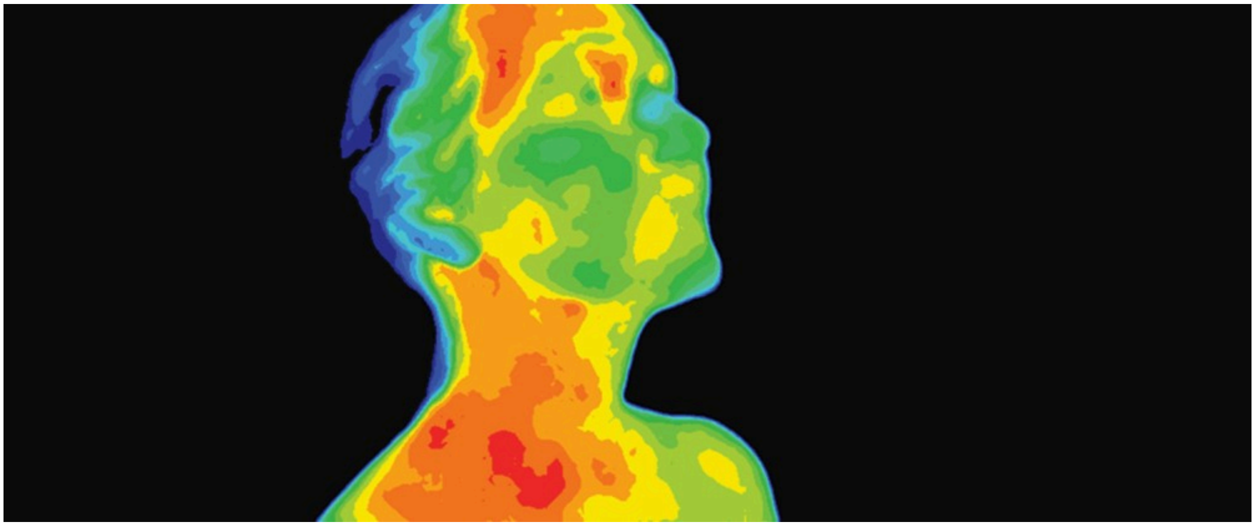


# Determining Temperature Profile of a Metal Plate using Physics Informed Neural Networks (PINNs) - [when one point at the corner of the plate is continuously heated]

**Recap form the previous Problem :**

This case is also solved using PINNs in a similar way as the previous problem [Link to the previous problem -
https://shorturl.at/n7nn9 ]

Below is a comprehensive, step-by-step explanation of the problem setup, governing equations, code structure, and final output analysis for the **corner-heated 2D plate** problem using PINNs.

---

## 1. Problem Statement

We have a 2D metal plate defined on the domain $\{(x, y) \mid 0 \leq x \leq 1,\ 0 \leq y \leq 1\}$. One **corner** of the plate (specifically, the point $(0, 0)$ is maintained at a high temperature of $100^\circ\mathrm{C}$, while **all other boundaries** are insulated (no heat flux). Initially, the entire plate is at a uniform temperature of $20^\circ\mathrm{C}$. We want to find the time evolution of the temperature field $T(x, y, t)$ for $t \in [0, 20]$ seconds.

---

## 2. Governing Equation and Boundary Conditions

## 2.1 Governing Equation

The transient heat conduction in the plate is described by the **2D heat equation**:

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

where

$\alpha \;=\; \frac{k}{\rho\, C_p}$ is the thermal diffusivity, with:

- $\rho$ = density,
- $Cp$ = specific heat capacity,
- $k$ = thermal conductivity.

## 2.2 Boundary Conditions

1. **Heated Corner** $(x = 0, y = 0)$

   A Dirichlet boundary condition is imposed at the corner:

   $$T(0, 0, t) \;=\; 100^{\circ}\mathrm{C}.$$

2. **Insulated Boundaries**

   The rest of the boundary (i.e., all edges except the corner point) is assumed to have **zero heat flux**. Mathematically, that means the derivative of $T$ in the outward normal direction is zero. For example, along $x = 0$ with $y > 0$:

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

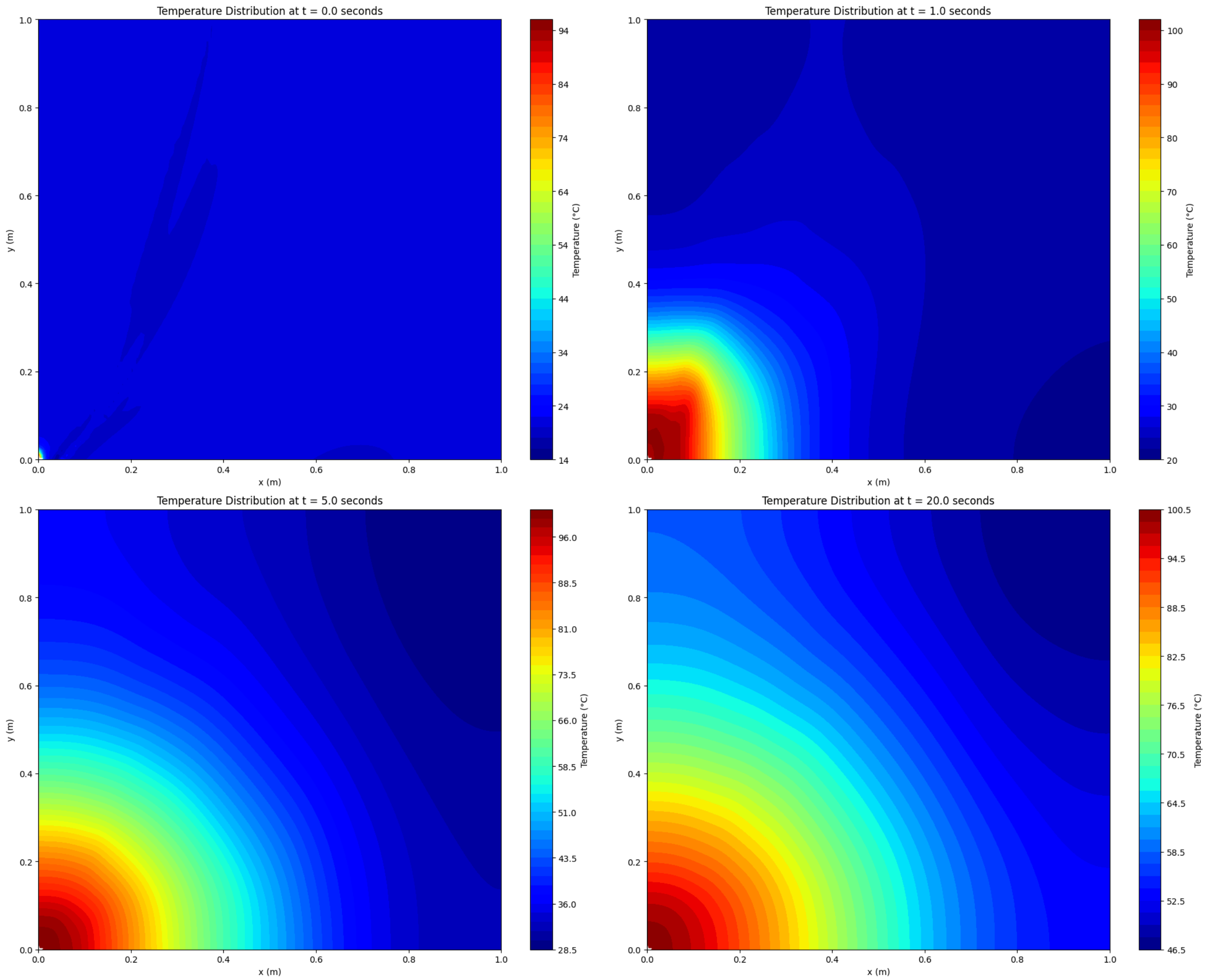
   and similarly for $x = 1, y = 0$, and $y = 1$.

## 2.3 Initial Condition

At $t = 0$, the plate is at a uniform temperature of $20^{\circ}\mathrm{C}$:

$$T(x, y, 0) \;=\; 20 \circ C.$$

---




# 3. Code Explanation

Below is the code you provided, broken down into logical sections:

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import os

# Configure GPU (if available)
physical_devices = tf.config.list_physical_devices('GPU')
if physical_devices:
    try:
        for device in physical_devices:
            tf.config.experimental.set_memory_growth(device, True)
        print("GPU is available and will be used.")
    except RuntimeError as e:
```

```
        print(e)
else:
    print("No GPU available. Running on CPU.")

# Set random seeds
tf.random.set_seed(42)
np.random.seed(42)

# Problem parameters
rho = 8000.0       # Density (kg/m³)
C_p = 0.466        # Specific heat capacity (J/(kg·K))
k = 45.0           # Thermal conductivity (W/(m·K))
alpha = k / (rho * C_p)  # Thermal diffusivity (m²/s)
T_heated = 100.0   # Temperature at heated corner (°C)
T_initial = 20.0   # Initial temperature (°C)

# Domain boundaries
x_min, x_max = 0.0, 1.0  # Plate dimensions (meters)
y_min, y_max = 0.0, 1.0
t_min, t_max = 0.0, 20.0 # Simulation time (seconds)

# Neural network definition
class HeatConductionPINN(tf.keras.Model):
    def __init__(self, num_hidden_layers=8, num_neurons_per_layer=40, **kwargs):
        super(HeatConductionPINN, self).__init__(**kwargs)
        self.input_layer = tf.keras.layers.InputLayer(input_shape=(3,))
        self.hidden_layers = tf.keras.Sequential([
            tf.keras.layers.Dense(num_neurons_per_layer,
                                  activation='tanh',
                                  kernel_initializer='glorot_normal')
            for _ in range(num_hidden_layers)
        ])
        self.temp_output = tf.keras.layers.Dense(1, activation=None)

    def call(self, inputs, **kwargs):
        x = self.hidden_layers(inputs)
        return self.temp_output(x)

# Generate training data
N_f = 10000  # Collocation points
N_b = 2000   # Points per boundary condition
N_i = 5000   # Initial condition points

# Collocation points (x, y, t)
x_f = tf.random.uniform((N_f, 1), x_min, x_max, dtype=tf.float32)
```

```
y_f = tf.random.uniform((N_f, 1), y_min, y_max, dtype=tf.float32)
t_f = tf.random.uniform((N_f, 1), t_min, t_max, dtype=tf.float32)
X_f = tf.concat([x_f, y_f, t_f], axis=1)

# Dirichlet boundary condition at (0,0) corner
x_b_dir = x_min * tf.ones((N_b, 1), dtype=tf.float32)
y_b_dir = y_min * tf.ones((N_b, 1), dtype=tf.float32)
t_b_dir = tf.random.uniform((N_b, 1), t_min, t_max, dtype=tf.float32)
X_b_dir = tf.concat([x_b_dir, y_b_dir, t_b_dir], axis=1)
T_b_dir = T_heated * tf.ones((N_b, 1), dtype=tf.float32)

# Neumann boundary conditions (insulated boundaries)
# Left edge (x=0, y>0)
x_b_left = x_min * tf.ones((N_b, 1), dtype=tf.float32)
y_b_left = tf.random.uniform((N_b, 1), y_min + 1e-3, y_max, dtype=tf.float32)
t_b_left = tf.random.uniform((N_b, 1), t_min, t_max, dtype=tf.float32)
X_b_left = tf.concat([x_b_left, y_b_left, t_b_left], axis=1)

# Right edge (x=1)
x_b_right = x_max * tf.ones((N_b, 1), dtype=tf.float32)
y_b_right = tf.random.uniform((N_b, 1), y_min, y_max, dtype=tf.float32)
t_b_right = tf.random.uniform((N_b, 1), t_min, t_max, dtype=tf.float32)
X_b_right = tf.concat([x_b_right, y_b_right, t_b_right], axis=1)

# Bottom edge (y=0, x>0)
x_b_bottom = tf.random.uniform((N_b, 1), x_min + 1e-3, x_max, dtype=tf.float32)
y_b_bottom = y_min * tf.ones((N_b, 1), dtype=tf.float32)
t_b_bottom = tf.random.uniform((N_b, 1), t_min, t_max, dtype=tf.float32)
X_b_bottom = tf.concat([x_b_bottom, y_b_bottom, t_b_bottom], axis=1)

# Top edge (y=1)
x_b_top = tf.random.uniform((N_b, 1), x_min, x_max, dtype=tf.float32)
y_b_top = y_max * tf.ones((N_b, 1), dtype=tf.float32)
t_b_top = tf.random.uniform((N_b, 1), t_min, t_max, dtype=tf.float32)
X_b_top = tf.concat([x_b_top, y_b_top, t_b_top], axis=1)

# Initial condition points (t=0)
x_i = tf.random.uniform((N_i, 1), x_min, x_max, dtype=tf.float32)
y_i = tf.random.uniform((N_i, 1), y_min, y_max, dtype=tf.float32)
t_i = t_min * tf.ones((N_i, 1), dtype=tf.float32)
X_i = tf.concat([x_i, y_i, t_i], axis=1)
T_i = T_initial * tf.ones((N_i, 1), dtype=tf.float32)

@tf.function
def heat_loss_fn(model, X_f, X_b_dir, T_b_dir,
```

```
            X_b_left, X_b_right, X_b_bottom, X_b_top,
            X_i, T_i):
    with tf.GradientTape(persistent=True) as tape:
        # PDE residual calculation
        tape.watch(X_f)
        T_pred = model(X_f)
        T_pred = tf.squeeze(T_pred)

        # First derivatives
        grads = tape.gradient(T_pred, X_f)
        dT_dx = grads[:, 0]
        dT_dy = grads[:, 1]
        dT_dt = grads[:, 2]

        # Second derivatives
        d2T_dx2 = tape.gradient(dT_dx, X_f)[:, 0]
        d2T_dy2 = tape.gradient(dT_dy, X_f)[:, 1]

    # PDE residual
    pde_residual = dT_dt - alpha * (d2T_dx2 + d2T_dy2)
    pde_loss = tf.reduce_mean(tf.square(pde_residual))

    # Dirichlet BC (heated corner)
    T_pred_dir = model(X_b_dir)
    dir_loss = tf.reduce_mean(tf.square(T_pred_dir - T_b_dir))

    # Neumann BCs (insulated boundaries)
    def neumann_loss(X, normal_derivative_index):
        with tf.GradientTape() as t:
            t.watch(X)
            T = model(X)
        dT_dn = t.gradient(T, X)[:, normal_derivative_index]
        return tf.reduce_mean(tf.square(dT_dn))

    left_loss = neumann_loss(X_b_left, 0)    # dT/dx = 0 at x=0
    right_loss = neumann_loss(X_b_right, 0)  # dT/dx = 0 at x=1
    bottom_loss = neumann_loss(X_b_bottom, 1) # dT/dy = 0 at y=0 (x>0)
    top_loss = neumann_loss(X_b_top, 1)       # dT/dy = 0 at y=1

    # Initial condition
    T_pred_i = model(X_i)
    ic_loss = tf.reduce_mean(tf.square(T_pred_i - T_i))

    # Total loss
    return (pde_loss + dir_loss + ic_loss +
```

```
        left_loss + right_loss + bottom_loss + top_loss)

# Initialize model and optimizer
model = HeatConductionPINN()
optimizer = tf.keras.optimizers.Adam(learning_rate=1e-3)

@tf.function
def train_step():
    with tf.GradientTape() as tape:
        loss = heat_loss_fn(model, X_f, X_b_dir, T_b_dir,
                            X_b_left, X_b_right, X_b_bottom, X_b_top,
                            X_i, T_i)
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
    return loss

def train_model(epochs=30000):
    for epoch in range(epochs):
        loss = train_step()
        if epoch % 100 == 0:
            print(f"Epoch {epoch}, Loss: {loss.numpy():.5f}")

def visualize_heat_distribution(model):
    # Create spatial grid
    x = np.linspace(x_min, x_max, 100)
    y = np.linspace(y_min, y_max, 100)
    X, Y = np.meshgrid(x, y)

    # Time points for visualization
    time_points = [0.0, 1.0, 5.0, t_max]

    plt.figure(figsize=(20, 16))
    for i, t in enumerate(time_points):
        XY = np.hstack([X.flatten()[:, None],
                        Y.flatten()[:, None],
                        t * np.ones_like(X).flatten()[:, None]])

        T_pred = model(tf.constant(XY, dtype=tf.float32))
        T_plot = T_pred.numpy().reshape(X.shape)

        plt.subplot(2, 2, i+1)
        plt.contourf(X, Y, T_plot, levels=50, cmap='jet')
        plt.colorbar(label='Temperature (°C)')
        plt.title(f'Temperature Distribution at t = {t} seconds')
        plt.xlabel('x (m)')
```

```
        plt.ylabel('y (m)')
        # Mark the heated corner
        plt.scatter([0], [0], c='white', s=100, marker='*')

    plt.tight_layout()
    plt.show()

# Save training data
np.savez_compressed('training_data_corner.npz',
                    X_f=X_f.numpy(), X_b_dir=X_b_dir.numpy(), T_b_dir=T_b_dir.numpy(),
                    X_b_left=X_b_left.numpy(), X_b_right=X_b_right.numpy(),
                    X_b_bottom=X_b_bottom.numpy(), X_b_top=X_b_top.numpy(),
                    X_i=X_i.numpy(), T_i=T_i.numpy())

# Train and visualize
train_model()
visualize_heat_distribution(model)
```

### 3.1 GPU Configuration and Seeds

- **GPU Check:** Allocates memory only as needed.
- **Seeds:** Ensures reproducible random number generation.

### 3.2 Physical Parameters

- $\rho$, $Cp$, $k$: Physical properties for computing $\alpha$.
- $T_{\text{heated}} = 100°\text{C}$: The corner temperature.
- $T_{\text{initial}} = 20°\text{C}$: Initial uniform temperature.

### 3.3 Neural Network Definition

- **Architecture:** A custom **TensorFlow Keras** model (`HeatConductionPINN`) with 8 hidden layers of 40 neurons each, using `tanh` activations.
- **Input:** $[x, y, t]$
- **Output:** Scalar temperature $T$.

### 3.4 Data Generation

- **Collocation Points ($N_f$)**: Points in the interior of $(x, y, t)$ space where the PDE is enforced.
- **Dirichlet Corner ($N_b$)**: Points at $(0, 0)$ with random $t$, assigned $T_{\text{heated}} = 100°\text{C}$.
- **Neumann (Insulated) Boundaries**: Points along edges $x = 0, y > 0; x = 1; y = 0, x > 0; y = 1$ . The code enforces $\frac{\partial T}{\partial n} = 0$.

- **Initial Condition (NiN_i)**: Points at $t = 0$ across the domain with $T_{\text{initial}} = 20°\text{C}$.

### 3.5 Loss Function

- **PDE Residual:** Minimizes →
  $\left[\frac{\partial T}{\partial t} - \alpha(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2})\right]^2$
- **Dirichlet Loss (Corner):** Enforces $T = 100°\text{C}$ at $(0, 0)$.
- **Neumann Losses:** Zero flux at other boundaries.
- **Initial Condition Loss:** Enforces $T = 20 \circ C$ at $t = 0$.

### 3.6 Training

- **Optimizer:** Adam with a learning rate of $1 \times 10^{-3}$.
- **Training Loop:** Runs up to 30,000 epochs, printing the loss every 100 epochs.

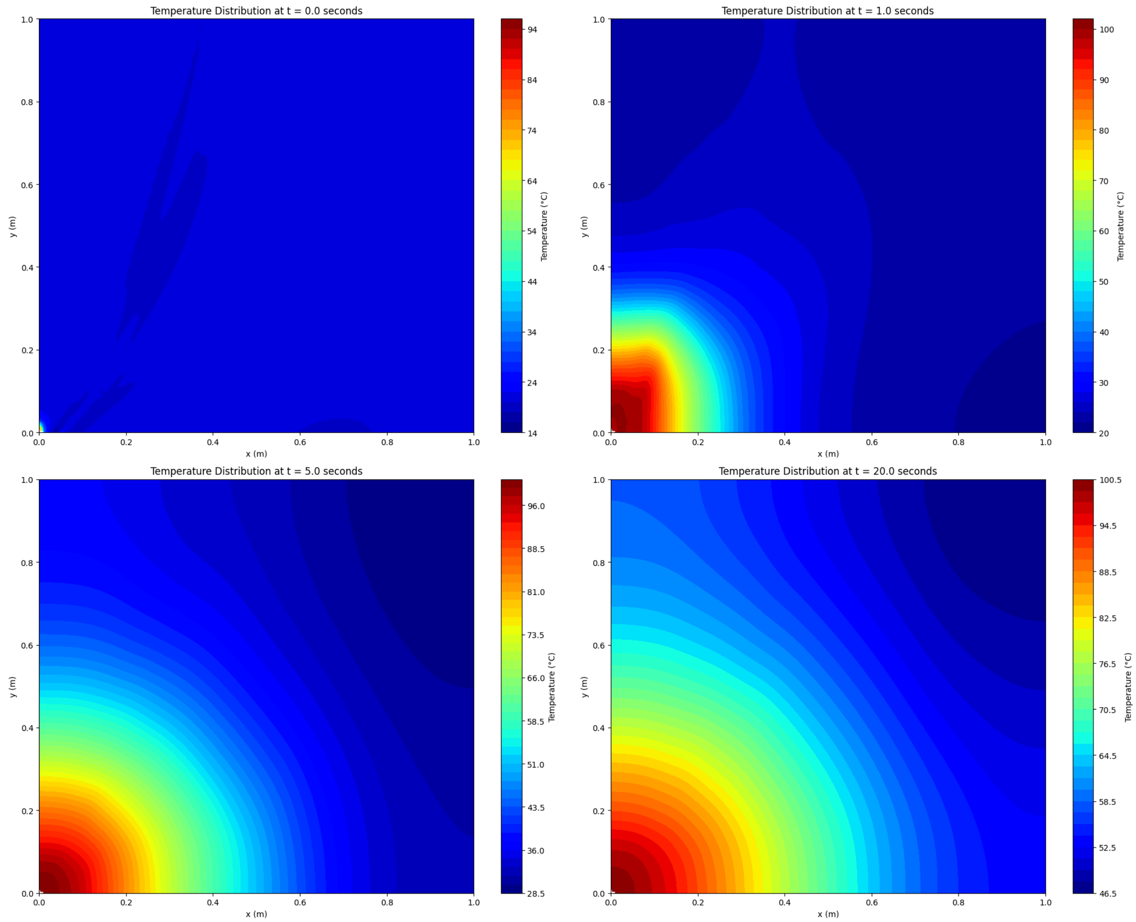
### 3.7 Visualization

- **Contour Plots:** Shows how $T$ evolves at four time snapshots:
  $t = \{0.0,\ 1.0,\ 5.0,\ 20.0\}$
- **Marked Corner:** A white star `*` indicates the heated corner at $(0, 0)$.

---

# 4. Output Analysis

Below is the provided image with four subplots of the temperature distribution at times t=0.0, 1.0, 5.0, 20.0t=0.0,\ 1.0,\ 5.0,\ 20.0 seconds:




1. $t = 0.0$ **seconds (Top-Left):**
    - The plate is uniformly at $20^{\circ}\mathrm{C}$, hence a dark-blue region.
    - The heated corner at (0,0) is a single point, so it does not visually change the colour distribution at this exact moment.
2. $t = 1.0$ **seconds (Top-Right):**
    - Heat begins to diffuse from the corner, creating a localized "hot spot."
    - You see a colour gradient near (0,0) transitioning from $\approx 100^{\circ}\mathrm{C}$ (red/orange) to $\approx 20^{\circ}\mathrm{C}$ (blue) farther away.
3. $t = 5.0$ **seconds (Bottom-Left):**
    - The heated region expands further into the plate, with a larger area showing intermediate temperatures (green/yellow).
    - The gradient extends radially from the corner, consistent with heat conduction in two dimensions.
4. $t = 20.0$ **seconds (Bottom-Right):**
    - A substantial portion of the plate is now warmer.

- The highest temperature remains at the corner, while the outermost areas are still cooler but significantly higher than the initial

  $20°\mathrm{C}.$

- Given enough time (and fully insulated boundaries), the temperature would continue to spread until the entire plate eventually approached equilibrium near $100°\mathrm{C}$.

**Key Observations:**

- **Localized Heating from a Corner:**

  Unlike heating an entire edge, the heat source is now concentrated at a single point. This leads to more radially symmetric diffusion patterns emanating from the corner.

- **Expanding Thermal Front:**

  Over time, the heated zone grows, and the temperature contours form approximate concentric arcs around (0,0).

- **Approach to Steady State:**

  At $t = 20$ seconds, the plate has not reached a uniform temperature, but the influence of the corner heating is evident over a large portion of the domain.

---

# 5. Summary

In this setup, the 2D heat conduction problem is solved using a PINN approach:

1. **Physics-Informed Neural Network:**

   The neural network is trained to satisfy the heat equation, boundary conditions, and initial condition simultaneously.

2. **Corner Heating vs. Edge Heating:**

   Instead of imposing a high temperature along an entire boundary, only a single corner point is heated, resulting in a more localized thermal gradient.

3. **Resulting Temperature Fields:**

   The plots demonstrate how heat diffuses radially from the corner, gradually warming the plate. The solution shows a clear evolution over time, which aligns with physical expectations of heat transfer from a single-point source.

By inspecting the contours at different time snapshots, we confirm that the model is capturing the expected physics: localized heating at (0,0)(0,0) diffuses into the interior, creating a growing hot region bounded by cooler areas.

---