

# Sesión 4: Cálculo Diferencial Multivariable

En problemas de Aprendizaje de Máquina, a menudo trabajamos con funciones de varias variables (por ejemplo, funciones de costo que dependen de múltiples parámetros). El cálculo diferencial multivariable permite analizar cómo cambia la función en relación con cada variable, identificando direcciones de mayor incremento o disminución y localizando máximos o mínimos en espacios de alta dimensión.

 **por Kibernet Capacitación S.A.**

# Introducción al Cálculo Diferencial Multivariable

En el campo del Aprendizaje de Máquina, muchos de los problemas de optimización involucran funciones de varias variables, cuyos parámetros se ajustan para minimizar o maximizar alguna métrica de desempeño (por ejemplo, la función de pérdida o error). Para comprender y resolver estos problemas de manera eficiente, es fundamental contar con bases sólidas en el cálculo diferencial multivariable.

El cálculo multivariable es fundamental en el campo del Machine Learning por varias razones, ya que muchos de los problemas que se abordan involucran funciones de varias variables, es decir, dependen de múltiples parámetros. A continuación, se describen algunos de sus beneficios y dificultades:

# Beneficios del Cálculo Multivariable en Machine Learning



## Optimización de Funciones de Costo

El cálculo multivariable permite calcular el gradiente, que es el vector de derivadas parciales de la función de costo respecto a cada parámetro. Este gradiente indica la dirección de mayor aumento de la función, lo cual es crucial para algoritmos de optimización como el descenso de gradiente, que ajustan iterativamente los parámetros para minimizar el error del modelo.

La matriz Hessiana, que contiene las segundas derivadas parciales, proporciona información sobre la curvatura de la función. Esto ayuda a diseñar métodos de optimización más sofisticados (como el método de Newton) que pueden converger más rápidamente que los métodos de primer orden.



## Reducción de Dimensionalidad y Análisis de Componentes Principales (PCA)

Técnicas como el PCA se basan en conceptos del cálculo multivariable, ya que involucran la descomposición de la matriz de covarianza de los datos en valores y vectores propios. Esto ayuda a identificar las direcciones de mayor varianza en los datos y a reducir la dimensionalidad sin perder información esencial.



## Modelado y Análisis de Datos en Espacios de Alta Dimensión

Los modelos de Machine Learning, como las redes neuronales, a menudo involucran un gran número de parámetros. El cálculo multivariable es esencial para entender cómo cada uno de estos parámetros afecta el comportamiento del modelo, permitiendo un análisis más profundo de la influencia de cada variable.



## Interpretación Geométrica de Modelos

El cálculo multivariable permite visualizar y entender de manera intuitiva la forma de las funciones de costo, los gradientes y la curvatura de la superficie en el espacio de parámetros. Esta interpretación geométrica facilita la detección de problemas como puntos de silla o mínimos locales, y ayuda a diseñar estrategias para evitarlos.

# Dificultades del Cálculo Multivariable en Machine Learning



## Complejidad Computacional

El cálculo de gradientes y Hessianos en modelos con muchos parámetros puede ser computacionalmente costoso. Esto es especialmente cierto en redes neuronales profundas, donde la cantidad de parámetros puede ser extremadamente alta.

La evaluación y el almacenamiento de matrices de gran tamaño (como la Hessiana) pueden requerir una cantidad considerable de memoria y potencia de procesamiento.



## Problemas de Convergencia

En espacios de alta dimensión, las funciones de costo pueden tener múltiples mínimos locales, puntos de silla y regiones planas, lo que dificulta la convergencia de los algoritmos de optimización.

Ajustar adecuadamente la tasa de aprendizaje y otros hiperparámetros es crucial para garantizar que el algoritmo converja de manera eficiente.



## Estabilidad Numérica

Los cálculos de derivadas en modelos complejos pueden sufrir problemas de estabilidad numérica, especialmente cuando se usan métodos numéricos aproximados. Esto puede afectar la precisión de los gradientes y, en consecuencia, el rendimiento del modelo.



## Interpretación y Visualización

Aunque el cálculo multivariable proporciona una rica interpretación geométrica, visualizar funciones y gradientes en espacios de más de tres dimensiones es inherentemente difícil. Esto limita la capacidad de analizar visualmente el comportamiento de los modelos en entornos de alta dimensionalidad.

# Derivadas Parciales

## Definición y Significado Geométrico

Para una función  $f(x,y)$  la derivada parcial con respecto a  $x$  se define manteniendo  $y$  constante. Se denota como  $\partial f / \partial x$ .

Geométricamente, describe la pendiente de la superficie  $f(x,y)$  en la dirección del eje  $x$ .

## Notación y Ejemplos

Si  $f(x,y) = x^2 + xy + y^2$ , entonces:

$$\frac{\partial f}{\partial x} = 2x + y, \quad \frac{\partial f}{\partial y} = x + 2y.$$

Para funciones con más variables, se sigue el mismo principio, considerando constantes todas las demás.

# Conceptos y Ejemplo de Derivadas Parciales

Consideramos la función:

$$f(x,y) = x^2 + xy + y^2$$

Esta función puede representar una función de costo simple que depende de dos parámetros (por ejemplo,  $x$  e  $y$ ) en un modelo de regresión.

Las derivadas parciales miden cómo cambia  $f(x,y)$  cuando solo varía una de las variables (manteniendo la otra constante).

Caso de uso en Machine Learning: En la optimización de una función de costo, calcular las derivadas parciales permite conocer cómo cada parámetro (por ejemplo, coeficientes en un modelo de regresión) afecta el error del modelo.

```
import sympy as sp

# Definir las variables simbólicas
x, y = sp.symbols('x y')

# Definir la función f(x, y)
f_expr = x**2 + x*y + y**2

# Calcular derivadas parciales
f_x = sp.diff(f_expr, x) #  $\partial f / \partial x$ 
f_y = sp.diff(f_expr, y) #  $\partial f / \partial y$ 

print("Función f(x, y) =", f_expr)
print("Derivada parcial con respecto a x:  $\partial f / \partial x$  =", f_x)
print("Derivada parcial con respecto a y:  $\partial f / \partial y$  =", f_y)
```

→ Función  $f(x, y) = x^2 + xy + y^2$   
Derivada parcial con respecto a x:  $\partial f / \partial x = 2x + y$   
Derivada parcial con respecto a y:  $\partial f / \partial y = x + 2y$

### Caso de uso en Machine Learning:

En la optimización de una función de costo, calcular las derivadas parciales permite conocer cómo cada parámetro (por ejemplo, coeficientes en un modelo de regresión) afecta el error del modelo.

# Gradiente

## Definición como Vector de Derivadas Parciales

El gradiente de una función  $f(x)$  en  $\mathbb{R}^n$  es el vector formado por todas sus derivadas parciales. Para  $f(x_1, x_2, \dots, x_n)$ .

$$\nabla f = \left( \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right).$$

Cálculo del Gradiente: Si  $f(x,y) = 3x^2 + xy - y^2$  el gradiente es:

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) = (6x + y, x - 2y).$$


## Interpretación Geométrica

El gradiente apunta en la dirección de máximo incremento de la función, y su magnitud indica la tasa de cambio en esa dirección.



# Conceptos y Ejemplo del Gradiente

En Machine Learning, durante la optimización se utiliza en métodos como el descenso de gradiente para moverse en la dirección opuesta y minimizar la función de costo.

```
 # Calcular el gradiente (ya definido como el vector de derivadas parciales)
gradient = sp.Matrix([f_x, f_y])

print("Gradiente de f(x, y):", gradient)
```

```
 Gradiente de f(x, y): Matrix([[2*x + y], [x + 2*y]])
```

Caso de uso en Machine Learning: En un algoritmo de descenso de gradiente, se actualizan los parámetros de un modelo restando una fracción del gradiente, lo que permite minimizar la función de costo y mejorar el ajuste del modelo.

# Matriz Hessiana

## Definición

La matriz Hessiana de una función  $f(x)$  es la matriz cuadrada de sus segundas derivadas parciales. Para  $f(x, y)$ :

$$H = \begin{pmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{pmatrix}.$$

La matriz Hessiana de  $f(x,y)$  es la matriz de segundas derivadas parciales:

```
# Calcular la matriz Hessiana de f(x,y)
hessian = sp.hessian(f_expr, (x, y))

print("Matriz Hessiana de f(x, y):\n", hessian)
```

```
⇒ Matriz Hessiana de f(x, y):
Matrix([[2, 1], [1, 2]])
```

## Caso de uso en Machine Learning

Al optimizar una función de costo, la Hessiana se puede usar para verificar si un punto crítico es un mínimo (cuando la Hessiana es definida positiva). Esto es útil en métodos de optimización de segundo orden, que pueden acelerar la convergencia del algoritmo.

## Relación con la Curvatura de la Función

El Hessiano describe la curvatura de la función y permite determinar la naturaleza de los puntos críticos (mínimos locales, máximos locales o puntos de silla).



```
con((0) =  
    naicoritt)  
ttatccitt)  
    a((0)) = (nionttinoittoitt)  
npyi((0)) it)  
  
oa(ttto = ((ctort)(0))  
    naicoritt)  
con((0) =  
    a((0)) na = uittitt)  
npyi((0)) )
```

# Uso de Python para Favorecer el Aprendizaje

## Cálculos Numéricos y Simbólicos

- NumPy: Ideal para operaciones vectorizadas y cálculo numérico.
- SymPy: Permite derivaciones simbólicas de manera sencilla.

## Visualización Gráfica de Superficies

- Matplotlib o Plotly: Para graficar funciones en 2D/3D y representar gradientes.

## Resolución de Problemas de Optimización

- SciPy: Provee métodos para encontrar mínimos y máximos de funciones multivariantes (por ejemplo, `scipy.optimize.minimize`).

# Ejemplo integrando derivadas parciales, gradiente y matriz Hessiana

```
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# Definir variables simbólicas
x, y = sp.symbols('x y', real=True)

# Definir una función de dos variables
f_expr = x**2 + x*y + y**2

# Derivadas parciales
f_dx = sp.diff(f_expr, x)
f_dy = sp.diff(f_expr, y)

# Gradiente
gradient_expr = (f_dx, f_dy)

# Matriz Hessiana
hessian_expr = sp.hessian(f_expr, (x, y))
print("f(x, y) =", f_expr)
print("∂f/∂x =", f_dx)
print("∂f/∂y =", f_dy)
print("Gradiente =", gradient_expr)
print("Hessiana =\n", hessian_expr)
```

```
⇒ f(x, y) = x**2 + x*y + y**2
   ∂f/∂x = 2*x + y
   ∂f/∂y = x + 2*y
   Gradiente = (2*x + y, x + 2*y)
   Hessiana =
   Matrix([[2, 1], [1, 2]])
```

# Ejemplo 2: Ajuste de un modelo de regresión

Contexto: ajustar un modelo de regresión sencillo mediante la minimización de una función de costo. En este ejemplo, consideramos un modelo lineal con dos parámetros (pendiente  $w$  y sesgo  $b$ ) y dos datos de entrenamiento, y definimos la función de costo como el error cuadrático medio (MSE) para estos puntos.

Supongamos que tenemos los siguientes datos de entrenamiento:

- $(x_1, y_1) = (1, 2)$
- $(x_2, y_2) = (2, 3)$

La función de costo se define como:

$$J(w, b) = \frac{1}{2} \left[ (w \cdot 1 + b - 2)^2 + (w \cdot 2 + b - 3)^2 \right]$$

## Paso 1: Definir la Función de Costo y Calcular las Derivadas Parciales

Utilizamos Sympy para definir simbólicamente la función de costo y calcular las derivadas parciales respecto a  $w$  y  $b$ .

```
import sympy as sp

# Definir las variables simbólicas
w, b = sp.symbols('w b', real=True)

# Definir la función de costo J(w, b)
J = 0.5 * ( (w*1 + b - 2)**2 + (w*2 + b - 3)**2 )

# Calcular las derivadas parciales
J_w = sp.diff(J, w) # ∂J/∂w
J_b = sp.diff(J, b) # ∂J/∂b

print("Función de costo J(w, b):")
sp.pprint(sp.simplify(J))

print("\nDerivada parcial respecto a w (∂J/∂w):")
sp.pprint(sp.simplify(J_w))

print("\nDerivada parcial respecto a b (∂J/∂b):")
sp.pprint(sp.simplify(J_b))
```

```
⇒ Función de costo J(w, b):
      2      2
0.5·(b + w - 2) + 0.5·(b + 2·w - 3)

Derivada parcial respecto a w (∂J/∂w):
3.0·b + 5.0·w - 8.0

Derivada parcial respecto a b (∂J/∂b):
2.0·b + 3.0·w - 5.0
```

### Interpretación:

Las derivadas parciales indican cómo varía el error con respecto a cada uno de los parámetros. Esto es esencial para saber en qué dirección ajustar  $w$  y  $b$  para minimizar el error del modelo.



# Continuación del Ejemplo 2

## Paso 2: Calcular el Gradiente y la Matriz Hessiana

El gradiente es el vector formado por las derivadas parciales y señala la dirección de mayor incremento de la función de costo. La matriz Hessiana contiene las segundas derivadas y proporciona información sobre la curvatura de la función.

```
# Calcular el gradiente como vector de derivadas parciales
gradient = sp.Matrix([J_w, J_b])

print("\nGradiente de J(w, b):")
sp.pprint(sp.simplify(gradient))

# Calcular la matriz Hessiana (segundas derivadas parciales)
Hessian = sp.hessian(J, (w, b))
print("\nMatriz Hessiana de J(w, b):")
sp.pprint(sp.simplify(Hessian))
```



```
Gradiente de J(w, b):
[3.0·b + 5.0·w - 8.0]
[2.0·b + 3.0·w - 5.0]
```

```
Matriz Hessiana de J(w, b):
[5.0  3.0]
[3.0  2.0]
```

Interpretación:

- El gradiente nos indica la dirección en la que la función de costo aumenta más rápidamente. En métodos de optimización, como el descenso de gradiente, se utiliza su valor para ajustar los parámetros en la dirección opuesta.
- La Hessiana nos ayuda a entender la curvatura de la función de costo, permitiendo diferenciar entre mínimos locales, máximos o puntos de silla.

# Finalización del Ejemplo 2

## Paso 3: Encontrar los Parámetros Óptimos

Al establecer el gradiente a cero, podemos resolver el sistema de ecuaciones para encontrar el punto donde la función de costo es mínima.

```
# Resolver el sistema de ecuaciones: gradiente = 0
solution = sp.solve(gradient, (w, b))

print("\nParámetros óptimos (mínimo de la función de costo):")
sp.pprint(solution)
```



```
Parámetros óptimos (mínimo de la función de costo):
{b: 1.0, w: 1.0}
```

Interpretación: La solución obtenida corresponde a los valores de  $w$  y  $b$  que minimizan la función de costo, es decir, ajustan el modelo de regresión de manera óptima a los datos de entrenamiento.

Este ejemplo muestra cómo se pueden aplicar:

- Derivadas parciales para entender cómo cada parámetro afecta el error,
- El gradiente para indicar la dirección de ajuste de los parámetros, y
- La matriz Hessiana para analizar la curvatura y asegurar que el punto encontrado sea un mínimo.

En un contexto de Machine Learning, estos cálculos son la base para optimizar modelos de regresión y otros algoritmos. Por ejemplo, en algoritmos de descenso de gradiente, el gradiente se utiliza para actualizar los parámetros de forma iterativa, y la información de la Hessiana puede ayudar en métodos de segundo orden para acelerar la convergencia y mejorar la estabilidad del entrenamiento.

Este proceso no solo ilustra los fundamentos teóricos del cálculo multivariable, sino que también demuestra su aplicación práctica en la optimización de modelos en Machine Learning.

# Actividad Práctica Guiada: Identificando y clasificando punto crítico

Objetivo: Aplicar el cálculo diferencial multivariable para identificar y clasificar un punto crítico de una función de dos variables usando Python (SymPy y Matplotlib).

## Paso a Paso:

1. Definir la Función y Calcular sus Derivadas
  - Crear una función  $g(x,y)$  que presente un punto crítico claro (por ejemplo,  $g(x,y) = x^2 + 2xy + 2y^2$ ).
  - Usar `sp.diff` para obtener las derivadas parciales y la matriz Hessiana.
2. Encontrar y Clasificar el Punto Crítico
  - Resolver las ecuaciones  $\partial g / \partial x = 0$  y  $\partial g / \partial y = 0$  para encontrar los puntos críticos.
  - Evaluar la Hessiana en dichos puntos para determinar si son mínimos, máximos o puntos de silla.
3. Visualizar la Función
  - Generar un `meshgrid` en NumPy y evaluar  $g(x,y)$ .
  - Graficar la superficie 3D con Matplotlib o Plotly.
  - Destacar el punto crítico en la gráfica.



```

# -----
# Paso 2: Encontrar y clasificar el punto crítico
# -----
# Resolver el sistema de ecuaciones: dg_dx = 0 y dg_dy = 0
critical_points = sp.solve([dg_dx, dg_dy], (x, y))

print("\nPunto(s) crítico(s):")
sp.pprint(critical_points)

# Supongamos que el punto crítico encontrado es (x, y) = (0, 0)
# Evaluar la Hessiana en el punto crítico para clasificarlo
hessian_at_critical = hessian.subs({x: 0, y: 0})

print("\nMatriz Hessiana evaluada en el punto crítico (0,0):")
sp.pprint(hessian_at_critical)

# Determinar la naturaleza del punto crítico
# Convertir la Hessiana a una matriz numérica
hessian_numeric = np.array(hessian_at_critical).astype(np.float64)
eigenvalues = np.linalg.eigvals(hessian_numeric)

print("\nValores propios de la Hessiana en (0,0):", eigenvalues)

# Interpretación: Si todos los valores propios son positivos, es un mínimo;
# si negativos, un máximo; si mixtos, un punto de silla.
if all(e > 0 for e in eigenvalues):
    classification = "Mínimo local"
elif all(e < 0 for e in eigenvalues):
    classification = "Máximo local"
else:
    classification = "Punto de silla"

print("\nClasificación del punto crítico (0,0):", classification)

```

```

# -----
# Paso 3: Visualizar la función y el punto crítico
# -----
# Convertir la función simbólica a función numérica usando lambdify
g_numeric = sp.lambdify((x, y), g_expr, 'numpy')

# Crear un meshgrid para los valores de x e y
x_vals = np.linspace(-5, 5, 100)
y_vals = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x_vals, y_vals)
Z = g_numeric(X, Y)

# Graficar la superficie 3D de g(x,y)
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
surf = ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.8)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('g(x, y)')
ax.set_title('Superficie de g(x,y) = x^2 + 2xy + 2y^2')

# Destacar el punto crítico (0,0) sobre la gráfica
g_critical = g_numeric(0, 0)
ax.scatter(0, 0, g_critical, color='red', s=100, label=f'Punto Crítico (0,0): {classification}')
ax.legend()

# Mostrar la gráfica
plt.show()

```



Función  $g(x,y)$ :

$$x^2 + 2xy + 2y^2$$

Derivada parcial respecto a  $x$  ( $\partial g / \partial x$ ):  
 $2x + 2y$

Derivada parcial respecto a  $y$  ( $\partial g / \partial y$ ):  
 $2x + 4y$

Matriz Hessiana de  $g(x,y)$ :

$$\begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

Punto(s) crítico(s):  
 $\{x: 0, y: 0\}$

Matriz Hessiana evaluada en el punto crítico  $(0,0)$ :

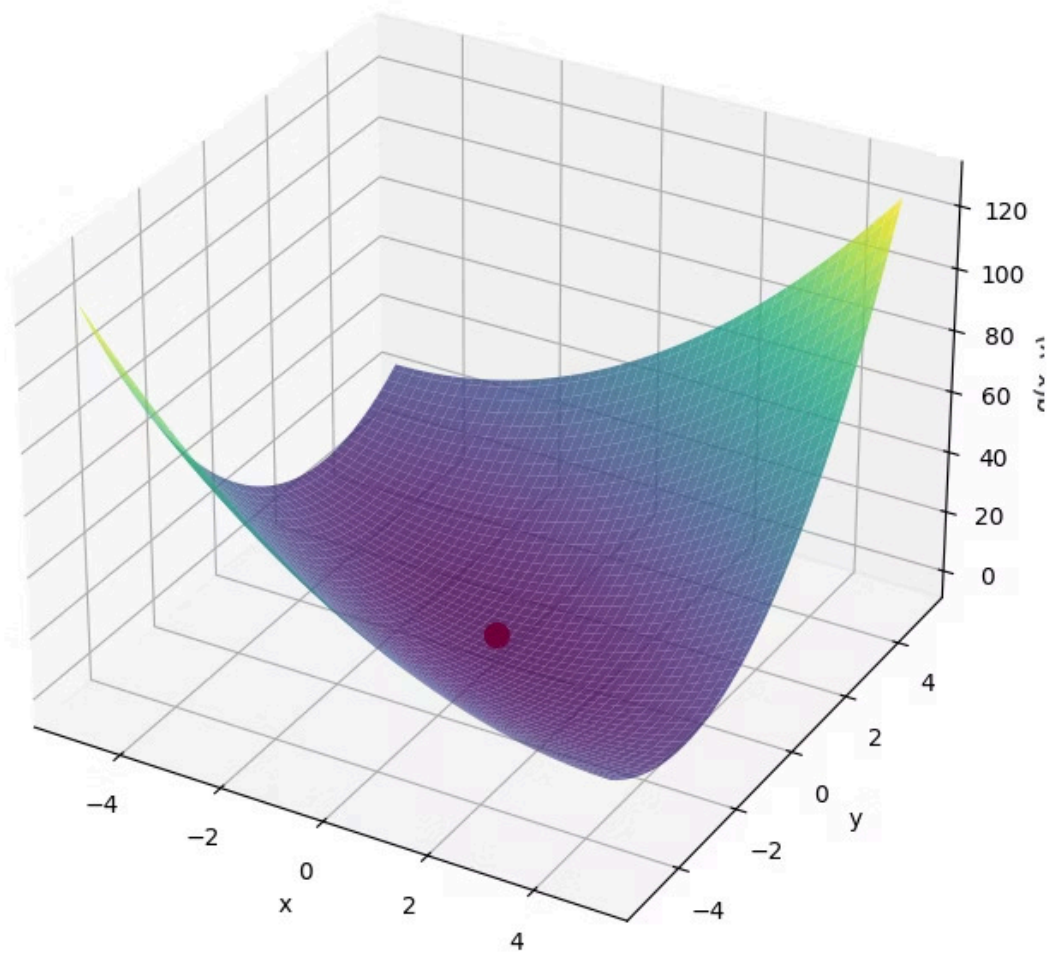
$$\begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

Valores propios de la Hessiana en  $(0,0)$ :  $[0.76393202 \ 5.23606798]$

Clasificación del punto crítico  $(0,0)$ : Mínimo local

Superficie de  $g(x,y) = x^2 + 2xy + 2y^2$

● Punto Crítico (0,0): Mínimo local



# Explicación del Código:

1. Definir la Función y Derivadas:
  - Se definen las variables simbólicas  $x$  y  $y$ .
  - Se establece la función  $g(x,y) = x^2 + 2xy + 2y^2$
  - Se calculan las derivadas parciales utilizando `sp.diff`.
  - Se construye la matriz Hessiana usando `sp.hessian`.
2. Encontrar y Clasificar el Punto Crítico:
  - Se resuelve el sistema  $\partial g/\partial x=0$  y  $\partial g/\partial y=0$  con `sp.solve` para hallar los puntos críticos.
  - Se evalúa la Hessiana en el punto crítico (en este caso,  $(0,0)$ ) y se calculan sus valores propios.
  - La clasificación del punto crítico se determina según el signo de los valores propios (todos positivos indican un mínimo local).
3. Visualizar la Función:
  - Se convierte la función simbólica a una función numérica usando `sp.lambdify`.
  - Se genera un `meshgrid` con `numpy` para cubrir un rango de valores.
  - Se grafica la superficie 3D de  $g(x,y)$  usando `matplotlib` y se destaca el punto crítico en rojo, indicando su clasificación.