402 MÉTODO DE DESCENSO DE GRADIENTE - TAREA 1

Octubre 06, 2025

Pregunta 1

La función de Himmelblau es un estándar para probar algoritmos de optimización, ya que posee cuatro mínimos locales idénticos. El objetivo es implementar y comparar el rendimiento del método de descenso de gradiente (GD) estándar con el descenso de gradiente con momento (GDM). Se debe analizar cómo la elección de los hiperparámetros (tasa de aprendizaje y momento) afecta la convergencia.

La función está definida como:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

Actividades:

1. Implementación: Defina la función de Himmelblau y su gradiente. Implemente el algoritmo de GD y GDM.
2. Experimentación:
   - Ejecute ambos algoritmos desde el punto inicial (−2.5, 2.5).
   - Para GD, pruebe con tasas de aprendizaje α = [0.001, 0.005, 0.01].
   - Para GDM, utilice una tasa de aprendizaje fija η = 0.005 y pruebe con coeficientes de momentos β = [0.5, 0.7, 0.9].
   - Utilice un máximo de 10000 iteraciones y una tolerancia de 1e-6 como criterios de detención.

Código Fuente de la Implementación

```
from numpy import array as в, linspace as л, meshgrid as м
from matplotlib import pyplot as п

# Vectores
e = [в([-2.5, 2.5]), в([-2.5, 2.5]), в([-2.5, 2.5])]
ε = [в([-2.5, 2.5]), в([-2.5, 2.5]), в([-2.5, 2.5])]

# Tasas de aprendizaje
α = [0.001, 0.005, 0.01]
η = 0.005

# Factor de momento
β = [0.5, 0.7, 0.9]

# Velocidad
ω = [в([0, 0]), в([0, 0]), в([0, 0])]

#  f(x, y) = (x² + y − 11)² + (x + y² − 7)²
f = lambda x: (x[0] ** 2 + x[1] − 11) ** 2 + (x[0] + x[1] ** 2 − 7) ** 2
# ∇f(x, y) = [ 4x(x² + y − 11) + 2(x + y² − 7), 2(x² + y − 11) + 4y(x + y² − 7) ]
φ = lambda x: в(
    [
        4 * x[0] * (x[0] ** 2 + x[1] − 11) + 2 * (x[0] + x[1] ** 2 − 7),
        2 * (x[0] ** 2 + x[1] − 11) + 4 * x[1] * (x[0] + x[1] ** 2 − 7),
    ]
)
# v(x, y, β, ω) = βω + (1 − β)∇f(x, y)
v = lambda w, x, b: b * w + (1 − b) * φ(x)
# g(x, y, α) = (x, y) + α∇f(x, y)
g = lambda x, a: x − a * φ(x)
# γ(x, y, ω) = (x, y) − ηω
γ = lambda x, w: x − η * w

# Historial de vectores
y = [[], [], []]
υ = [[], [], []]

# Costo
c = [[], [], []]
κ = [[], [], []]

# Iteraciones
i = [0, 0, 0]
ι = [0, 0, 0]

# Inicialización
y[0].append(e[0].copy())
y[1].append(e[1].copy())
y[2].append(e[2].copy())
υ[0].append(ε[0].copy())
υ[1].append(ε[1].copy())
υ[2].append(ε[2].copy())
c[0].append(f(e[0]))
c[1].append(f(e[1]))
c[2].append(f(e[2]))
κ[0].append(f(ε[0]))
κ[1].append(f(ε[1]))
κ[2].append(f(ε[2]))

for _ in range(10000):
    e[0] = g(e[0], α[0])
    y[0].append(e[0].copy())
    c[0].append(f(e[0]))
    i[0] += 1
    if f(e[0]) < 1e−6:
        break

for _ in range(10000):
    e[1] = g(e[1], α[1])
    y[1].append(e[1].copy())
    c[1].append(f(e[1]))
    i[1] += 1
```

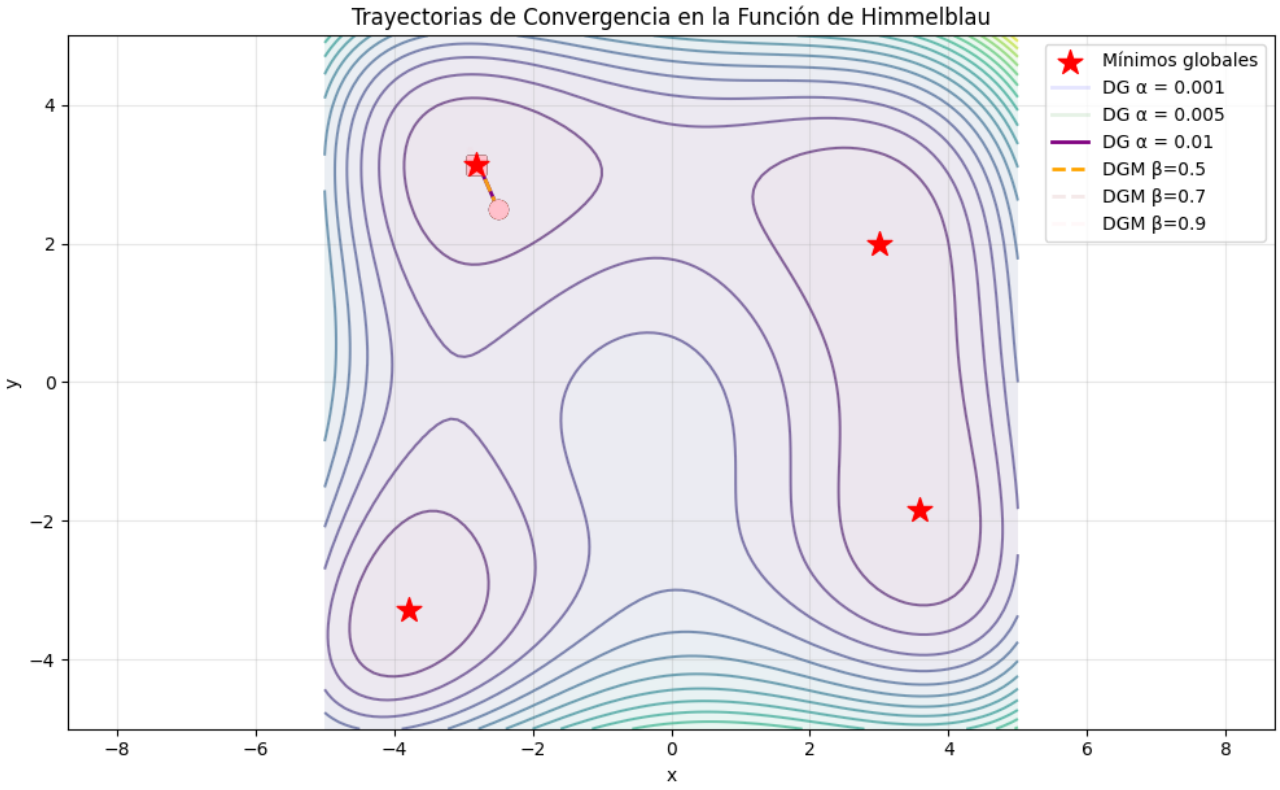
```
    if f(e[1]) < 1e-6:
        break

for _ in range(10000):
    e[2] = g(e[2], α[2])
    y[2].append(e[2].copy())
    c[2].append(f(e[2]))
    i[2] += 1
    if f(e[2]) < 1e-6:
        break

for _ in range(10000):
    ω[0] = v(ω[0], ε[0], β[0])
    ε[0] = ɣ(ε[0], ω[0])
    υ[0].append(ε[0].copy())
    κ[0].append(f(ε[0]))
    ι[0] += 1
    if f(ε[0]) < 1e-6:
        break

for _ in range(10000):
    ω[1] = v(ω[1], ε[1], β[1])
    ε[1] = ɣ(ε[1], ω[1])
    υ[1].append(ε[1].copy())
    κ[1].append(f(ε[1]))
    ι[1] += 1
    if f(ε[1]) < 1e-6:
        break

for _ in range(10000):
    ω[2] = v(ω[2], ε[2], β[2])
    ε[2] = ɣ(ε[2], ω[2])
    υ[2].append(ε[2].copy())
    κ[2].append(f(ε[2]))
    ι[2] += 1
    if f(ε[2]) < 1e-6:
        break

# Gráficos
∂ = л(-5, 5, 100)
б = л(5, -5, 100)
M, N = м(∂, б)
п.figure(figsize=(12, 10))
п.contour(M, N, f(в([M, N])), cmap="viridis", alpha=0.6, levels=20)
п.contourf(M, N, f(в([M, N])), cmap="viridis", alpha=0.1, levels=20)
μ = в(
    [[3.0, 2.0], [-2.805118, 3.131312], [-3.779310, -3.283186], [3.584428, -1.848126]]
)
п.scatter(
    μ[:, 0],
    μ[:, 1],
    color="red",
    marker="*",
    s=200,
    label="Mínimos globales",
    zorder=5,
)

ц = ["blue", "green", "purple", "orange", "brown", "pink"]
э = [f"DG α = {ϑ}" for ϑ in α] + [f"DGM β={ʊ}" for ʊ in β]

for ʊ in range(0, 6, 1):
    if ʊ < 3:
        Y = в(y[ʊ])
        п.plot(
            Y[:, 0],
            Y[:, 1],
            color=ц[ʊ],
            linewidth=2,
            label=э[ʊ],
            alpha=0.1 if ʊ != 2 else 1,
        )
        п.scatter(Y[0, 0], Y[0, 1], color=ц[ʊ], marker="o", s=100, zorder=4)
        п.scatter(Y[-1, 0], Y[-1, 1], color=ц[ʊ], marker="s", s=100, zorder=4)
    else:
```

```
        Y = в(ʊ[ʑ - 3])
        п.plot(
            Y[:, 0],
            Y[:, 1],
            color=ц[ʑ],
            linewidth=2,
            label=э[ʑ],
            linestyle="--",
            alpha=0.1 if ʑ != 3 else 1,
        )
        п.scatter(Y[0, 0], Y[0, 1], color=ц[ʑ], marker="o", s=100, zorder=4)
        п.scatter(Y[-1, 0], Y[-1, 1], color=ц[ʑ], marker="s", s=100, zorder=4)

п.xlabel("x")
п.ylabel("y")
п.title("Trayectorias de Convergencia en la Función de Himmelblau")
п.legend()
п.grid(True, alpha=0.3)
п.axis("equal")
п.xlim(-5, 5)
п.ylim(-5, 5)

# Tabla comparativa
print("\n" + "=" * 90)
print("TABLA COMPARATIVA: GRADIENTE DESCENDENTE vs GRADIENTE DESCENDENTE CON MOMENTO")
print("=" * 90)
print(
        f"{'Algoritmo':<15}  {'Hiperparámetros':<20}  {'Mínimo  (x,y)':<25}  {'f(x,y)':<12}
{'Iteraciones':<12}"
)
print("-" * 90)

# Resultados DG
for j in range(3):
    print(
          f"{'DG':<15} {f'α={α[j]}':<20} ({float(e[j][0]):<10.6f}, {float(e[j][1]):<11.6f})
{f(e[j]):<12.3e} {i[j]:<12}"
    )

# Resultados DGM
for j in range(3):
    print(
            f"{'DGM':<15}  {f'α={η},  β={β[j]}':<20}  ({float(ε[j][0]):<10.6f},  {float(ε[j]
[1]):<11.6f}) {f(ε[j]):<12.2e} {ι[j]:<12}"
    )

print("=" * 90)

п.show()
```

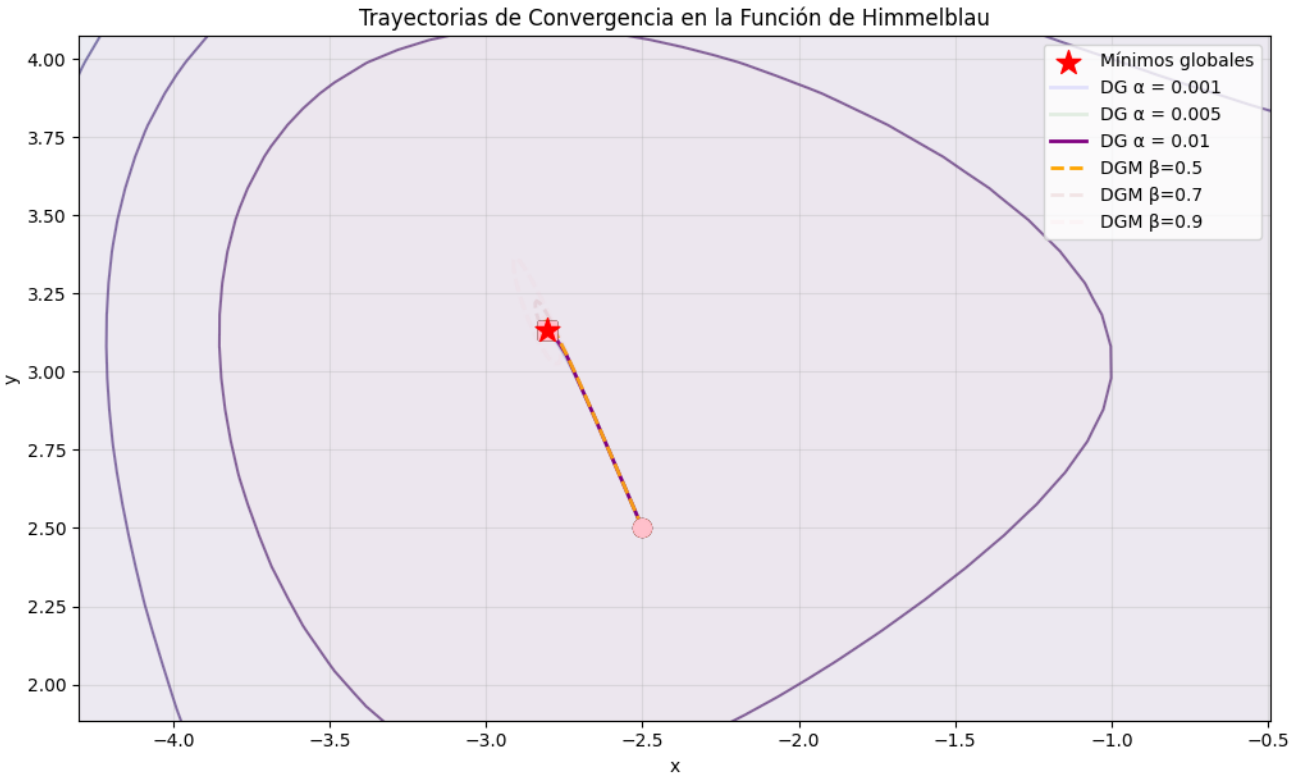
3. Análisis y visualización:
   - Genere una tabla comparativa que muestre para cada configuración: el algoritmo, los hiperparámetros usados, el mínimo encontrado, el valor de la función en ese mínimo y el número de iteraciones requeridas.

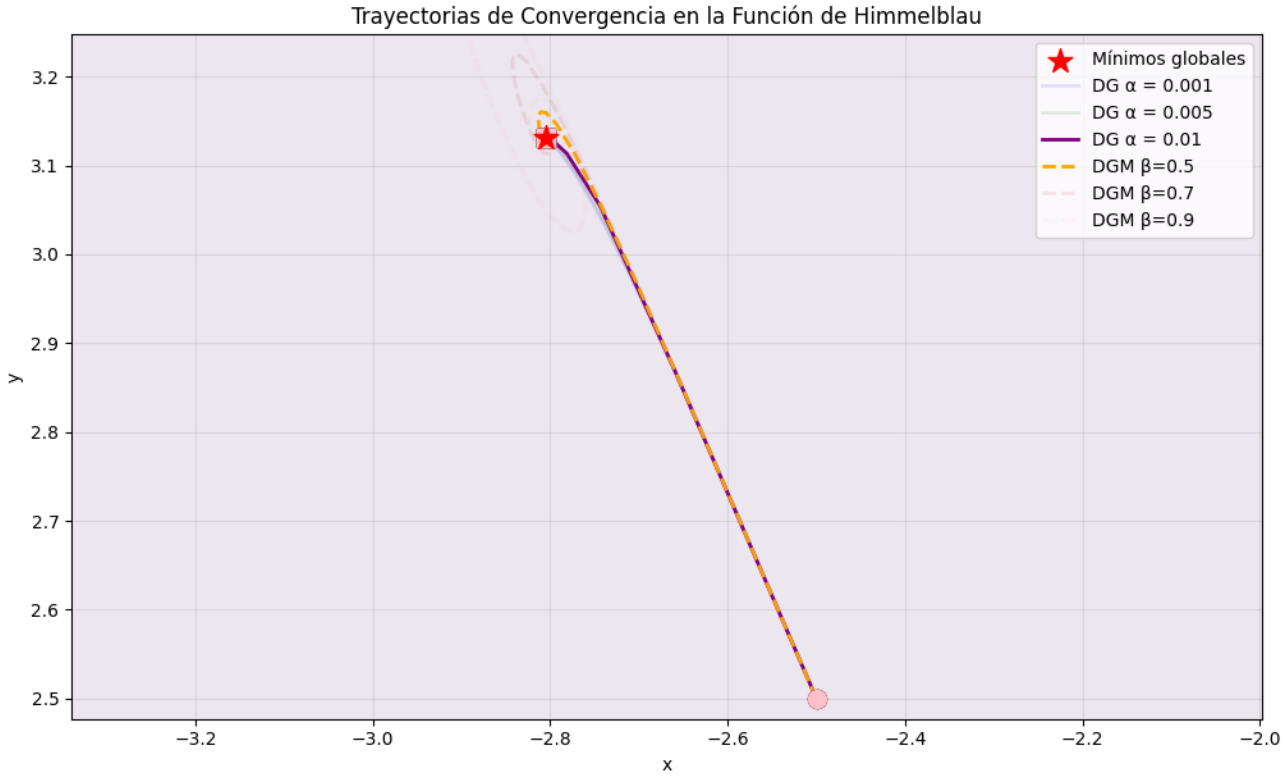
```
================================================================================================
TABLA COMPARATIVA: GRADIENTE DESCENDENTE vs GRADIENTE DESCENDENTE CON MOMENTO
================================================================================================
Algoritmo       Hiperparámetros      Mínimo (x,y)              f(x,y)       Iteraciones
------------------------------------------------------------------------------------------------
DG              α=0.001              (-2.804961 , 3.131256   ) 9.214e-07    118
DG              α=0.005              (-2.804993 , 3.131282   ) 5.411e-07    21
DG              α=0.01               (-2.804978 , 3.131296   ) 6.434e-07    8
DGM             α=0.005, β=0.5       (-2.805062 , 3.131431   ) 6.77e-07     25
DGM             α=0.005, β=0.7       (-2.804985 , 3.131215   ) 9.45e-07     43
DGM             α=0.005, β=0.9       (-2.805105 , 3.131250   ) 1.64e-07     135
================================================================================================
```

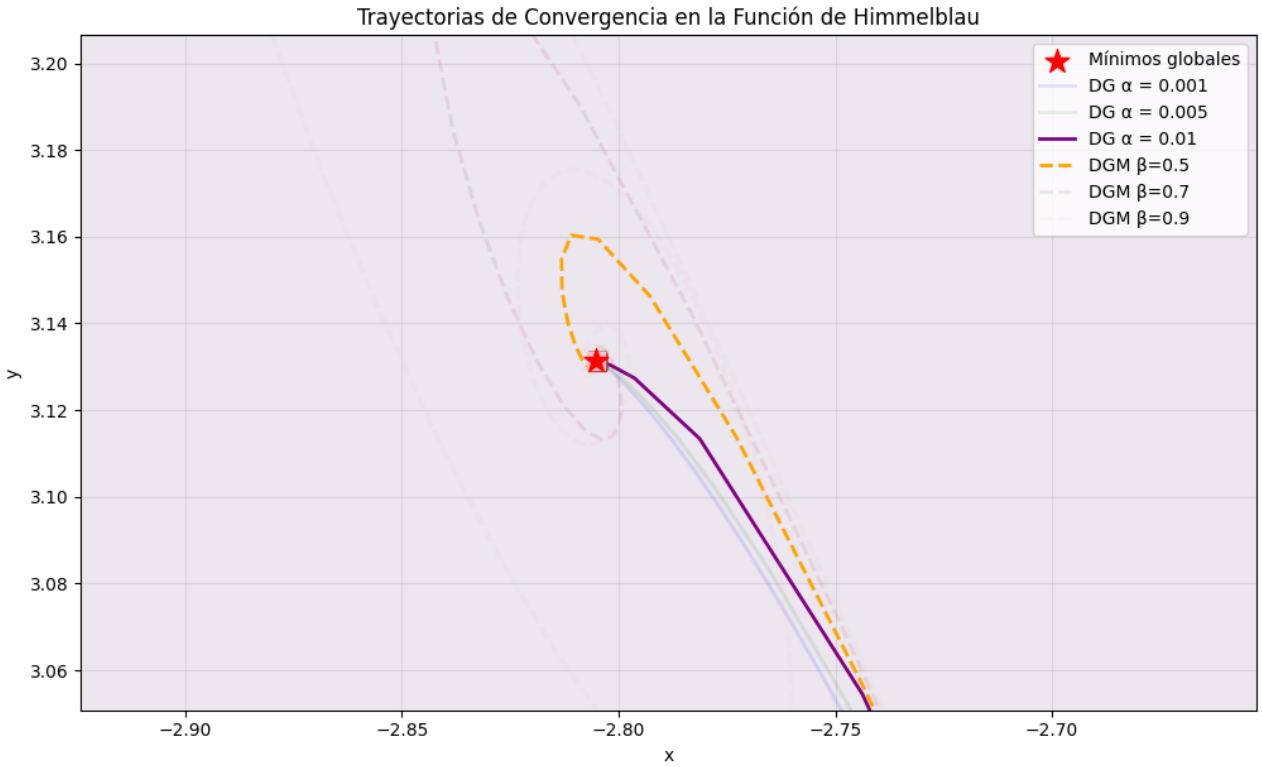
   - Genere un gráfico de contorno de la función de Himmelblau. Sobre este, grafique las trayectorias de convergencia para la mejor configuración de GD y la mejor configuración de GDM.



Aunque se pide sólo la graficación de los mejores resultados de cada conjunto de algoritmos, de igual manera se graficaron (con baja transparencia) los otros caminos.


Trayectorias de Convergencia en la Función de Himmelblau
Mínimos globales
DG α = 0.001
DG α = 0.005
DG α = 0.01
DGM β=0.5
DGM β=0.7
DGM β=0.9
4.00
3.75
3.50
3.25
3.00
2.75
2.50
2.25
2.00
y
−4.0
−3.5
−3.0
−2.5
−2.0
−1.5
−1.0
−0.5
x


Trayectorias de Convergencia en la Función de Himmelblau
Mínimos globales
DG α = 0.001
DG α = 0.005
DG α = 0.01
DGM β=0.5
DGM β=0.7
DGM β=0.9
3.2
3.1
3.0
2.9
2.8
2.7
2.6
2.5
y
−3.2
−3.0
−2.8
−2.6
−2.4
−2.2
−2.0
x



- Escriba una conclusión breve explicando qué algoritmo convergió más eficientemente y por qué el momento ayuda (o perjudica) en este caso particular

El algoritmo que convergió más eficientemente fue el descenso del gradiente tradicional con la tasa de aprendizaje igual a 0.01, la cual consiguió caer por debajo de la tolerancia establecida en sólo 8 iteraciones, y el motivo es relativamente simple: el valor de α alto permite descender el gradiente de manera rápida sin pasarse. Directo al objetivo, la curva incluso se puede notar algo cruda, con giros puntiagudos debido a que realizó muy pocos saltos a diferencia de los otros descensos tradicionales.

Por otro lado, para este caso particular, el momento es un incordio, pues el momento causa que la curva en todos los casos falle la caída hacia el mínimo, lo cual sería práctico en funciones con mínimos locales para perseguir el mínimo global, aquí causa que el algoritmo de vueltas de manera innecesaria en torno al verdadero máximo. Un mejor resultado se obtiene al reducir el factor de momento, lo cual es lamentable, porque eso significa realizar igualmente los cálculos extra para momento cuando estos realmente aportan poco. Para la función de Himmelblau, el descenso tradicional con la tasa de aprendizaje de 0.01 es superior.

Pregunta 2

Una tasa de aprendizaje fija no siempre es óptima. Las estrategias de decaimiento pueden acelerar la convergencia y evitar oscilaciones cerca del mínimo. El objetivo, es implementar y comparar tres estrategias diferentes de decaimiento de la tasa de aprendizaje para minizar la función de Rosenbrock:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

Actividades:

1. Implementación:
   - Modifique el algoritmo de descenso de gradiente para incorporar un planificador de tasa de aprendizaje.
   - Implemente las siguientes tres estrategias de decaimiento para αk en la iteración k:
     1. Decaimiento por tiempo: $\alpha k = \alpha 0 \,/\, 1+k \cdot \tau$
     2. Decaimiento por pasos: $\alpha k = \alpha 0 \cdot d^{\lfloor k/s \rfloor}$, donde d es un factor de decaimiento y s es el tamaño del paso
     3. Decaimiento exponencial: $\alpha k = \alpha 0 \cdot e^{(-k \cdot \tau)}$
2. Experimentación:
   - Minimice la función de Rosenbrock partiendo desde $(-1, -1)$.
   - Utilice los siguientes parámetros: $\alpha 0 = 0.002$, max_iter = 10000.
   - Para el decaimiento por tiempo y exponencial, use: $\tau = \alpha 0 \,/\, 10000$.
   - Para el decaimiento por pasos, use un factor $d = 0.5$ y un tamaño de paso $s = 2000$.
   - Compare estos tres métodos con el descenso de gradiente con α fijo de 0.002

Código Fuente de la Implementación

```
from numpy import array as в
from matplotlib import pyplot as п
from math import floor as ф, exp as э

e = [в([-1, -1]), в([-1, -1]), в([-1, -1]), в([-1, -1])]
α = [0.002, 0.002, 0.002, 0.002]
d = 0.5
s = 2000
τ = α[0] / 10_000
i = [0, 0, 0, 0]
h = [[], [], [], []]
η = [[], [], [], []]

#  f(x, y) = (1 - x)² + 100(y - x²)²
f = lambda x: (1 - x[0]) ** 2 + 100 * (x[1] - x[0] ** 2) ** 2
# ∇f(x, y) = [ -2(1 - x) - 400x(y - x²), 200(y - x²) ]
φ = lambda x: в(
    [
        -2 * (1 - x[0]) - 400 * x[0] * (x[1] - x[0] ** 2),
        200 * (x[1] - x[0] ** 2),
    ]
)
# g(x, y, α) = (x, y) + α∇f(x, y)
g = lambda x, a: x - a * φ(x)

# Inicializar
h[0].append(f(e[0]))
h[1].append(f(e[1]))
h[2].append(f(e[2]))
h[3].append(f(e[3]))
η[0].append(α[0])
η[1].append(α[1])
η[2].append(α[2])
η[3].append(α[3])

for k in range(0, 10000, 1):
    e[0] = g(e[0], α[0])
    h[0].append(f(e[0]))
    η[0].append(α[0])
    i[0] += 1
    if f(e[0]) < 1e-6:
        break

for k in range(0, 10000, 1):
    α[1] = α[0] / (1 + k * τ)
    e[1] = g(e[1], α[1])
    h[1].append(f(e[1]))
    η[1].append(α[1])
    i[1] += 1
    if f(e[1]) < 1e-6:
        break

for k in range(0, 10000, 1):
    α[2] = α[0] * (d ** ф(k / s))
    e[2] = g(e[2], α[2])
    h[2].append(f(e[2]))
    η[2].append(α[2])
    i[2] += 1
    if f(e[2]) < 1e-6:
        break

for k in range(0, 10000, 1):
    α[3] = α[0] * э(-k * τ)
    e[3] = g(e[3], α[3])
    h[3].append(f(e[3]))
    η[3].append(α[3])
    i[3] += 1
    if f(e[3]) < 1e-6:
        break

F = п.figure(figsize=(15, 6))
```

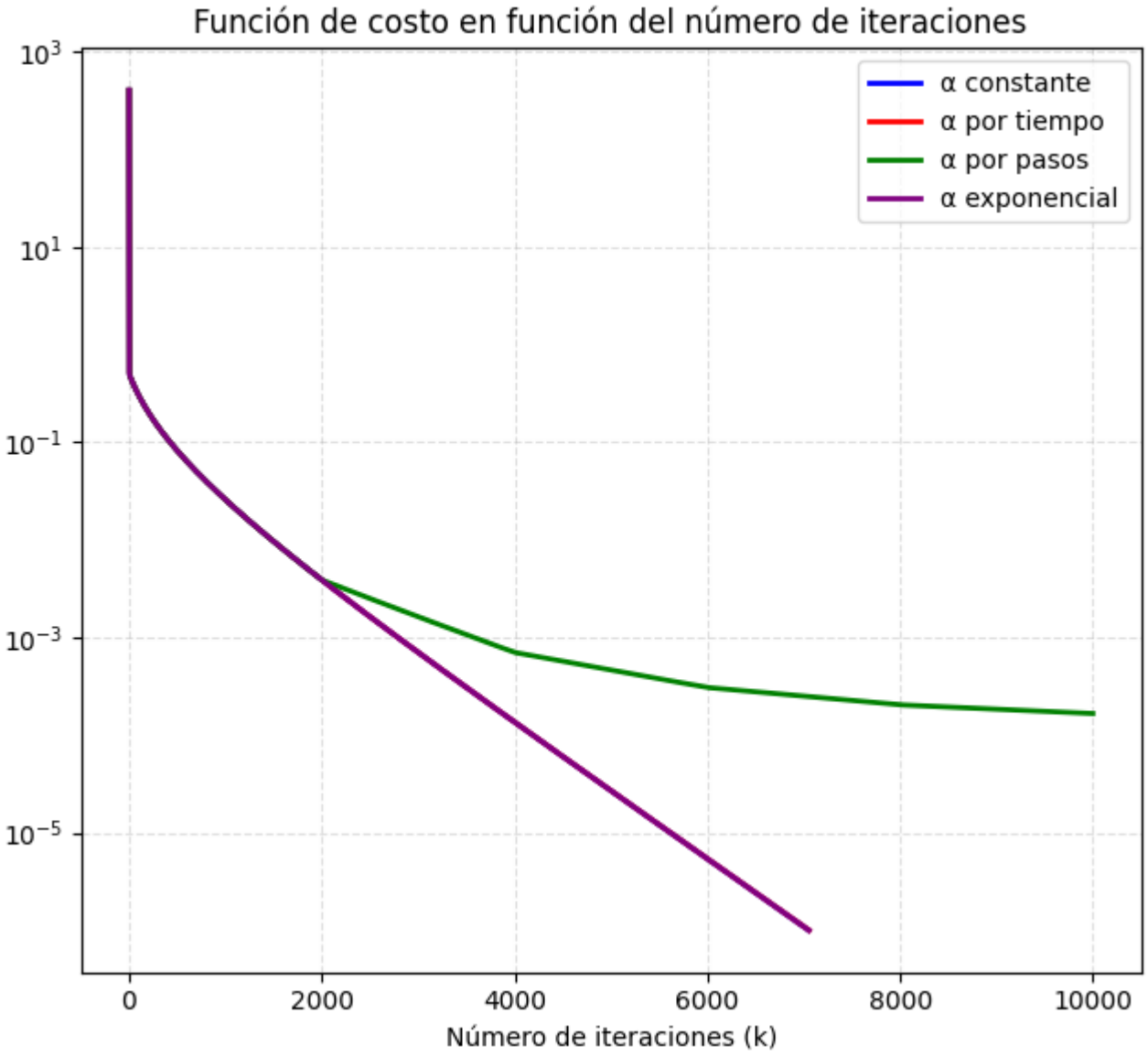
```
A = π.subplot(1, 2, 1)
A.grid(True, linestyle="--", alpha=0.4)
A.set_title("α en función del número de iteraciones")
A.set_xlabel("Número de iteraciones (k)")
A.set_ylabel("Valor de α")

for j, (label, color) in enumerate(
    [
        ("α constante", "blue"),
        ("α por tiempo", "red"),
        ("α por pasos", "green"),
        ("α exponencial", "purple"),
    ]
):
    iterations = range(len(η[j]))
    A.plot(iterations, η[j], "-", label=label, color=color, linewidth=2)
A.legend()

B = π.subplot(1, 2, 2)
B.grid(True, linestyle="--", alpha=0.4)
B.set_title("Función de costo en función del número de iteraciones")
B.set_xlabel("Número de iteraciones (k)")
B.set_ylabel("Valor de la función de costo")
B.set_yscale("log")  # Use log scale for better visualization

for j, (label, color) in enumerate(
    [
        ("α constante", "blue"),
        ("α por tiempo", "red"),
        ("α por pasos", "green"),
        ("α exponencial", "purple"),
    ]
):
    iterations = range(len(h[j]))
    B.plot(iterations, h[j], "-", label=label, color=color, linewidth=2)

B.legend()

π.tight_layout()

# Tabla comparativa
print("\n" + "=" * 90)
print("TABLA COMPARATIVA: DESCENSO DEL GRADIENTE ESTÁTICO vs DINÁMICO")
print("=" * 90)
print(
        f"{'Algoritmo':<15} {'Valores de α':<20} {'Mínimo (x,y)':<25} {'f(x,y)':<12}
{'Iteraciones':<12}"
)
print("-" * 90)

# Resultados
for j, k in enumerate(["α constante", "α por tiempo", "α por pasos", "α exponencial"]):
    print(
        f"{k:<15} {f'α = {α[j]:.6f}':<20} ({float(e[j][0]):<10.6f}, {float(e[j][1]):<11.6f})
{f(e[j]):<12.3e} {i[j]:<12}"
    )

print("=" * 90)

π.show()
```

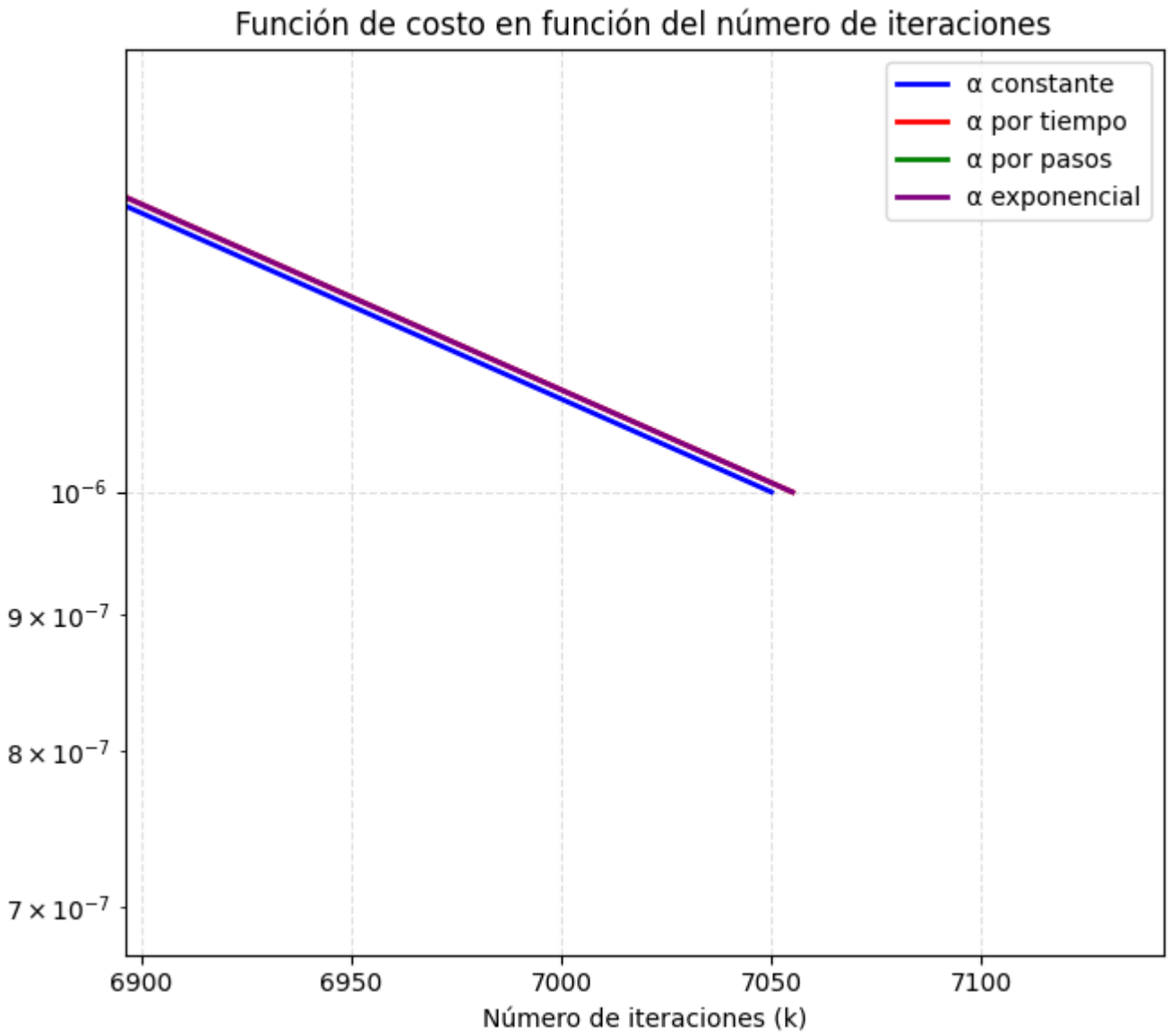
3. Análisis y visualización:
   - Genere una tabla comparando las cuatro estrategias (incluida la de α fijo), mostrando el punto final alcanzado y el número de iteraciones necesarias para converger.

```
================================================================================================
TABLA COMPARATIVA: DESCENSO DEL GRADIENTE ESTÁTICO vs DINÁMICO
================================================================================================
Algoritmo       Valores de α         Mínimo (x,y)              f(x,y)       Iteraciones
------------------------------------------------------------------------------------------------
α constante     α = 0.002000         (0.999001  , 0.997999   ) 9.998e-07    7050
α por tiempo    α = 0.001997         (0.999001  , 0.997999   ) 9.997e-07    7055
α por pasos     α = 0.000125         (0.987124  , 0.974362   ) 1.661e-04    10000
α exponencial   α = 0.001997         (0.999001  , 0.997999   ) 9.997e-07    7055
================================================================================================
```

Se puede apreciar cómo el algoritmo de actualización de la tasa de aprendizaje por pasos ni siquiera consigue converger, saliendo del bucle por acabarlo en lugar de optimizar el costo bajo la tolerancia elegida.

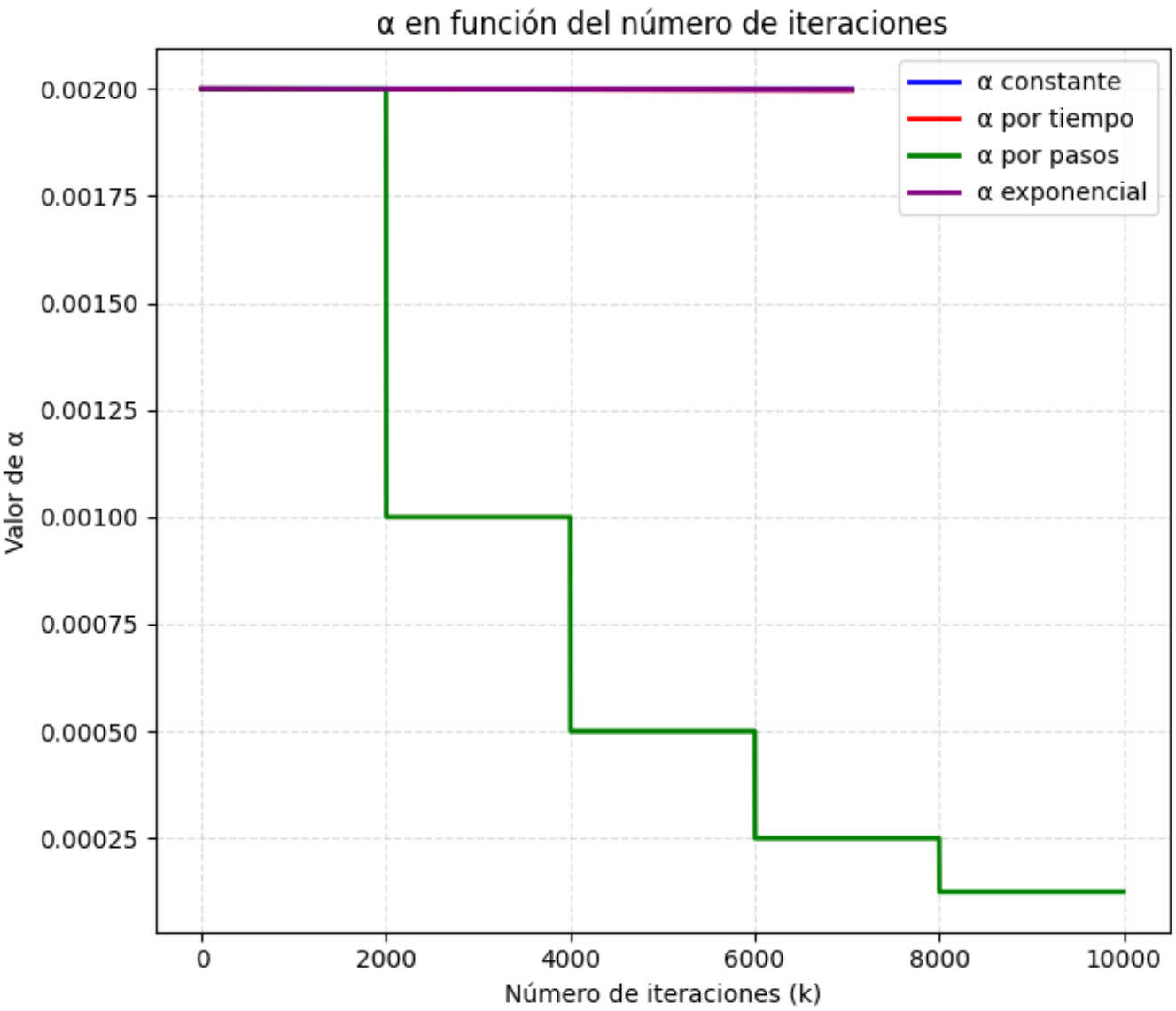
   - Cree un gráfico que muestre la evolución del valor de la función objetivo $f(x, y)$ a lo largo de las iteraciones para cada una de las cuatro estrategias. Use una escala logarítmica en el eje Y para una mejor visualización.

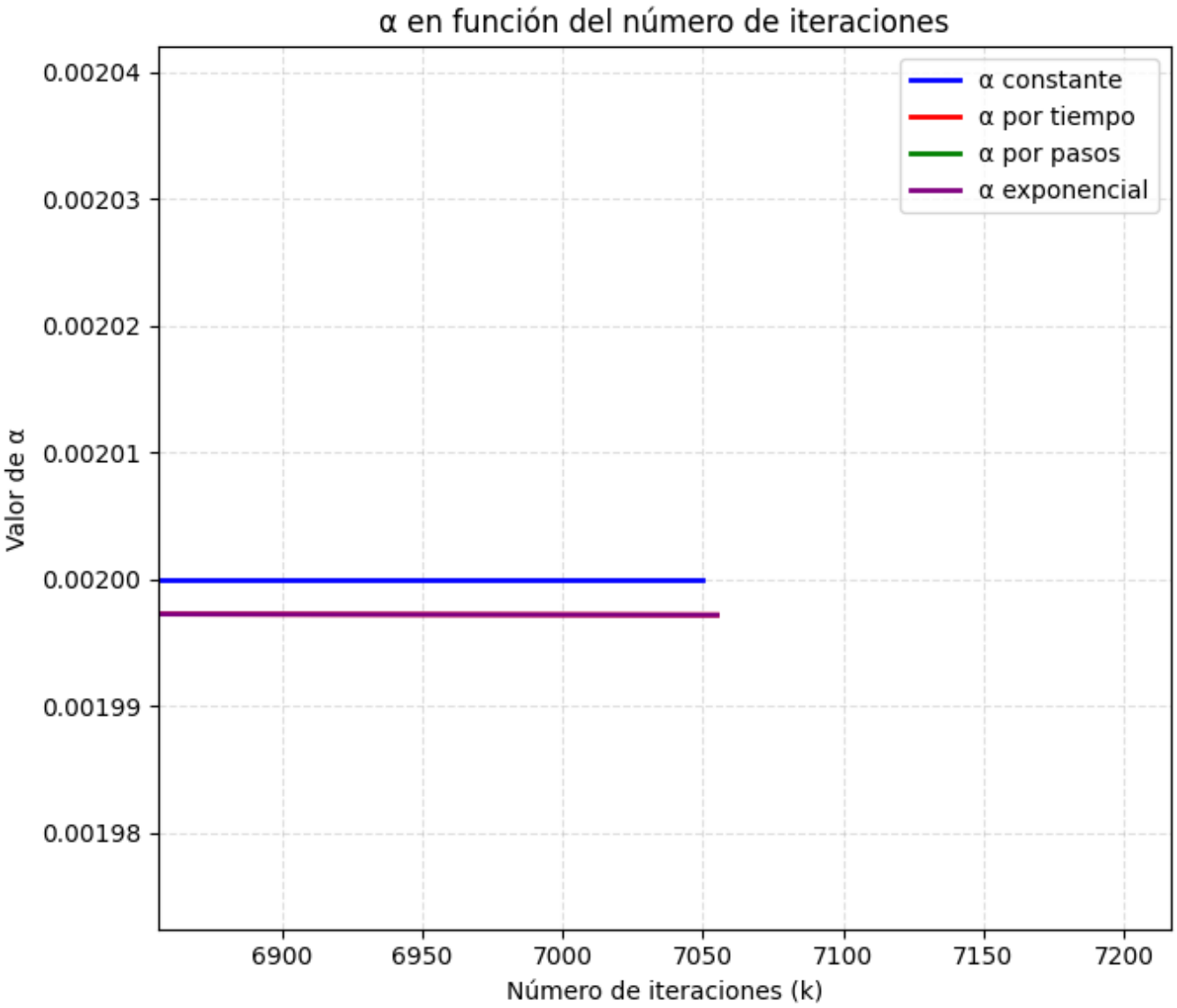

Función de costo en función del número de iteraciones
α constante
α por tiempo
α por pasos
α exponencial
10^{3}
10^{1}
10^{-1}
10^{-3}
10^{-5}
0
2000
4000
6000
8000
10000
Número de iteraciones (k)



Zoom al gráfico del valor de la función de costo en función del número de iteraciones para poder visualizar cómo la tasa de aprendizaje constante consigue un mejor desempeño, además de que la tasa por tiempo y exponencial coinciden perfectamente, superponiéndose en el gráfico.

- En un segundo gráfico, muestre cómo cambia el valor de la tasa de aprendizaje α_k en cada iteración para las tres estrategias de decaimiento.


α en función del número de iteraciones
α constante
α por tiempo
α por pasos
α exponencial
0.00200
0.00175
0.00150
0.00125
0.00100
0.00075
0.00050
0.00025
Valor de α
0
2000
4000
6000
8000
10000
Número de iteraciones (k)



- Concluya cuál estrategia de decaimiento funcionó mejor para este problema y por qué.

La mejor estrategia resulta que es no utilizar ningún algoritmo de decaimiento, ¿Por qué? Pues porque tener un valor de α que decae significa mayor dificultada para moverse sobre la superficie tan plana de la función de Rosenbrock. Es por eso que sobre esta topología tan concreta, tener una tasa de aprendizaje constante proporciona un balance perfecto entre velocidad de convergencia y estabilidad, evitando la desaceleración prematura que arruina a algoritmos como el de pasos.

Aunque gráficamente pueda parecer que el algoritmo exponencial o decaimiento por tiempo tampoco tienen malos resultados, se debe considerar que estos algoritmos requieren cálculos extra para actualizar la tasa de aprendizaje, y por lo tanto se invierte más esfuezo en calcular opciones inferiores a la tasa constante.

Pregunta 3

El método de mínimos cuadrados usado en la regresión lineal es muy sensible a los valores atípicos (outliers), ya que el error se eleva al cuadrado, magnificando el impacto de los puntos lejanos. Una alternativa más robusta es minimizar el Error Absoluto Medio (MAE).

La función objetivo MAE es:

$$f(\beta_0, \beta_1) = \frac{1}{m} \sum_{i=1}^{m} |y_i - (\beta_0 + \beta_1 x_i)|$$

Esta función no es diferenciable en todos sus puntos debido a la función de valor absoluto. Sin embargo, podemos usar su subgradiente para aplicar el descenso de gradiente.
El subgradiente de la función objetivo es:

$$\frac{\partial f}{\partial \beta_j} = \frac{1}{m} \sum_{i=1}^{m} x_{ij} \cdot \text{sign}(y_i - \hat{y}_i)$$

donde sign(z) es -1 si $z < 0$, $+1$ si $z > 0$, y 0 si $z = 0$.
El objetivo es implementar un algoritmo de regresión lineal robusto utilizando el descenso de sub-gradiente para minimizar el MAE y compararlo con el modelo estándar de Mínimos Cuadrados en presencia de datos atípicos.

Actividades:

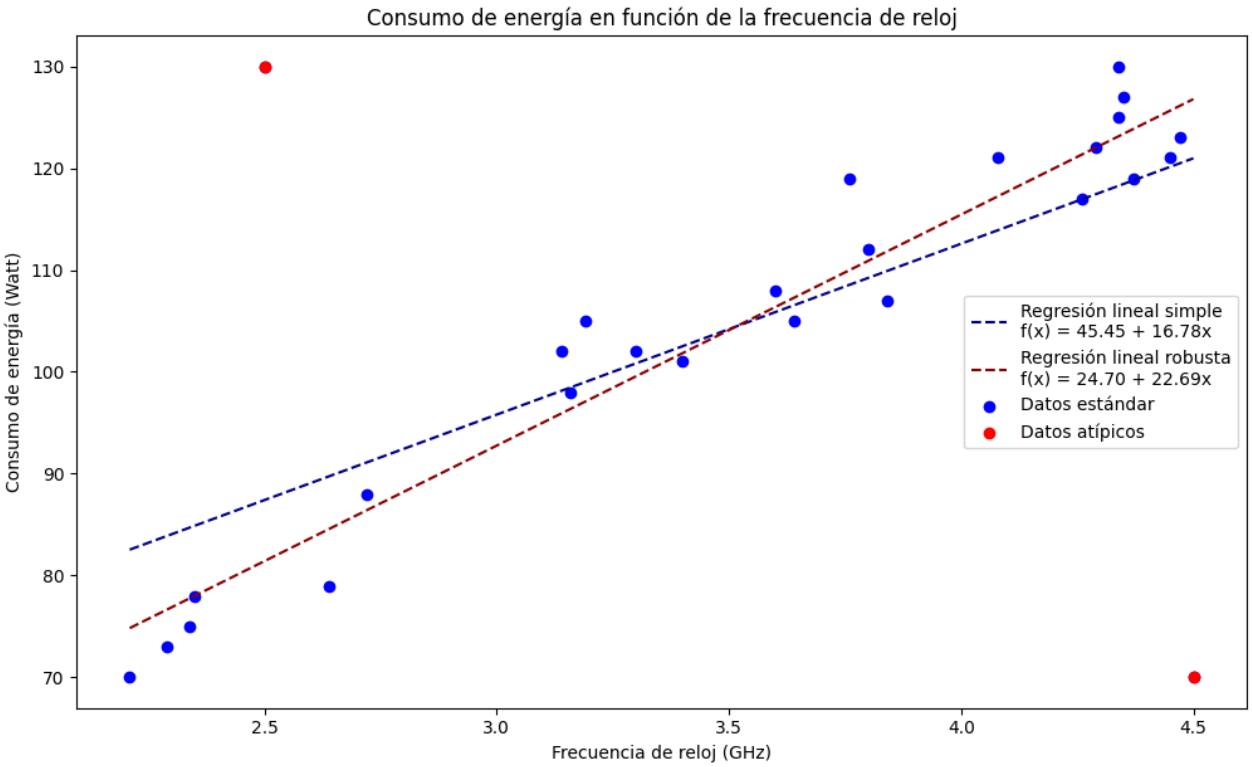
1. Datos:
   - Utilice los datos de frecuencia_reloj y consumo_energia del apunte de clases.
   - Introduzca dos valores atípicos (outliers) en los datos para simular errores de medición, para la frecuencia de reloj [2.5, 4.5] y para el consumo de energia [130, 70].
2. Implementación:
   - Defina la función objetivo MAE y su subgradiente.
   - Implemente un algoritmo de descenso de gradiente que utilice el subgradiente MAE para encontrar los parámetros β óptimos.

Código Fuente de la Implementación

```
from numpy import (
    array as в,
    transpose as т,
    sum as д,
    sign as ш,
    column_stack as к,
    ones as ö,
    linspace as л,
)
from matplotlib import pyplot as п

f = в(
    [
        4.29,
        4.34,
        2.72,
        4.08,
        3.60,
        3.30,
        3.84,
        2.34,
        3.64,
        3.76,
        3.14,
        3.80,
        4.34,
        2.64,
        3.16,
        4.35,
        4.45,
        2.29,
        3.19,
        3.40,
        4.26,
        2.35,
        4.47,
        4.37,
        2.21,
        2.5,
        4.5,
    ]
)
f = к([ö(f.shape[0]), f[:, None]])
c = в(
    [
        122,
        130,
        88,
        121,
        108,
        102,
        107,
        75,
        105,
        119,
        102,
        112,
        125,
        79,
        98,
        127,
        121,
        73,
        105,
        101,
        117,
        78,
        123,
        119,
        70,
        130,
        70,
```

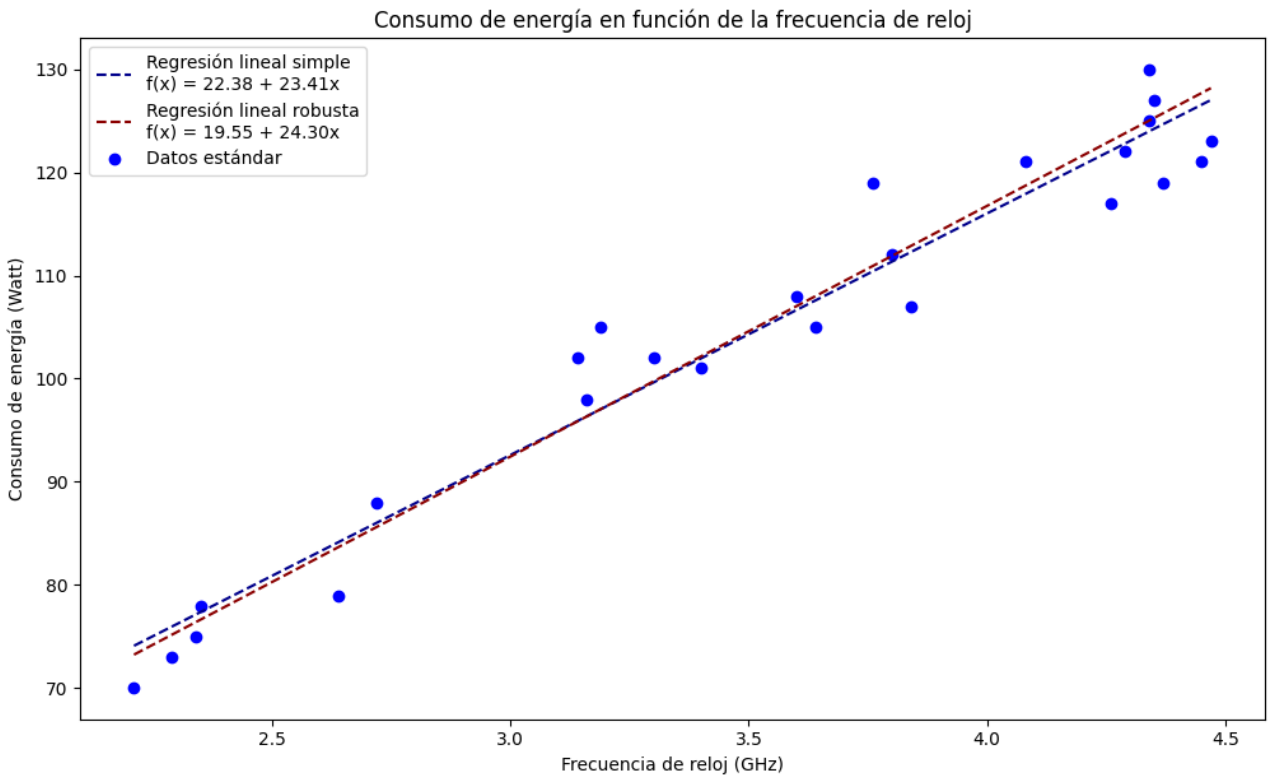
```
    ]
)

ψ = в([2.5, 4.5])
ψ = к([ö(ψ.shape[0]), ψ[:, None]])
ψ = в([130, 70])

θ = [в([0, 0]), в([0, 0])]
α = 0.01

h = lambda Θ: f @ Θ
J = lambda Θ: д((h(Θ) - c) ** 2) / (2 * f.shape[0])
j = lambda Θ: (т(f) @ (h(Θ) - c)) / f.shape[0]
Й = lambda Θ: д(abs(c - h(Θ))) / f.shape[0]
й = lambda Θ: (т(f) @ ш(h(Θ) - c)) / f.shape[0]

l = л(f[:, 1].min(), f[:, 1].max(), 100)

for _ in range(100000):
    θ[0] = θ[0] - α * j(θ[0])

for _ in range(100000):
    θ[1] = θ[1] - α * й(θ[1])

п.figure(figsize=(12, 8))
п.plot(
    l,
    θ[0][1] * l + θ[0][0],
    color="darkblue",
    linestyle="--",
    label=f"Regresión lineal simple\nf(x) = {θ[0][0]:.2f} + {θ[0][1]:.2f}x",
)
п.plot(
    l,
    θ[1][1] * l + θ[1][0],
    color="darkred",
    linestyle="--",
    label=f"Regresión lineal robusta\nf(x) = {θ[1][0]:.2f} + {θ[1][1]:.2f}x",
)
п.scatter(f[:, 1], c, color="blue", marker="o", label="Datos estándar")
п.scatter(ψ[:, 1], ψ, color="red", marker="o", label="Datos atípicos")
п.xlabel("Frecuencia de reloj (GHz)")
п.ylabel("Consumo de energía (Watt)")
п.title("Consumo de energía en función de la frecuencia de reloj")
п.tick_params()
п.legend()
п.show()
```

3. Experimentación y análisis:
   - Ejecute el algoritmo de regresión lineal original (basado en MSE) sobre el conjunto de datos con outliers.
   - Ejecute el nuevo algoritmo de regresión robusta (basado en MAE) sobre el mismo conjunto de datos con outliers.
   - Gráfico: Cree un único diagrama de dispersión que muestre los datos con los outliers. Sobre este gráfico, dibuje las dos rectas de regresión obtenidas:
     1. La recta del modelo de Mínimos Cuadrados (MSE).
     2. La recta del modelo de Error Absoluto Medio (MAE).



   - Análisis: En una breve conclusión, explique cuál de las dos rectas se ajusta mejor a la tendencia general de los datos originales. ¿Por qué el modelo basado en MAE es considerado más "robusto" ante valores atípicos? ¿Cómo se relaciona esto con la penalización que cada función de objetivo impone a los errores grandes?

El ejercicio nos pide trabajar en todo momento *con* los datos atípicos, pero si para responder la pregunta "¿Cuál de las dos rectas se ajusta mejor a la tendencia general de los *datos originales*?", entonces haciendo unas ligeras modificaciones al algoritmo para quitar los datos atípicos podemos encontrar el siguiente gráfico:



No es difícil apreciar cómo la regresión lineal robusta se adapta mejor al caso con datos atípicos, pero es que también obtiene mejor precisión que la regresión lineal simple en el caso *sin datos atípicos*, demostrando que en términos de robustés, sí, el algoritmo es notoriamente mejor en todo aspecto.

Los datos atípicos dentro del algoritmo simple se someten a una exponenciación, lo cual causa que los datos de mayor magnitud tengan un mayor peso a la hora desviar la función final. El algoritmo MAE tiene una gran ventaja aquí, jerarquizando los pesos al no utilizar el mecanismo de exponenciación, y en su lugar utilizando un valor absoluto (que realmente podría interpretarse como “calcular la raíz cuadrada de los datos exponenciados”, lo cual reduce nuevamente el boom que los datos atípicos obtienen en sus pesos) lo cual mantiene los datos más a raya, y convergiendo notoriamente más rápido a regresiones más precisas sin la necesidad de muchos más calculos que una MSE de toda la vida.