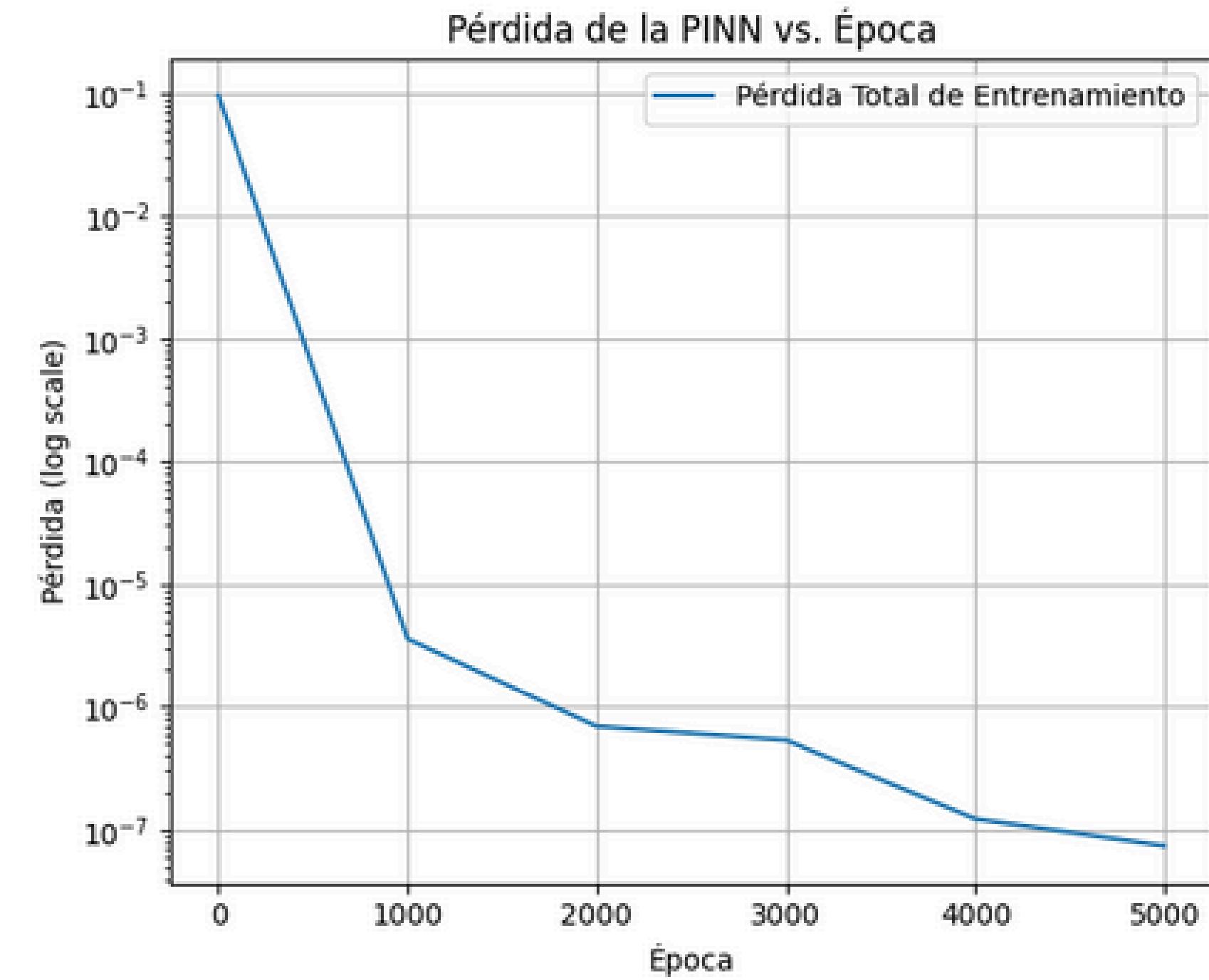
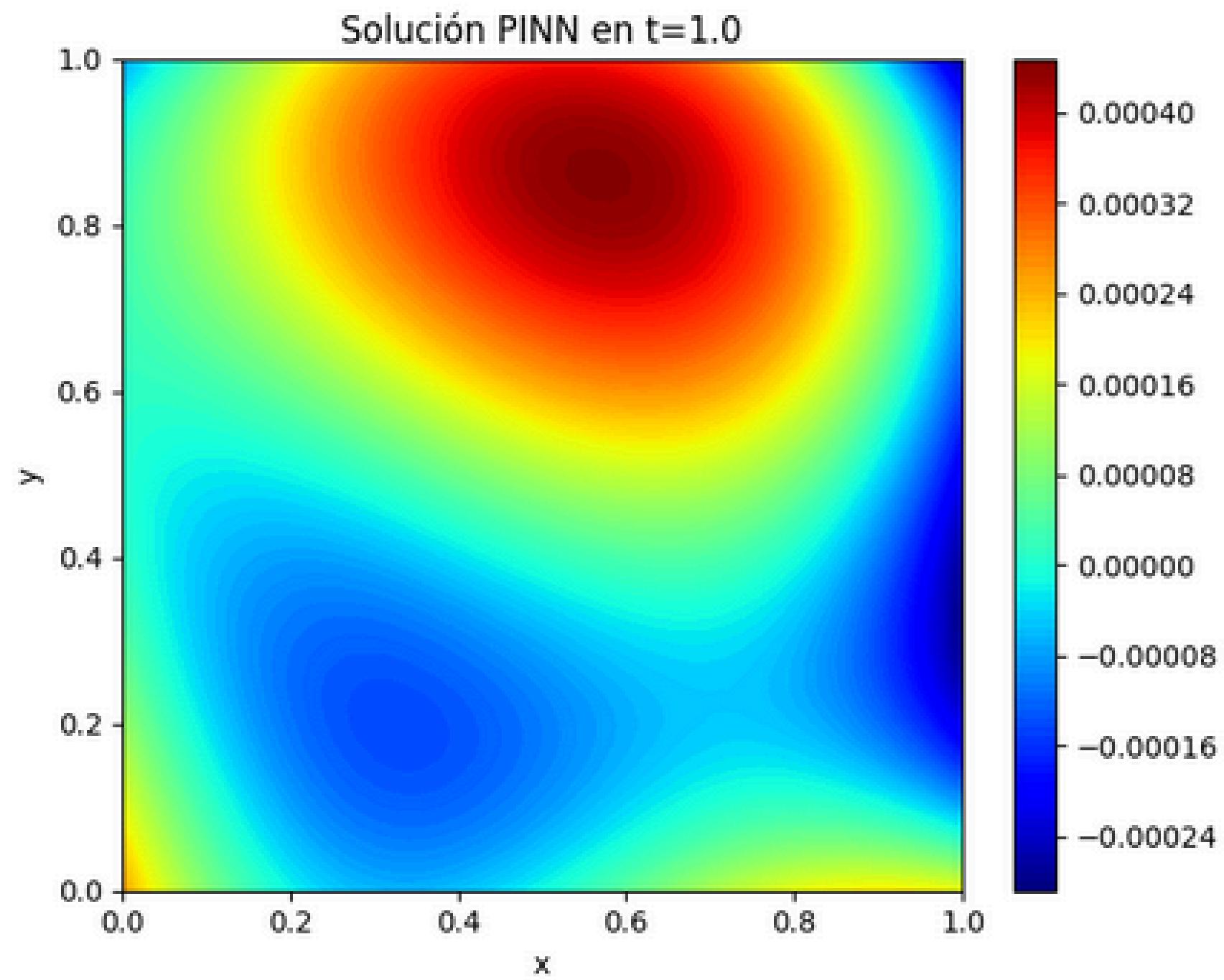
```
Best model at step 5000:
train loss: 7.42e-08
test loss: 7.42e-08
test metric: []
```

'train' took 56.639630 s

Entrenamiento de la PINN completado en 56.64 segundos.

4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción



Sección 2: Método Clásico (Diferencias Finitas)

```
# Implementación del Solucionador de Diferencias Finitas (FTCS)

# Parámetros para Diferencias Finitas
dx = (X_MAX - X_MIN) / (NX - 1)
dy = (Y_MAX - Y_MIN) / (NY - 1)
dt = (T_MAX - T_MIN) / NT_FD

# Constante de estabilidad (asegurarse de que sea < 0.25 para 2D FTCS)
stability_factor = ALPHA * dt * (1/dx**2 + 1/dy**2)
print(f"Factor de estabilidad para FD (FTCS): {stability_factor:.4f}")
if stability_factor >= 0.25:
    print("¡Advertencia! El esquema FTCS puede ser inestable con estos parámetros.")
    print(f"Se recomienda dt <= {0.24 / (ALPHA * (1/dx**2 + 1/dy**2)):.6f}")

# Inicializar la malla
T_fd = np.zeros((NX, NY))
x_fd_space = np.linspace(X_MIN, X_MAX, NX)
y_fd_space = np.linspace(Y_MIN, Y_MAX, NY)
X_fd_grid, Y_fd_grid = np.meshgrid(x_fd_space, y_fd_space)
```

4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

```
# Aplicar condición inicial T(x,y,0)
T_fd = T_analitica_func(X_fd_grid, Y_fd_grid, 0)

T_fd_new = np.copy(T_fd)

print("Iniciando simulación con Diferencias Finitas (FTCS)...")
start_time_fd = time.time()

# Bucle temporal FTCS
for n in range(NT_FD):
    # Vectorización para eficiencia
    T_fd_new[1:-1, 1:-1] = T_fd[1:-1, 1:-1] + ALPHA * dt * (
        (T_fd[2:, 1:-1] - 2*T_fd[1:-1, 1:-1] + T_fd[:-2, 1:-1]) / dx**2 +
        (T_fd[1:-1, 2:] - 2*T_fd[1:-1, 1:-1] + T_fd[1:-1, :-2]) / dy**2
    )
    T_fd = np.copy(T_fd_new)

end_time_fd = time.time()
tiempo_fd = end_time_fd - start_time_fd
print(f"Simulación con Diferencias Finitas completada en {tiempo_fd:.2f} segundos.")
```

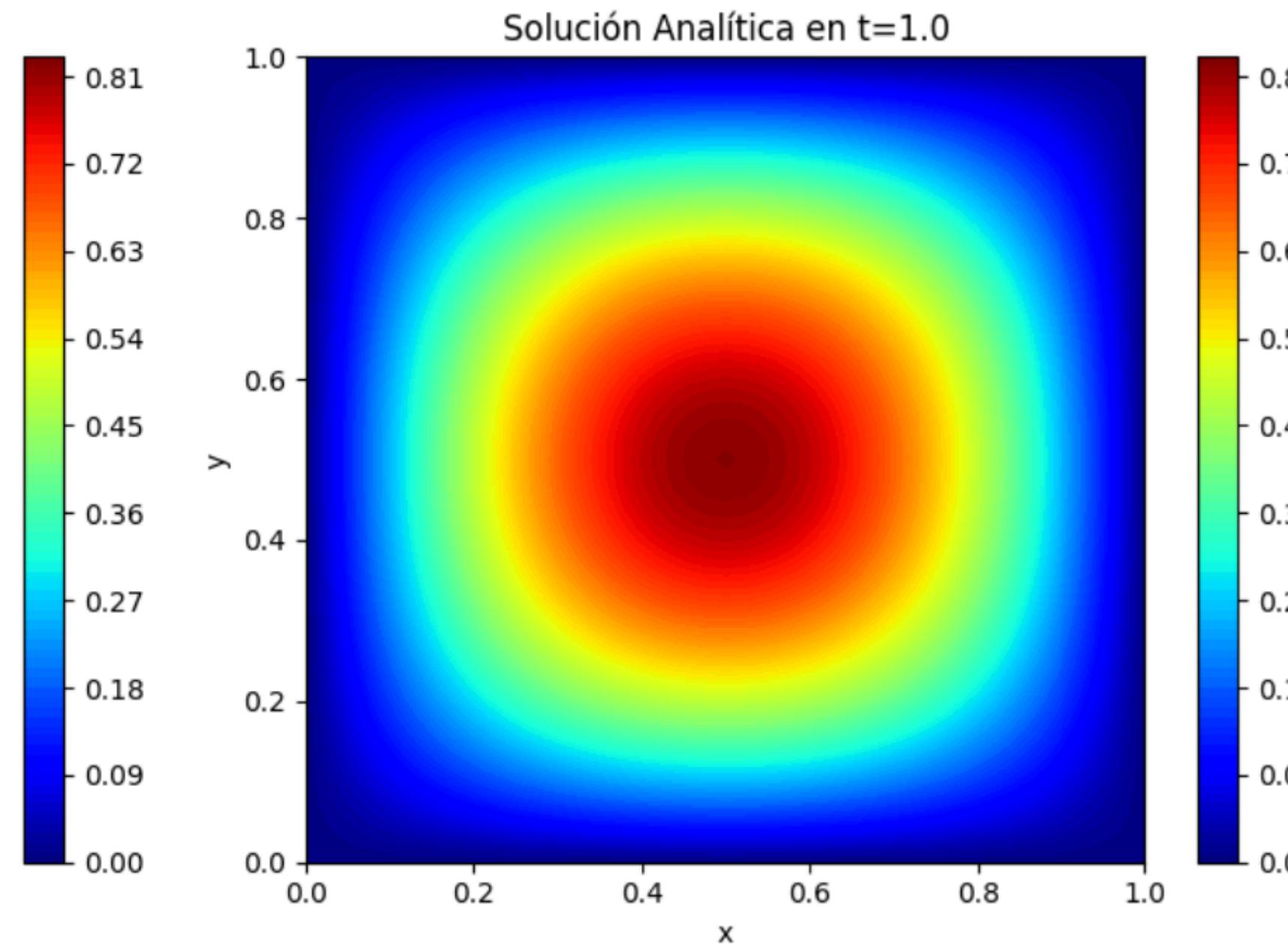
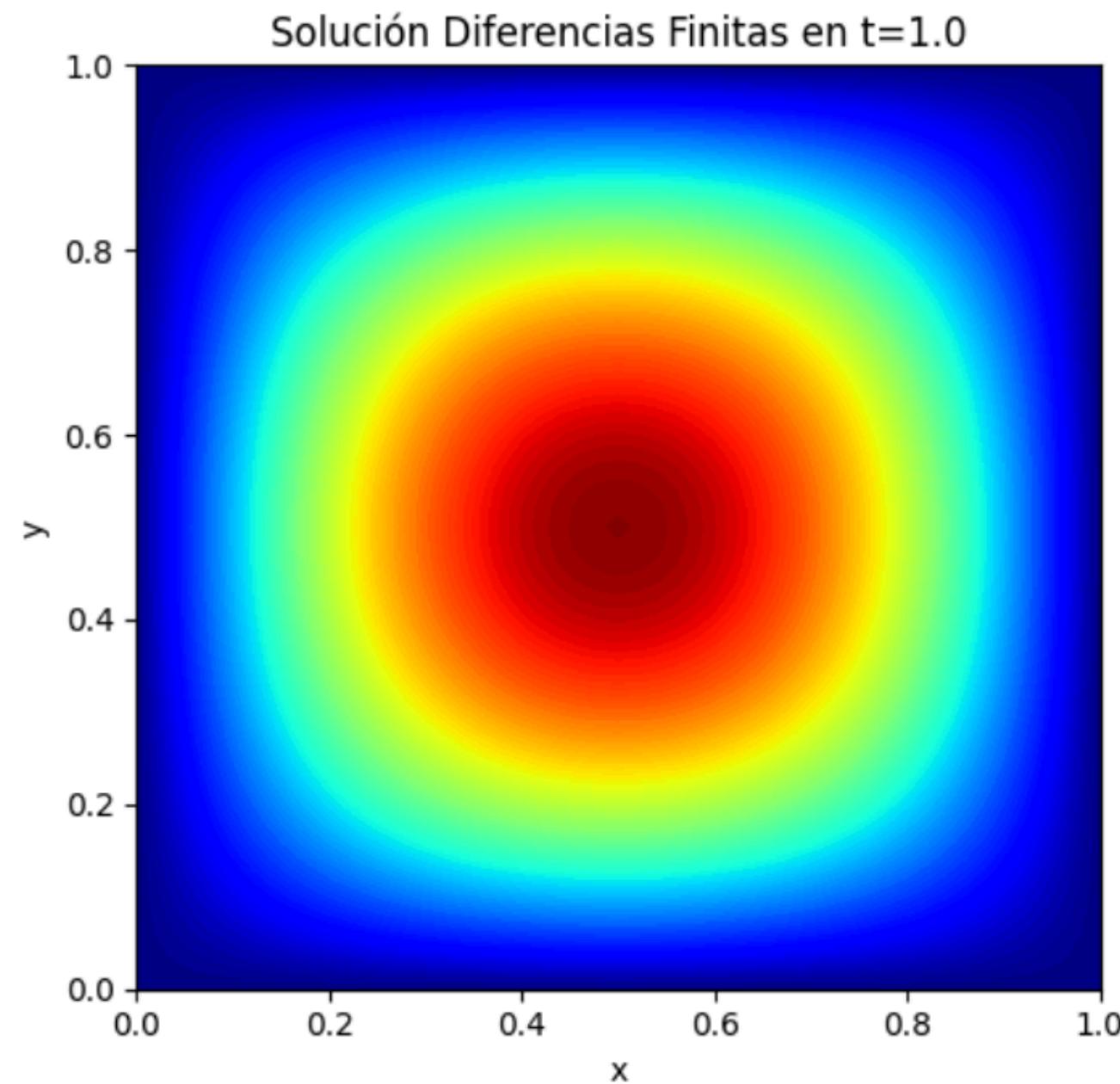
4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

Factor de estabilidad para FD (FTCS): 0.0250

Iniciando simulación con Diferencias Finitas (FTCS)...

Simulación con Diferencias Finitas completada en 0.08 segundos.

4. Casos de Estudio: PINNs en Acción



4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

4. Casos de Estudio: PINNs en Acción

--- Tiempos de Ejecución ---

Tiempo de entrenamiento/ejecución PINN: 56.64 segundos

Tiempo de simulación Diferencias Finitas: 0.08 segundos

--- Error Absoluto Medio (MAE) en $t=T_{MAX}$ ---

MAE PINN vs Analítica: 0.319494

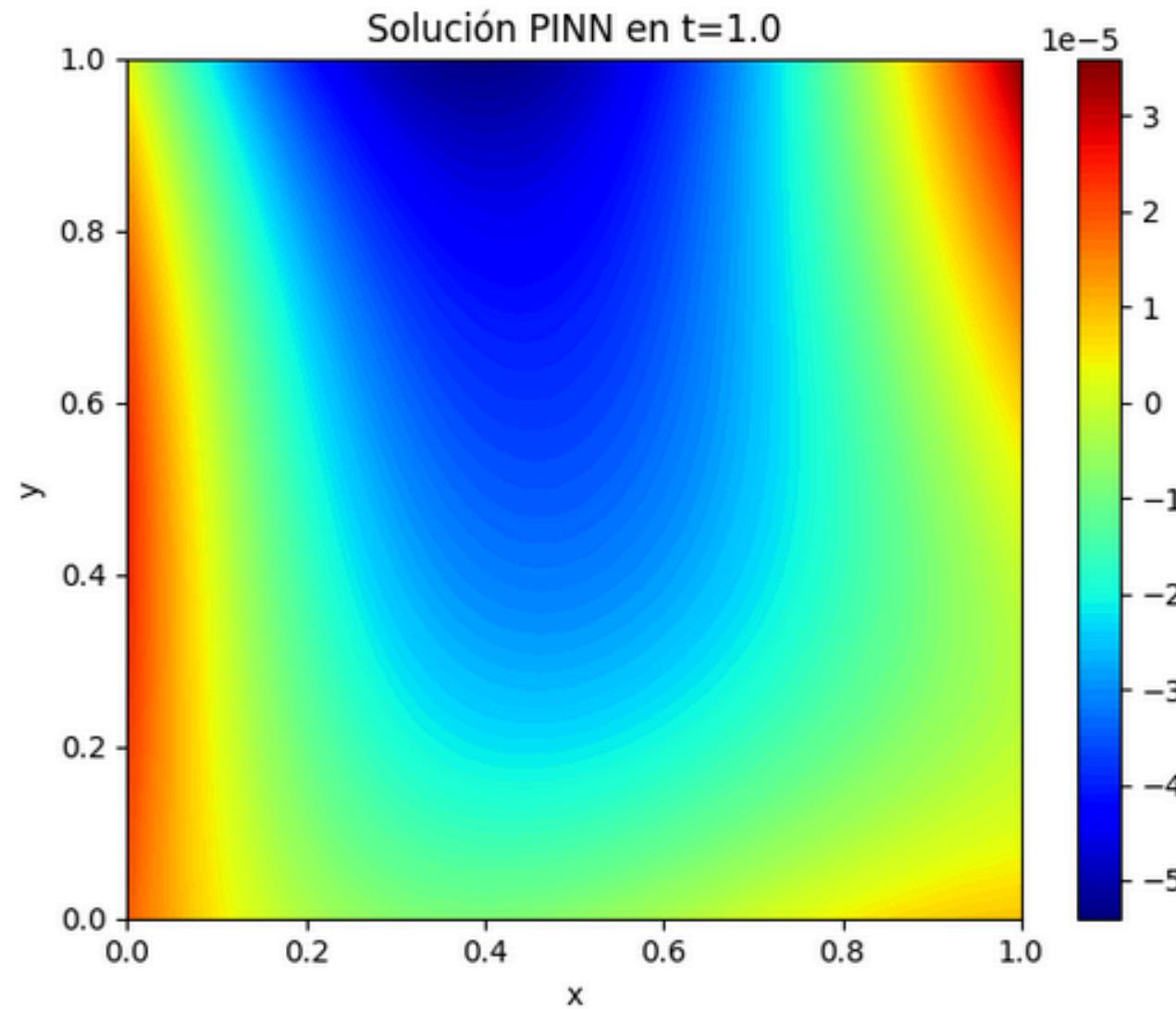
MAE FD vs Analítica: 0.000018

Las Diferencias Finitas fueron más precisas en este caso.

Recordar: PINN sin datos

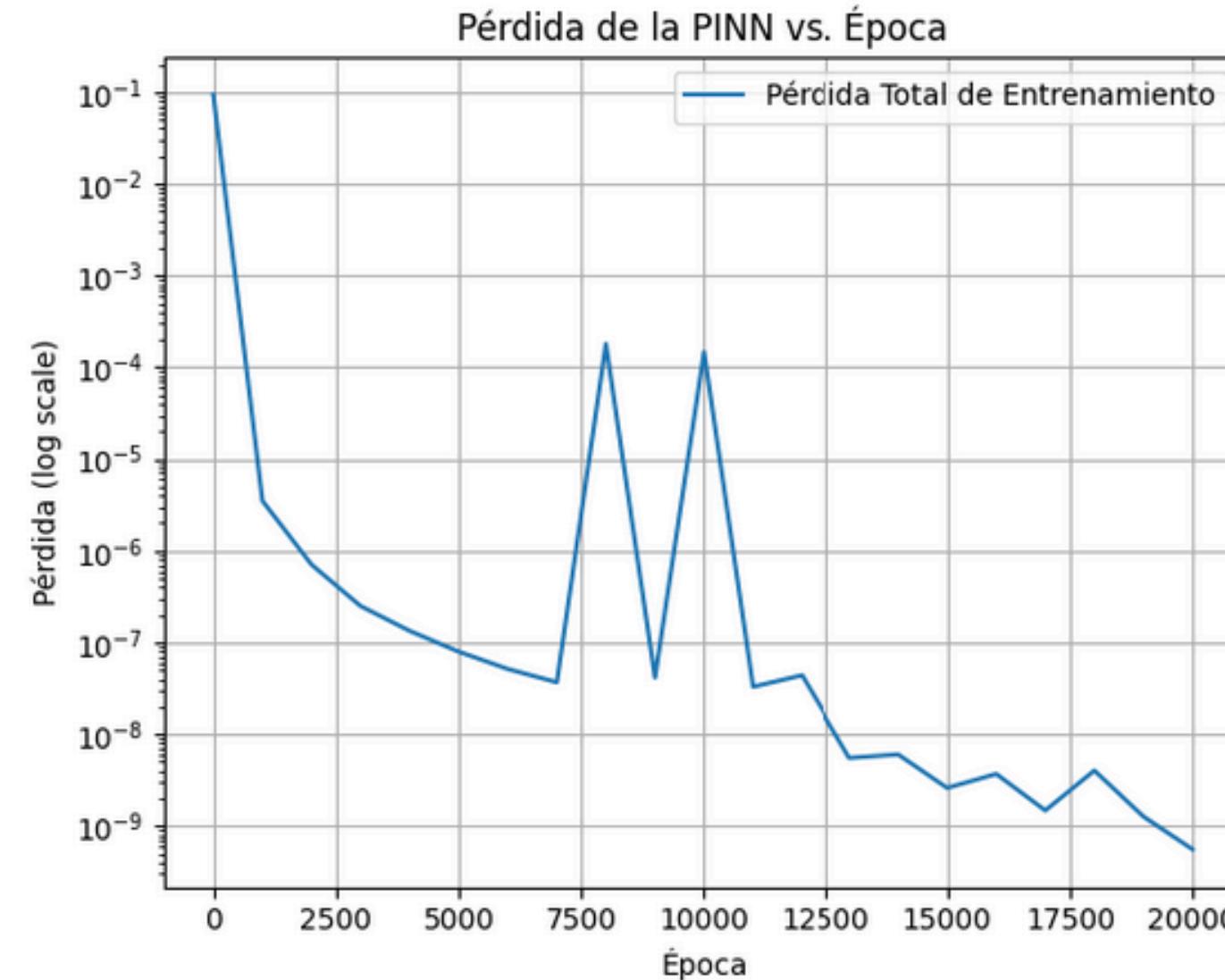
4.2. Ejemplo 2: El “Hola Mundo” - Ecuación del Calor 2D.

```
--- Tiempos de Ejecución ---  
Tiempo de entrenamiento/ejecución PINN: 632.55 segundos  
Tiempo de simulación Diferencias Finitas: 0.08 segundos  
  
--- Error Absoluto Medio (MAE) en t=T_MAX ---  
MAE PINN vs Analítica: 0.319574  
MAE FD vs Analítica: 0.000018  
  
Las Diferencias Finitas fueron más precisas en este caso.
```



4. Casos de Estudio: PINNs en Acción

```
# Parámetros para la PINN  
PINN_EPOCHS = 20000 # Idealmente >50000  
PINN_LR = 1e-3  
PINN_NUM_DOMAIN = 5000  
PINN_NUM_BOUNDARY = 500  
PINN_NUM_INITIAL = 500  
PINN_LAYERS = [3] + [32] * 4 + [1] # [t, x, y] -> [T]
```



Ecuacion de Poisson (2D)

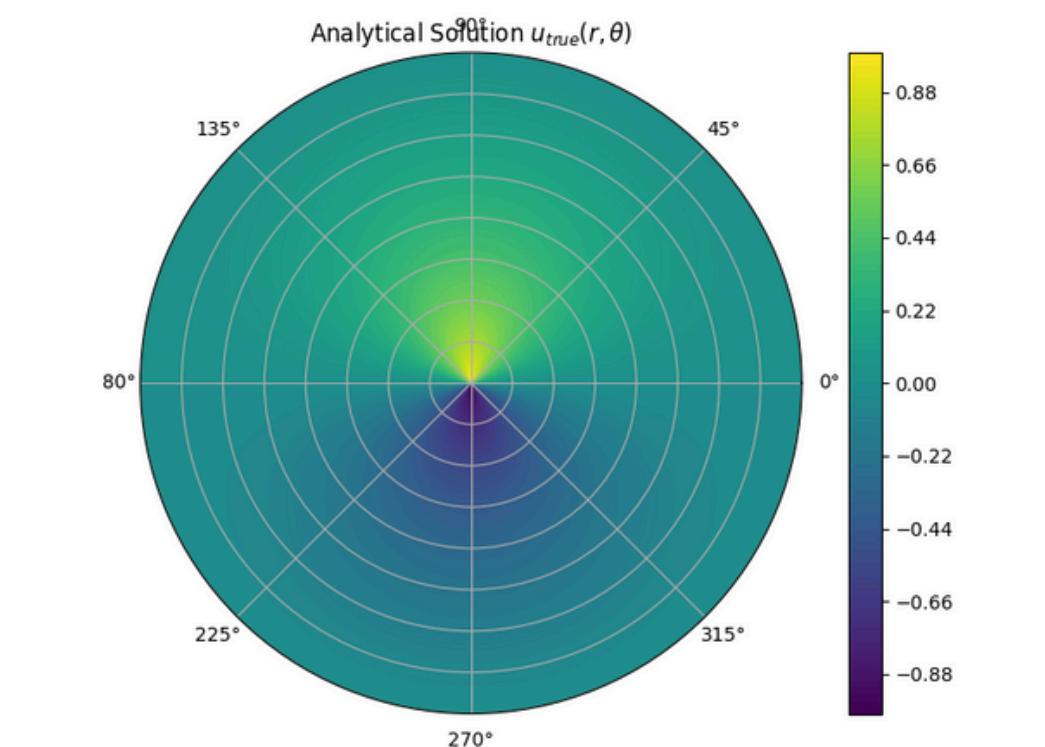
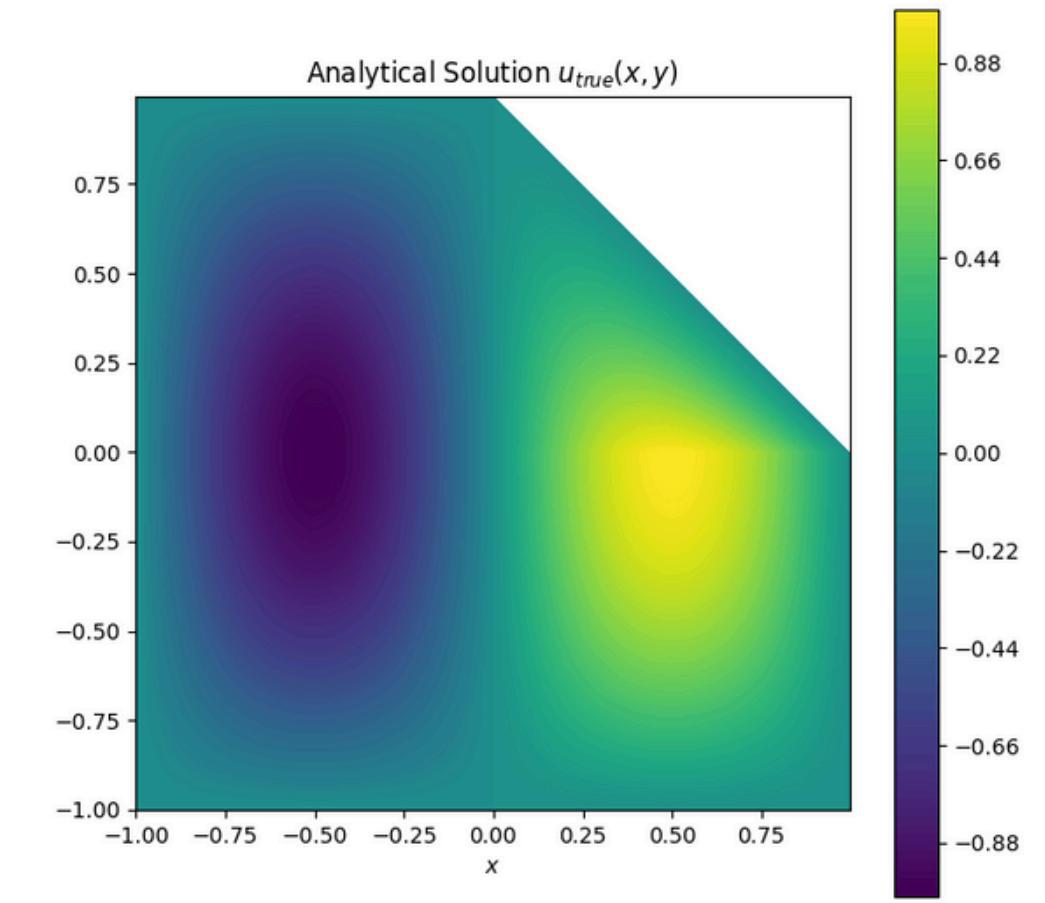
$$\nabla^2 f(\mathbf{x}) = \rho(x)$$

Analitica (tipo L)

$$f(x, y) = \sin(\pi x) \cos\left(\frac{\pi}{2}y\right)$$

Frontera (Anillo)

$$\begin{aligned} \rho(1) &= \cos(\theta) \\ \rho(3) &= 0 \end{aligned}$$



Implementacion

```
# Define the L-shaped domain using the vertices of a polygon.
# The domain is the square [-1, 1] x [-1, 1] with the quadrant [0, 1] x [-1, 0] removed.
# The re-entrant corner at (0, 0) introduces a singularity, a classic challenge.
vertices = [[-1, -1], [1, -1], [1, 0], [0, 0], [0, 1], [-1, 1]]
geom = dde.geometry.Polygon(vertices)
```

```
def boundary_func(x):
    """
    Computes the boundary values from the analytical solution.
    This handles the non-homogeneous part of the Dirichlet condition.
    """
    return analytical_solution(x)
```

4.2. Ejemplo 3: Ecuacion de Poisson

4. Casos de Estudio: PINNs en Acción

```
# Define the annulus geometry with inner radius r1=1 and outer radius r2=3.
disk_outer = dde.geometry.Disk(center=[0, 0], radius=3)
disk_inner = dde.geometry.Disk(center=[0, 0], radius=1)
geom = dde.geometry.CSGDifference(disk_outer, disk_inner)
```

```
def boundary_outer(x, on_boundary):
    return on_boundary and np.isclose(np.linalg.norm(x, axis=-1), 3)

def boundary_inner(x, on_boundary):
    return on_boundary and np.isclose(np.linalg.norm(x, axis=-1), 1)
```

4.2. Ejemplo 3: Ecuacion de Poisson

4. Casos de Estudio: PINNs en Acción

```
def pde(x, u):
    """
    Defines the residual of the Poisson equation:  $\nabla^2 u - f = 0$ .
    Args:
        x: A tensor of coordinates (x, y) with shape [N, 2].
        u: The network's output tensor u(x, y) with shape [N, 1].
    Returns:
        The PDE residual tensor.
    """
    # The Laplacian is computed by summing the second partial derivatives.
    # dde.grad.hessian(u, x, i=0, j=0) computes  $\partial^2 u / \partial x^2$ 
    # dde.grad.hessian(u, x, i=1, j=1) computes  $\partial^2 u / \partial y^2$ 
    dudx2 = dde.grad.hessian(u, x, i=0, j=0)
    dudy2 = dde.grad.hessian(u, x, i=1, j=1)
    laplacian = dudx2 + dudy2

    x_ = x[:, 0:1]
    y_ = x[:, 1:2]

    # Forcing function:  $f(x, y) = -(5/4)\pi^2 \sin(\pi x) \cos(\pi y / 2)$ 
    pi = np.pi
    forcing_term = -(5.0 / 4.0) * pi**2 * dde.backend.sin(pi * x_) * dde.backend.cos(pi * y_ / 2.0)

    return laplacian - forcing_term
```

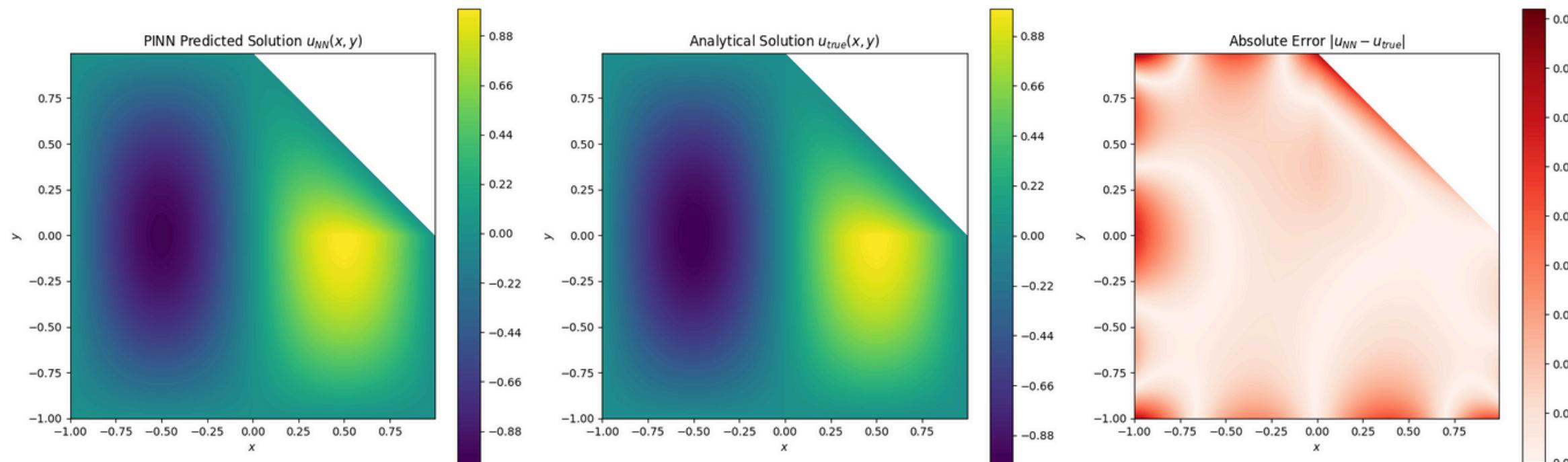
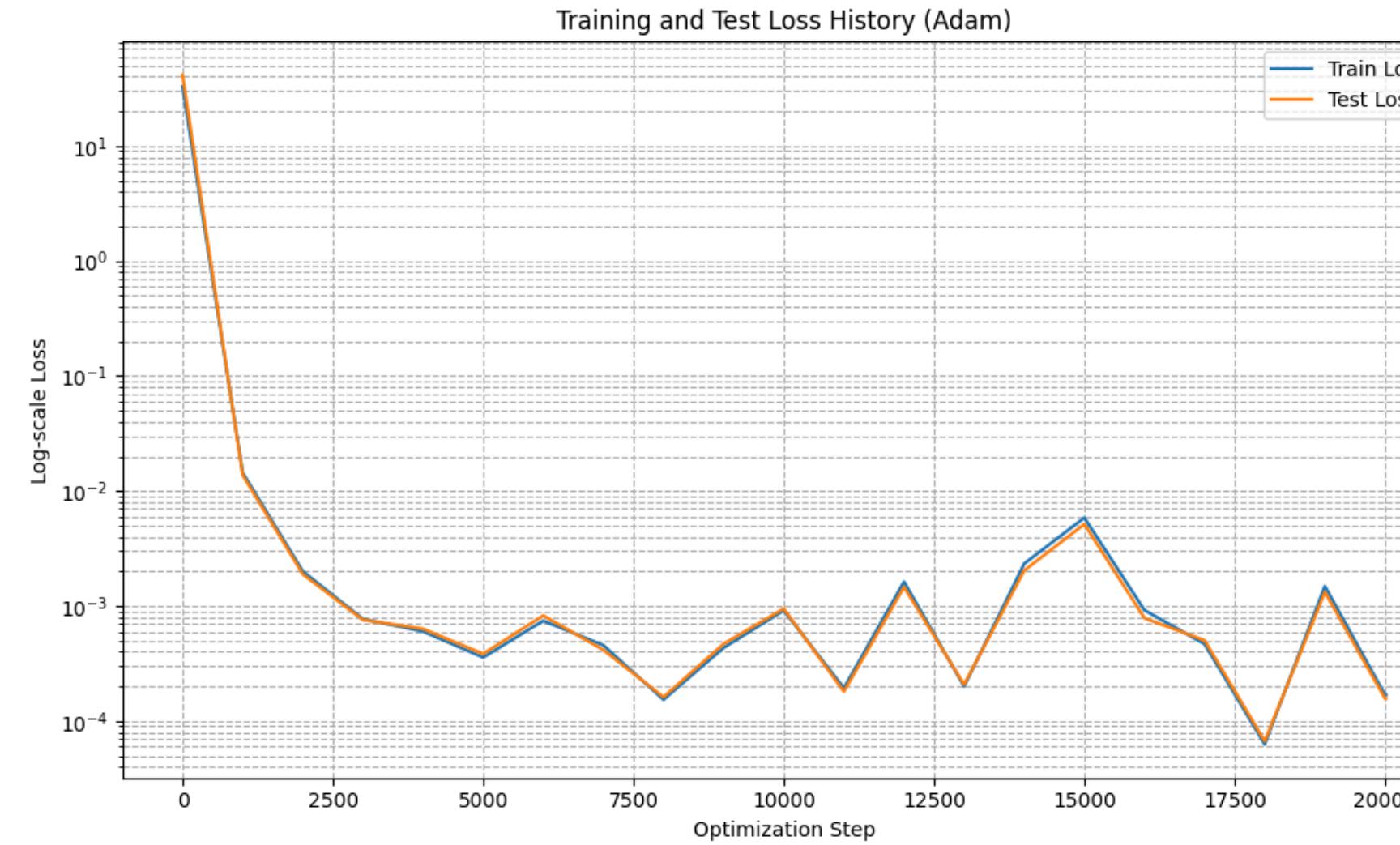
4.2. Ejemplo 3: Ecuacion de Poisson

4. Casos de Estudio: PINNs en Acción

```
# Define boundary condition value functions
def bc_inner_func(x):
    """
    Boundary condition on the inner circle (r=1): u = sin(theta).
    """
    y_ = x[:, 1:2]
    r_ = np.sqrt(x[:, 0:1]**2 + y_**2)
    # sin(theta) = y/r
    return y_ / r_

# Create the Dirichlet Boundary Condition objects.
# Outer boundary: u = 0
bc_outer = dde.DirichletBC(geom, lambda x: 0, boundary_outer)
# Inner boundary: u = sin(theta)
bc_inner = dde.DirichletBC(geom, bc_inner_func, boundary_inner)
```

4.2. Ejemplo 3: Ecuacion de Poisson

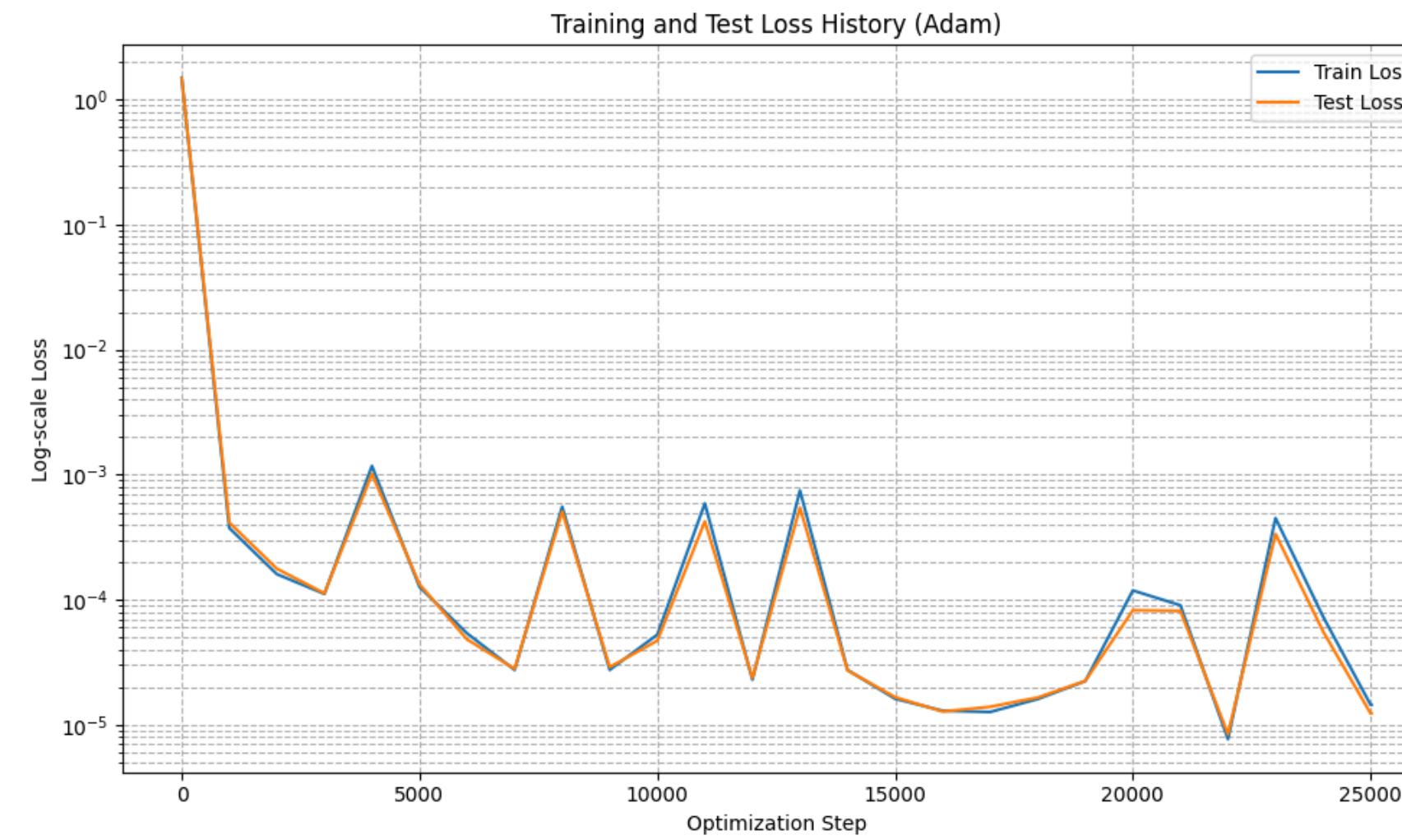


4. Casos de Estudio: PINNs en Acción

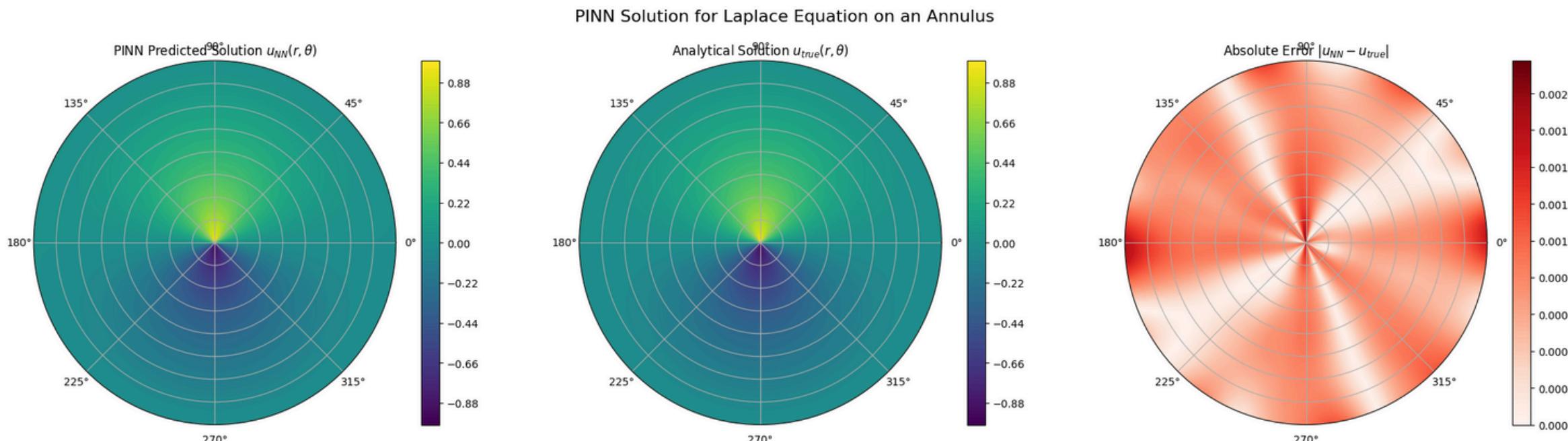
Épocas: 22000
Train Loss: 7.68e-06
Test Loss: 8.41e-06
Test Metric: [1.29e-03]
Tiempo: 1046.836716 s

4.2. Ejemplo 3: Ecuacion de Poisson

4. Casos de Estudio: PINNs en Acción



Épocas: 22000
Train Loss: 7.68e-06
Test Loss: 8.41e-06
Test Metric: 1.29e-03
Tiempo: 1046.836716 s



Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.

```
def Navier_Stokes_Equation(x, y):
    u = y[:, 0:1]
    v = y[:, 1:2]
    p = y[:, 2:3]
    du_x = dde.grad.jacobian(y, x, i=0, j=0)
    du_y = dde.grad.jacobian(y, x, i=0, j=1)
    du_t = dde.grad.jacobian(y, x, i=0, j=2)
    dv_x = dde.grad.jacobian(y, x, i=1, j=0)
    dv_y = dde.grad.jacobian(y, x, i=1, j=1)
    dv_t = dde.grad.jacobian(y, x, i=1, j=2)
    dp_x = dde.grad.jacobian(y, x, i=2, j=0)
    dp_y = dde.grad.jacobian(y, x, i=2, j=1)
    du_xx = dde.grad.hessian(y, x, component=0, i=0, j=0)
    du_yy = dde.grad.hessian(y, x, component=0, i=1, j=1)
    dv_xx = dde.grad.hessian(y, x, component=1, i=0, j=0)
    dv_yy = dde.grad.hessian(y, x, component=1, i=1, j=1)
    continuity = du_x + dv_y
    x_momentum = du_t + C1 * (u * du_x + v * du_y) + dp_x - C2 * (du_xx + du_yy)
    y_momentum = dv_t + C1 * (u * dv_x + v * dv_y) + dp_y - C2 * (dv_xx + dv_yy)
    return [continuity, x_momentum, y_momentum]
```

Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.

```
# Parameters to be identified
C1 = dde.Variable(0.0)
C2 = dde.Variable(0.0)

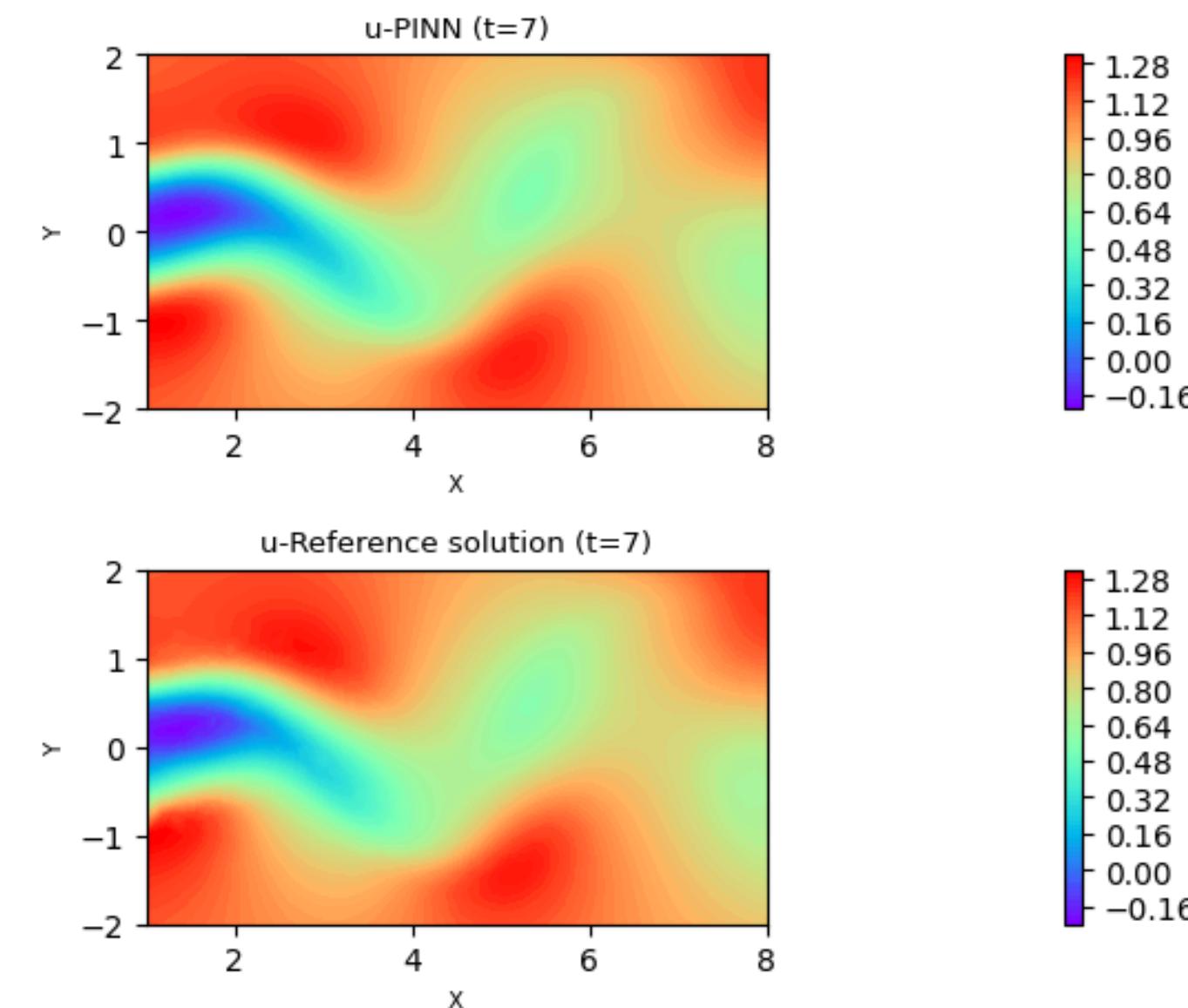
# Define Navier Stokes Equations (Time-dependent PDEs)
def Navier_Stokes_Equation(x, y):
    u = y[:, 0:1]
    v = y[:, 1:2]
    p = y[:, 2:3]
    du_x = dde.grad.jacobian(y, x, i=0, j=0)
    du_y = dde.grad.jacobian(y, x, i=0, j=1)
    du_t = dde.grad.jacobian(y, x, i=0, j=2)
    dv_x = dde.grad.jacobian(y, x, i=1, j=0)
    dv_y = dde.grad.jacobian(y, x, i=1, j=1)
    dv_t = dde.grad.jacobian(y, x, i=1, j=2)
    dp_x = dde.grad.jacobian(y, x, i=2, j=0)
    dp_y = dde.grad.jacobian(y, x, i=2, j=1)
    du_xx = dde.grad.hessian(y, x, component=0, i=0, j=0)
    du_yy = dde.grad.hessian(y, x, component=0, i=1, j=1)
    dv_xx = dde.grad.hessian(y, x, component=1, i=0, j=0)
    dv_yy = dde.grad.hessian(y, x, component=1, i=1, j=1)
    continuity = du_x + dv_y
    x_momentum = du_t + C1 * (u * du_x + v * du_y) + dp_x - C2 * (du_xx + du_yy)
    y_momentum = dv_t + C1 * (u * dv_x + v * dv_y) + dp_y - C2 * (dv_xx + dv_yy)
    return [continuity, x_momentum, y_momentum]
```

Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.

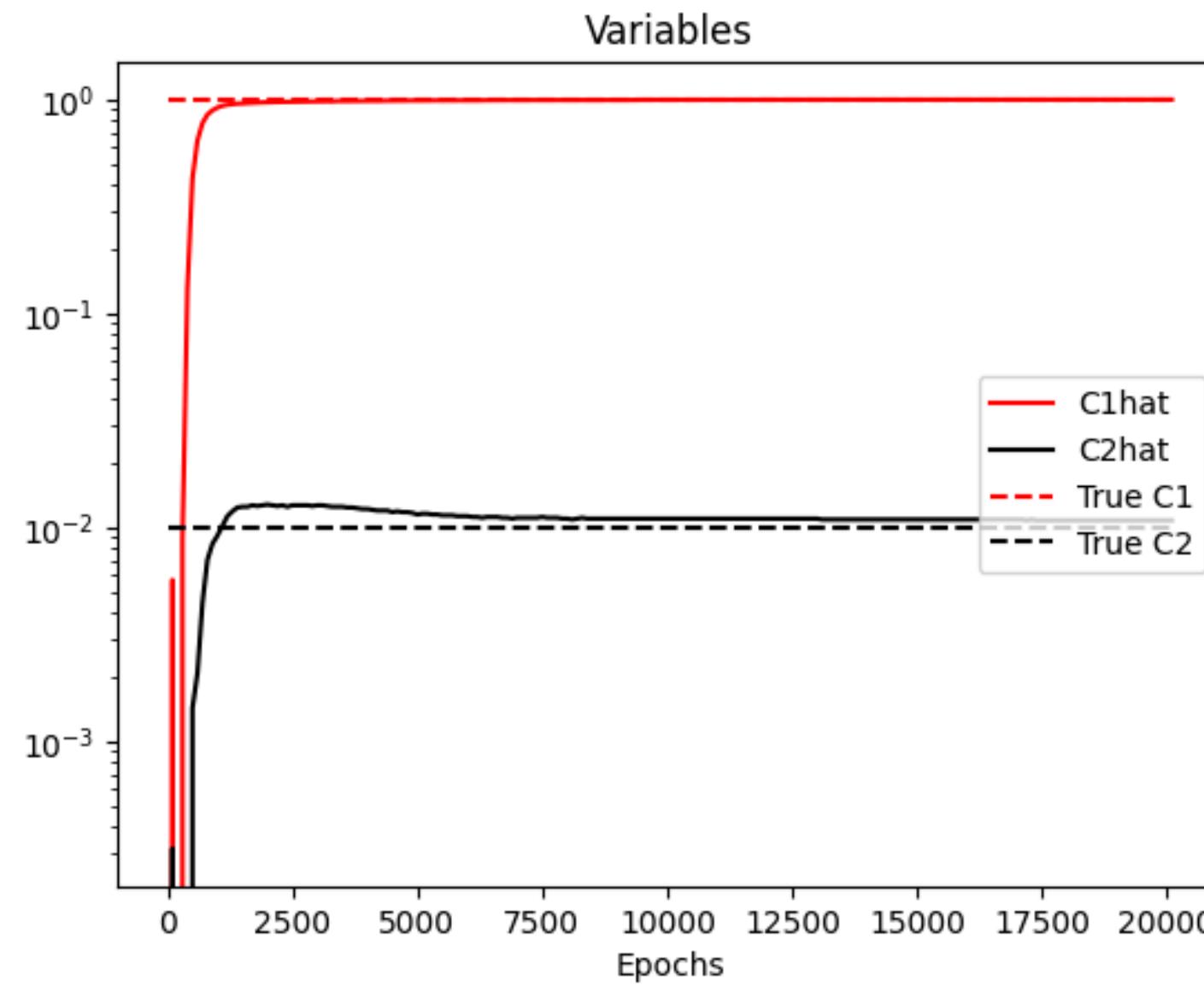
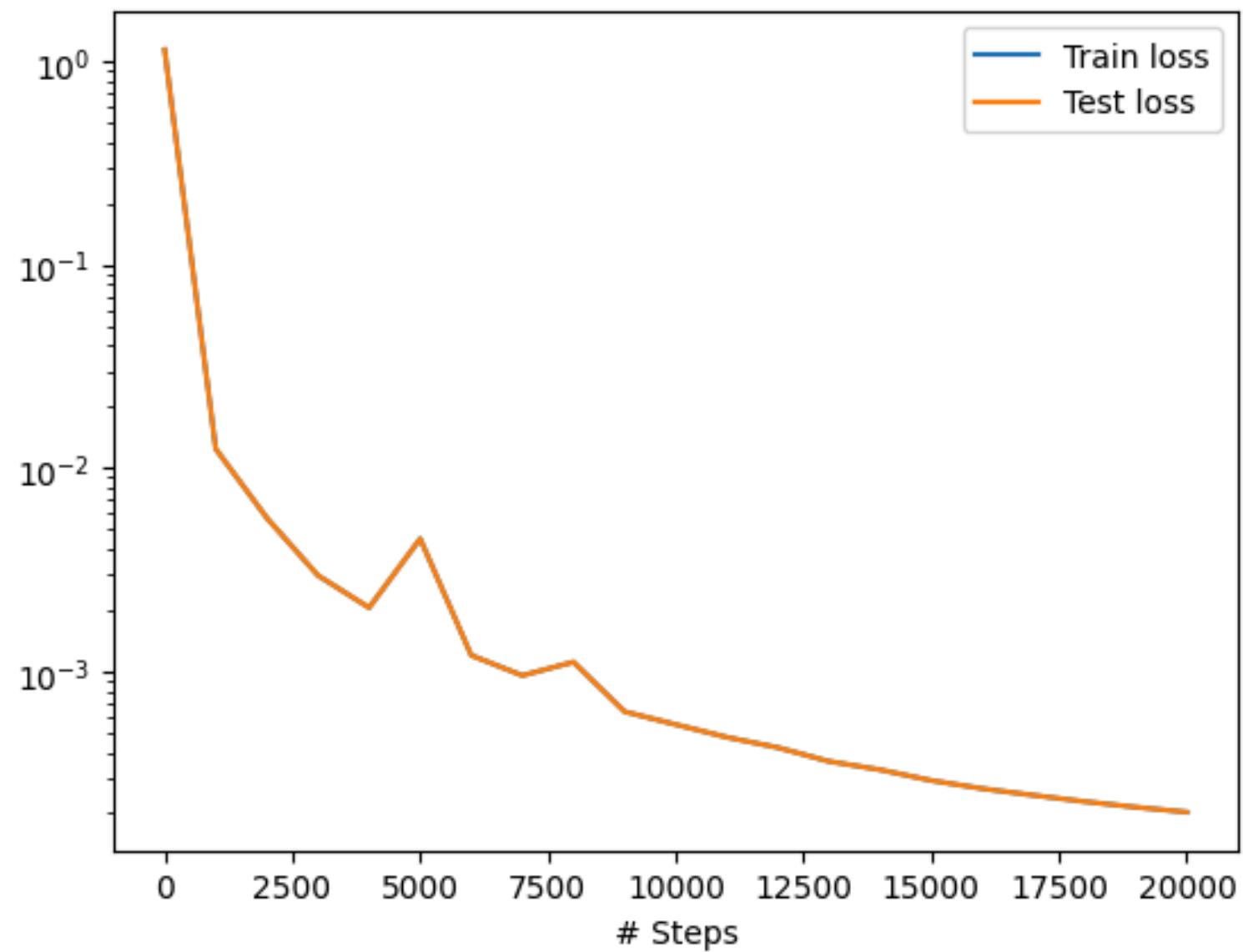
```
# Define Spatio-temporal domain
# Rectangular
Lx_min, Lx_max = 1.0, 8.0
Ly_min, Ly_max = -2.0, 2.0
# Spatial domain: X × Y = [1, 8] × [-2, 2]
space_domain = dde.geometry.Rectangle([Lx_min, Ly_min], [Lx_max, Ly_max])
# Time domain: T = [0, 7]
time_domain = dde.geometry.TimeDomain(0, 7)
# Spatio-temporal domain
geomtime = dde.geometry.GeometryXTime(space_domain, time_domain)
```

Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.

```
Best model at step 20000:  
train loss: 2.03e-04  
test loss: 2.03e-04  
test metric: []  
  
'train' took 483.036080 s
```



Ejemplo 4: La “Aplicación Estrella” - Problema Inverso de Navier-Stokes.



Ecuación de Klein-Gordon.

$$\frac{\partial^2 y}{\partial t^2} + \alpha \frac{\partial^2 y}{\partial x^2} + \beta y + \gamma y^k = -x \cos(t) + x^2 \cos^2(t)$$

with initial conditions

$$y(x, 0) = x, \quad \frac{\partial y}{\partial t}(x, 0) = 0$$

and Dirichlet boundary conditions

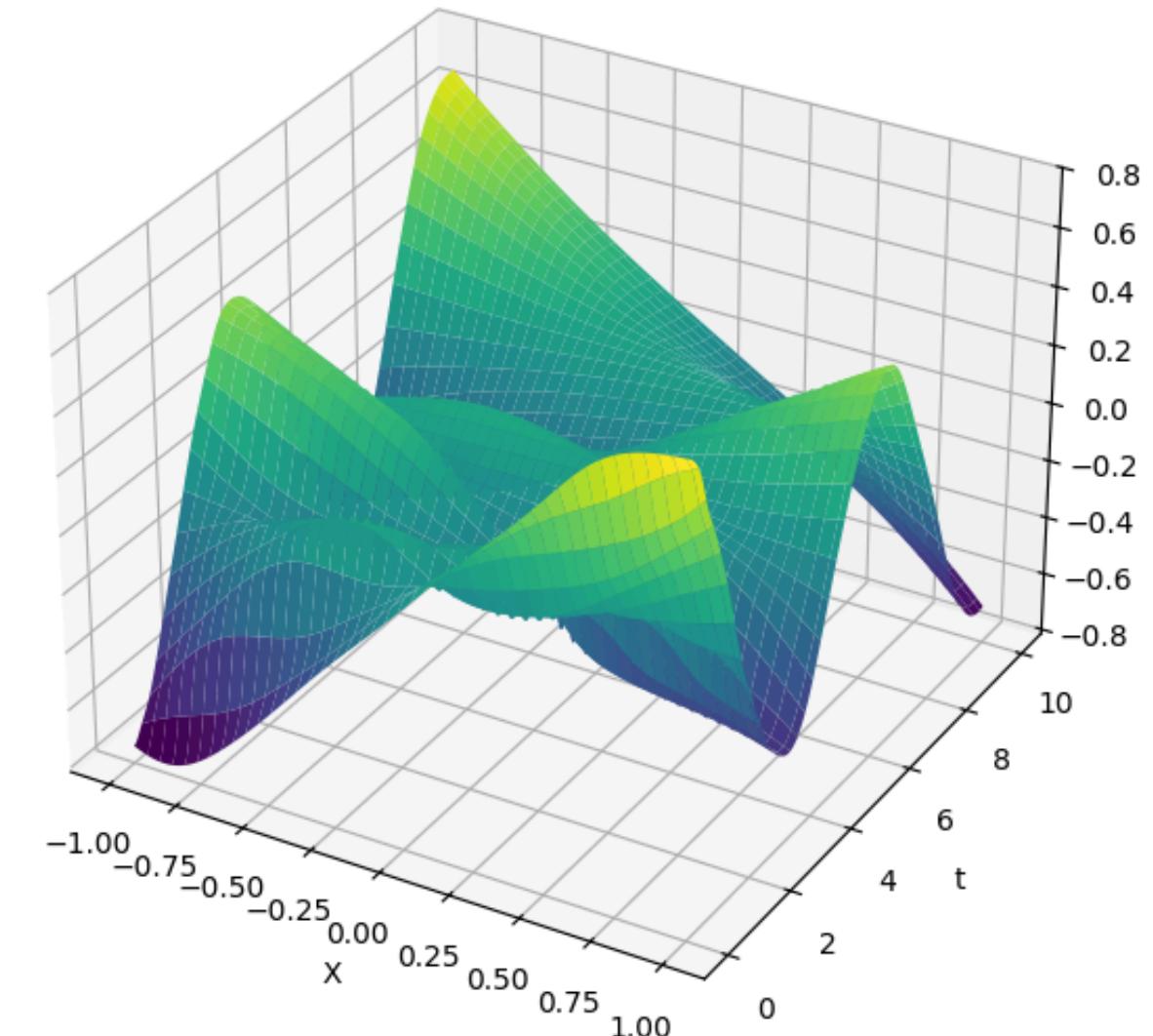
$$y(-1, t) = -\cos(t), \quad y(1, t) = \cos(t)$$

We also specify the following parameters for the equation:

$$\alpha = -1, \beta = 0, \gamma = 1, k = 2.$$

y es la función de onda

Solución de la EDP



4.5. Ejemplo 5: Ecuación de Klein-Gordon.

4. Casos de Estudio: PINNs en Acción

```
"""Backend supported: tensorflow.compat.v1, tensorflow, paddle"""
import deepxde as dde
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import griddata

geom = dde.geometry.Interval(-1, 1)
timedomain = dde.geometry.TimeDomain(0, 10)
geomtime = dde.geometry.GeometryXTime(geom, timedomain)

# Define sine function
if dde.backend.backend_name in ["tensorflow.compat.v1", "tensorflow"]:
    from deepxde.backend import tf

    cos = tf.math.cos
elif dde.backend.backend_name == "paddle":
    import paddle

    cos = paddle.cos
```

4.5. Ejemplo 5: Ecuación de Klein-Gordon.

4. Casos de Estudio: PINNs en Acción

```
def pde(x, y):
    alpha, beta, gamma, k = -1, 0, 1, 2
    dy_tt = dde.grad.hessian(y, x, i=1, j=1)
    dy_xx = dde.grad.hessian(y, x, i=0, j=0)
    x, t = x[:, 0:1], x[:, 1:2]
    return (
        dy_tt
        + alpha * dy_xx
        + beta * y
        + gamma * (y ** k)
        + x * cos(t)
        - (x ** 2) * (cos(t) ** 2)
    )

def func(x):
    return x[:, 0:1] * np.cos(x[:, 1:2])
```

4.5. Ejemplo 5: Ecuación de Klein-Gordon.

4. Casos de Estudio: PINNs en Acción

```
bc = dde.icbc.DirichletBC(geomtime, func, lambda _, on_boundary: on_boundary)
ic_1 = dde.icbc.IC(geomtime, func, lambda _, on_initial: on_initial)
ic_2 = dde.icbc.OperatorBC(
    geomtime,
    lambda x, y, _: dde.grad.jacobian(y, x, i=0, j=1),
    lambda _, on_initial: on_initial,
)

data = dde.data.TimePDE(
    geomtime,
    pde,
    [bc, ic_1, ic_2],
    num_domain=30000,
    num_boundary=1500,
    num_initial=1500,
    solution=func,
    num_test=6000,
)
```

4.5. Ejemplo 5: Ecuación de Klein-Gordon.

4. Casos de Estudio: PINNs en Acción

```
layer_size = [2] + [40] * 2 + [1]
activation = "tanh"
initializer = "Glorot uniform"
net = dde.nn.FNN(layer_size, activation, initializer)

model = dde.Model(data, net)
model.compile(
    "adam", lr=0.001, metrics=["l2 relative error"], decay=("inverse time", 3000, 0.9)
)
model.train(iterations=20000)
model.compile("L-BFGS", metrics=["l2 relative error"])
losshistory, train_state = model.train()
dde.saveplot(losshistory, train_state, issave=True, isplot=True)

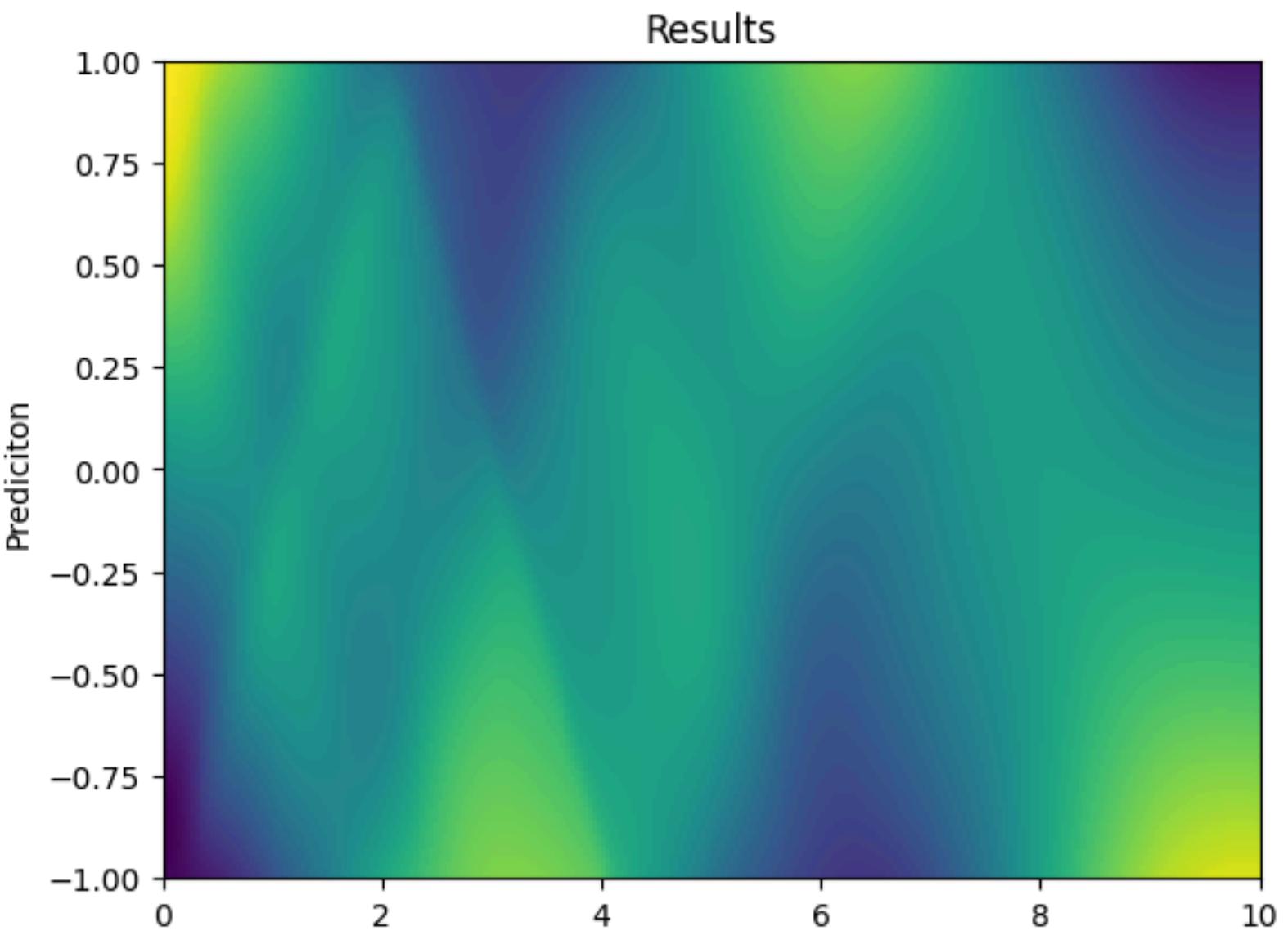
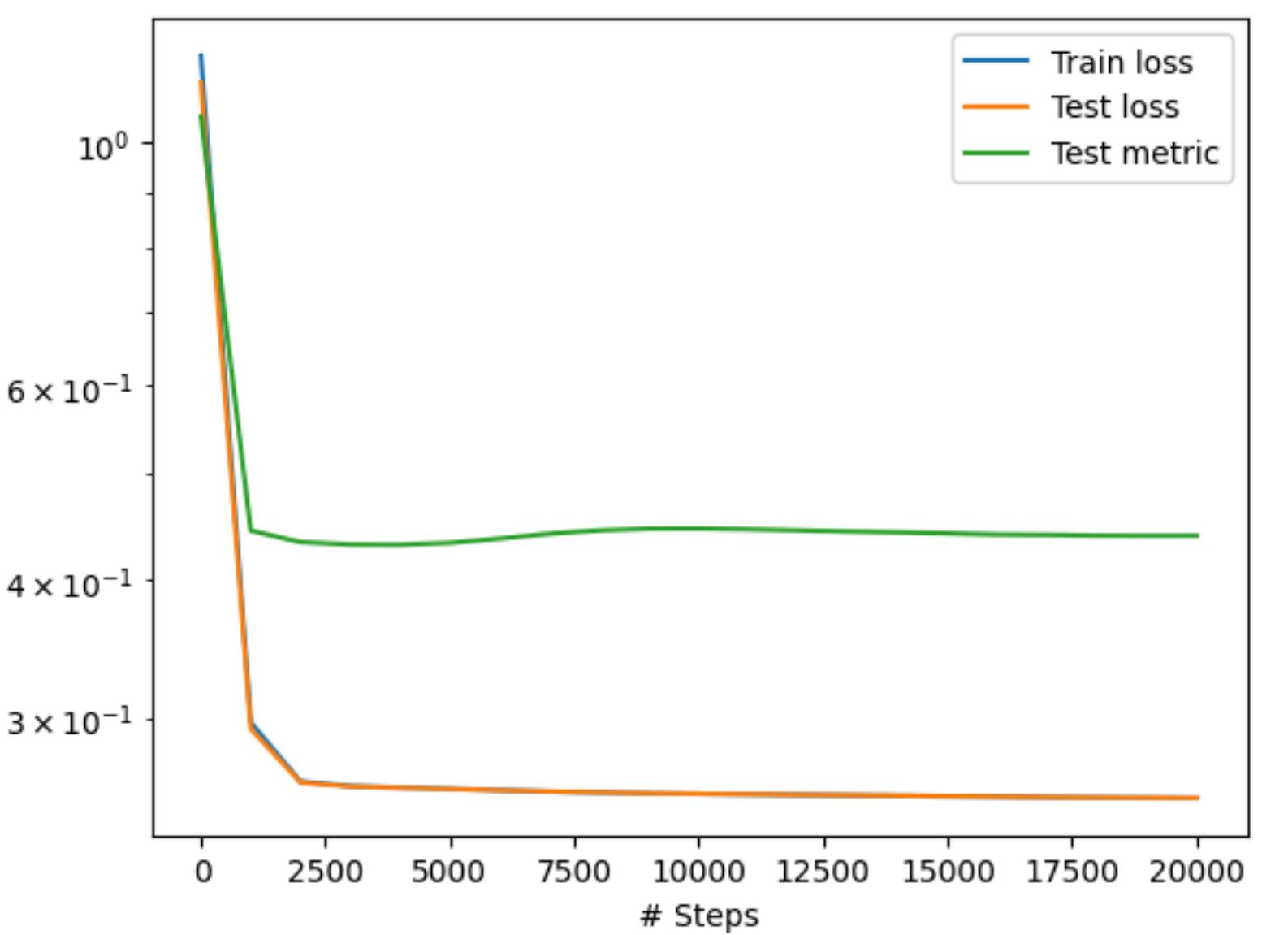
x = np.linspace(-1, 1, 256)
t = np.linspace(0, 10, 256)
X, T = np.meshgrid(x, t)

X_star = np.hstack((X.flatten()[:, None], T.flatten()[:, None]))
prediction = model.predict(X_star, operator=None)

v = griddata(X_star, prediction[:, 0], (X, T), method="cubic")
```

4.5. Ejemplo 5: Ecuación de Klein-Gordon.

4. Casos de Estudio: PINNs en Acción



5. Patologías de las PINN's

.

El Problema del "Sesgo Espectral" (Spectral Bias).

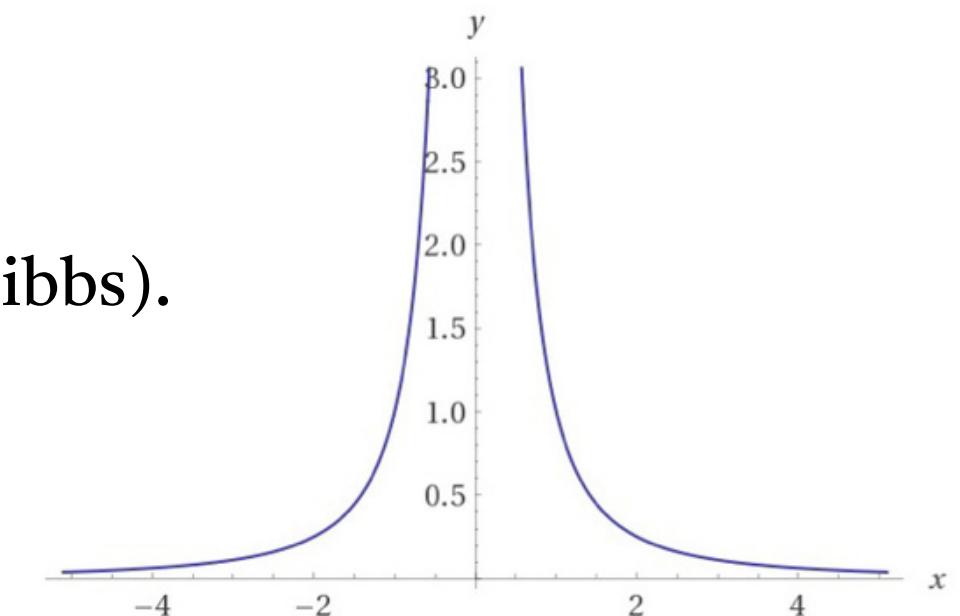
- ¿Qué es? Las redes neuronales aprenden primero las frecuencias bajas (patrones generales) y tardan mucho más en aprender las frecuencias altas (detalles finos).
- Causa: Es una tendencia natural del entrenamiento con descenso de gradiente.
- Problema en PINNs: Dificulta el aprendizaje de soluciones a Ecuaciones Diferenciales Parciales (EDPs) con:
 - Detalles finos y oscilaciones.
 - Gradientes abruptos (ej. ondas de choque).
 - Turbulencia.
- Consecuencia: La red "suaviza" las características complejas, fallando en capturar la física real del problema (ej. una onda de choque se ve como una transición difusa).
- Impacto: Es una barrera importante para aplicar PINNs a problemas complejos como en la dinámica de fluidos.

Inestabilidad en el Entrenamiento y Gradientes Desbalanceados.

- Origen: La función de pérdida de una PINN es una suma de múltiples términos (física, condiciones iniciales, de contorno).
- ¿Qué es? Durante el entrenamiento, las magnitudes de los gradientes de cada término de la pérdida pueden ser muy diferentes (varios órdenes de magnitud).
- Problema: El optimizador (ej. Adam) se ve dominado por el término con el gradiente más grande.
- Consecuencia: La red se enfoca en minimizar solo un objetivo (ej. una condición de contorno) e ignora los demás (ej. la física del problema).
- Resultado: El entrenamiento falla o converge a una solución físicamente incorrecta que no cumple todas las condiciones.
- Impacto: Hace que el entrenamiento de las PINNs sea inestable y muy sensible a la elección de hiperparámetros (como los pesos de cada término).

Manejo de Discontinuidades y Frentes de Choque.

- Problema fundamental: Las redes neuronales son aproximadores de funciones suaves por naturaleza (compuestas por funciones como la tangente hiperbólica, tanh).
- Limitación inherente: Su arquitectura no está diseñada para representar "saltos" o discontinuidades abruptas en una solución o sus derivadas.
- ¿Dónde es un obstáculo? En problemas físicos clave como:
 - Ondas de choque en dinámica de gases (Ecuaciones de Burgers, Euler).
 - Frentes de fase en ciencia de materiales.
 - Líneas de fractura en mecánica de sólidos.
- Consecuencias del fallo:
 - Oscilaciones no físicas cerca de la discontinuidad (similar al fenómeno de Gibbs).
 - Difuminación o "suavizado" excesivo del frente de choque.



Falta de Garantías Teóricas

- Problema central: Las PINNs carecen de un marco teórico y matemático riguroso.
- Contraste con métodos clásicos (como FEM): A diferencia de los métodos tradicionales, no hay garantías de:
 - Convergencia (que el entrenamiento encuentre la solución correcta).
 - Estabilidad del método.
 - Estimaciones fiables del error antes o después del cálculo.
- Consecuencia práctica: El éxito depende de la intuición, la experiencia y un tedioso proceso de prueba y error con los hiperparámetros.
- Barrera para la adopción: Esta falta de fiabilidad impide su uso en aplicaciones críticas (industria, ciencia) donde la cuantificación del error es esencial.

Conexión con la Física:

- Requisito clave: Para depurar y mejorar las PINNs se necesita una doble experiencia: en redes neuronales y en la física del problema.

6. ¿Medicina para las patologías?

Hamiltonian Neural Networks

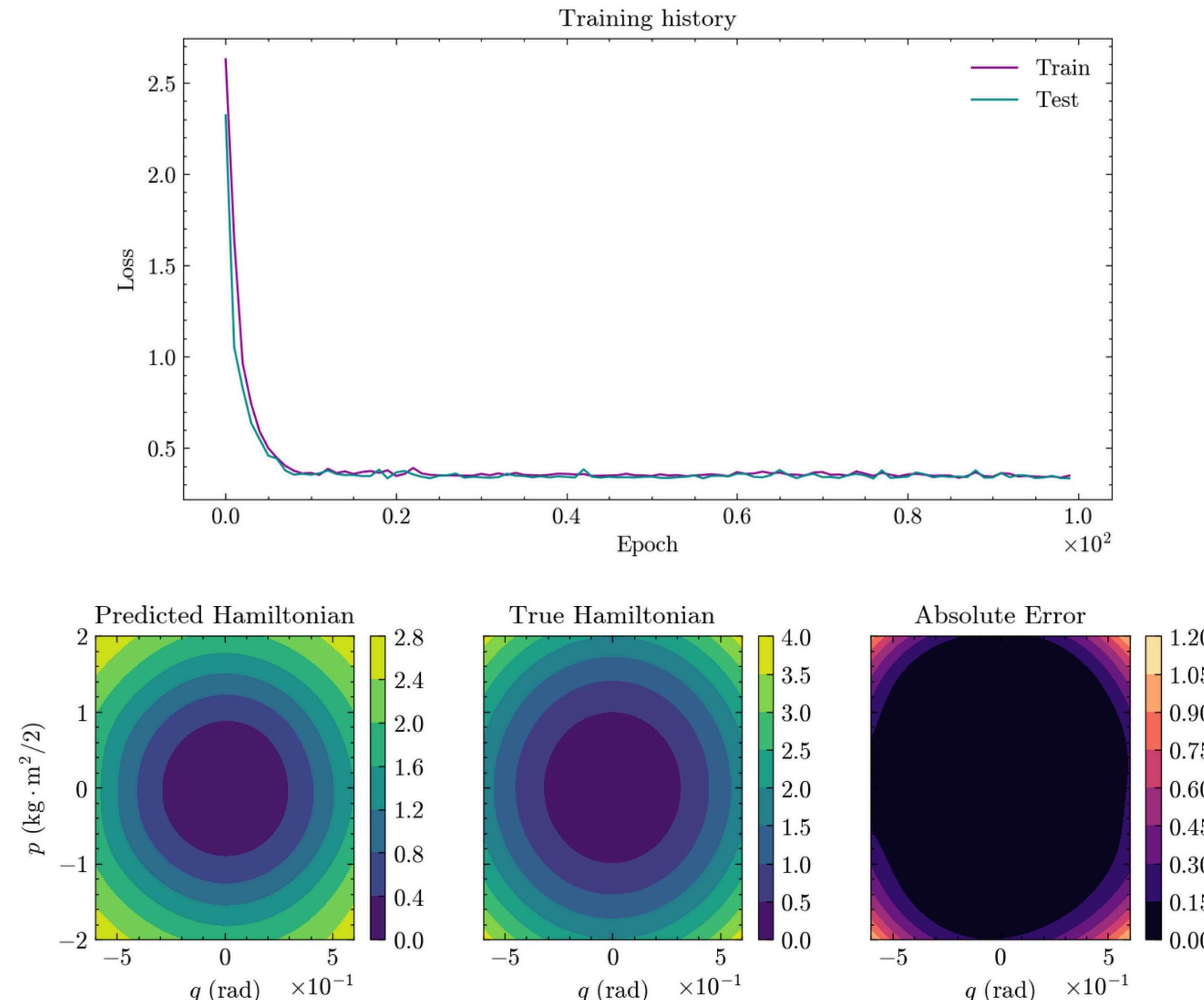
Hamilton's equations:

$$\dot{\mathbf{q}} = \frac{\partial H_{\theta}}{\partial \mathbf{p}} \quad \text{and} \quad \dot{\mathbf{p}} = -\frac{\partial H_{\theta}}{\partial \mathbf{q}}.$$

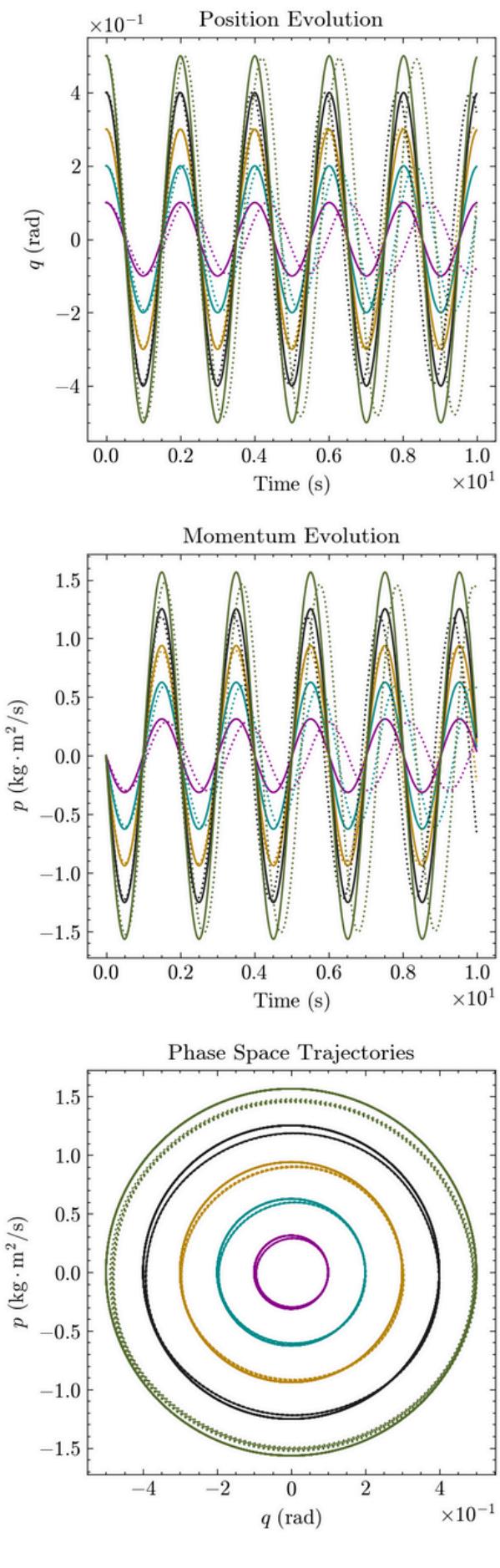
The loss function becomes:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \left\| \frac{\partial H_{\theta}}{\partial \mathbf{p}} \left(\mathbf{q}^{(i)}, \mathbf{p}^{(i)} \right) - \dot{\mathbf{q}}^{(i)} \right\|^2 + \left\| \frac{\partial H_{\theta}}{\partial \mathbf{q}} \left(\mathbf{q}^{(i)}, \mathbf{p}^{(i)} \right) + \dot{\mathbf{p}}^{(i)} \right\|^2.$$

6.1. Mitigación del Sesgo Espectral



6. ¿Medicina para las patologías?



Mitigación del Sesgo Espectral

- Mapeo de Características de Fourier (Fourier Feature Mapping)
- Funciones de Activación Adaptativas o Periódicas
- Arquitecturas Multiescala o Jerárquicas

Estrategias de Ponderación Adaptativa de la Pérdida (Adaptive Loss Weighting)

Objetivo: Automatizar y mejorar la estabilidad del entrenamiento de PINNs, evitando el ajuste manual de pesos.

- Ponderación Basada en Gradientes
- Ponderación Basada en Incertidumbre
- Ponderación Basada en el Historial de la Pérdida



Extensiones Arquitectónicas y Enfoques Híbridos

- XPINN (Extended PINN).

Se divide el dominio y se aplica una pinn a cada subdominio.

- PINO (Physics-Informed Neural Operator)

Se entrena para resolver el operador y evitar la necesidad de reentrenar para diferentes condiciones iniciales.

- Enfoques Híbridos PINN-FEM/FDM

Se busca establecer soluciones que usen elementos de los métodos numéricos clásicos.

Incorporación de Simetrías y Múltiples Soluciones

Simetrías de Lie

Se penaliza a la red si viola el grupo de simetrías de Lie.

Descubrimiento de Múltiples Soluciones

Se varían los inicializadores con tal de encontrar soluciones alternativas a las ED

7. Discusión Crítica y Conclusiones.

Resumen de Poderes.

- Sin malla.
- Geometrías complejas.
- Problemas inversos.
- Alta dimensionalidad.

¿Cuándo usar PINNs?

- Cuando SÍ: Problemas de alta dimensión, problemas inversos, problemas con geometrías complejas donde mallar es un dolor de cabeza, cuando se necesita una solución analítica y diferenciable.
- Cuándo NO (o con cuidado): Problemas de baja dimensión (1D, 2D) bien planteados donde existen solvers FEM/FDM ultra-optimizados y de alta precisión. Las PINNs pueden no ser tan precisas o rápidas en estos casos "fáciles".

Evaluación Crítica del Estado Actual.

las PINNs no deben ser vistas como un reemplazo universal de los métodos numéricos tradicionales como el FEM, sino como un complemento.

Existe una brecha significativa entre la visión idealizada de un solucionador de Ecuaciones Diferenciales Parciales basado en IA y la realidad práctica

Entrenar una PINN sigue siendo un "arte" que demanda una profunda experiencia tanto en aprendizaje automático como en la física del problema, limitando su aplicación a gran escala.

Nichos de Aplicación con Ventajas Claras.

- Problemas Inversos y de Identificación de Parámetros:
- Problemas de Alta Dimensionalidad:
- Modelado Sustituto (Surrogate Modeling):
- Problemas con Datos Faltantes o Ruidosos:

Direcciones Futuras

- Integración con Computación de Alto Rendimiento (HPC) y Computación Cuántica
- Hacia la Cuantificación de la Incertidumbre (UQ)
- El Potencial para el Descubrimiento de Nuevas Físicas
- Conclusión: La Visión de un Futuro Híbrido

GRACIAS :)

- Au-Yeung, R., Camino, B., Rathore, O., & Kendon, V. (2023). Quantum algorithms for scientific computing. <https://doi.org/10.48550/ARXIV.2312.14904>
- Greydanus, S., Dzamba, M., & Yosinski, J. (2019). Hamiltonian Neural Networks. arXiv. <https://doi.org/10.48550/ARXIV.1906.01563>
- Griffiths, J. (2024). AI in physics: selected studies in classical and quantum mechanics. Durham Theses, Durham University. <http://etheses.dur.ac.uk/15828/>
- Griffiths, J., Wrathmall, S. A., & Gardiner, S. A. (2025). Solving physics-based initial value problems with unsupervised machine learning. *Physical Review E*, 111(5), 055302. <https://doi.org/10.1103/physreve.111.055302>
- Hu, B., & McDaniel, D. (2023). Applying Physics-Informed Neural Networks to Solve Navier–Stokes Equations for Laminar Flow around a Particle. *Mathematical and Computational Applications*, 28(5), 102. <https://doi.org/10.3390/mca28050102>
- Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. *SIAM Review*, 63(1), 208–228. <https://doi.org/10.1137/19M1274067>
- Nasir, K., Reva, A., & Sekhar, J. (2025). Embedding Physics into Deep Learning: A Structured Review of Physics-Informed Neural Networks. <https://doi.org/10.20944/preprints202504.2577.v1>
- Rai, R., & Sahu, C. K. (2020). Driven by Data or Derived Through Physics? A Review of Hybrid Physics Guided Machine Learning Techniques With Cyber-Physical System (CPS) Focus. *IEEE Access*, 8, 71050–71073. <https://doi.org/10.1109/access.2020.2987324>
- Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378, 686–707. <https://doi.org/10.1016/j.jcp.2018.10.045>
- Raissi, Maziar, Perdikaris, P., Ahmadi, N., & Karniadakis, G. E. (2024). Physics-Informed Neural Networks and Extensions. arXiv. <https://doi.org/10.48550/ARXIV.2408.16806>
- Trahan, C., Loveland, M., & Dent, S. (2024). Quantum Physics-Informed Neural Networks. *Entropy*, 26(8), 649. <https://doi.org/10.3390/e26080649>