Trabajo Práctico N°2

Algoritmos Evolutivos (2024)

Tema: PSO

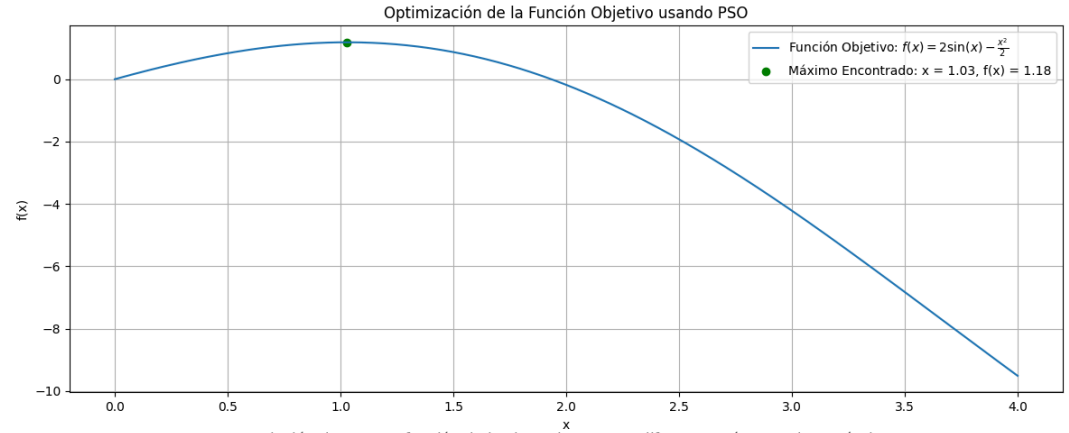
Alumno: Paz, Martin

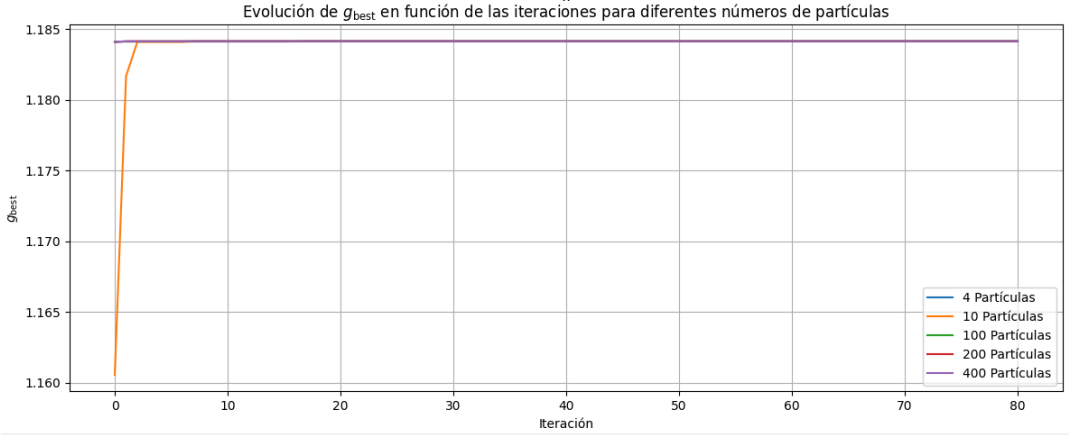
Link del repositorio: https://github.com/freischarler/ceia_algevo/tree/main/tp2

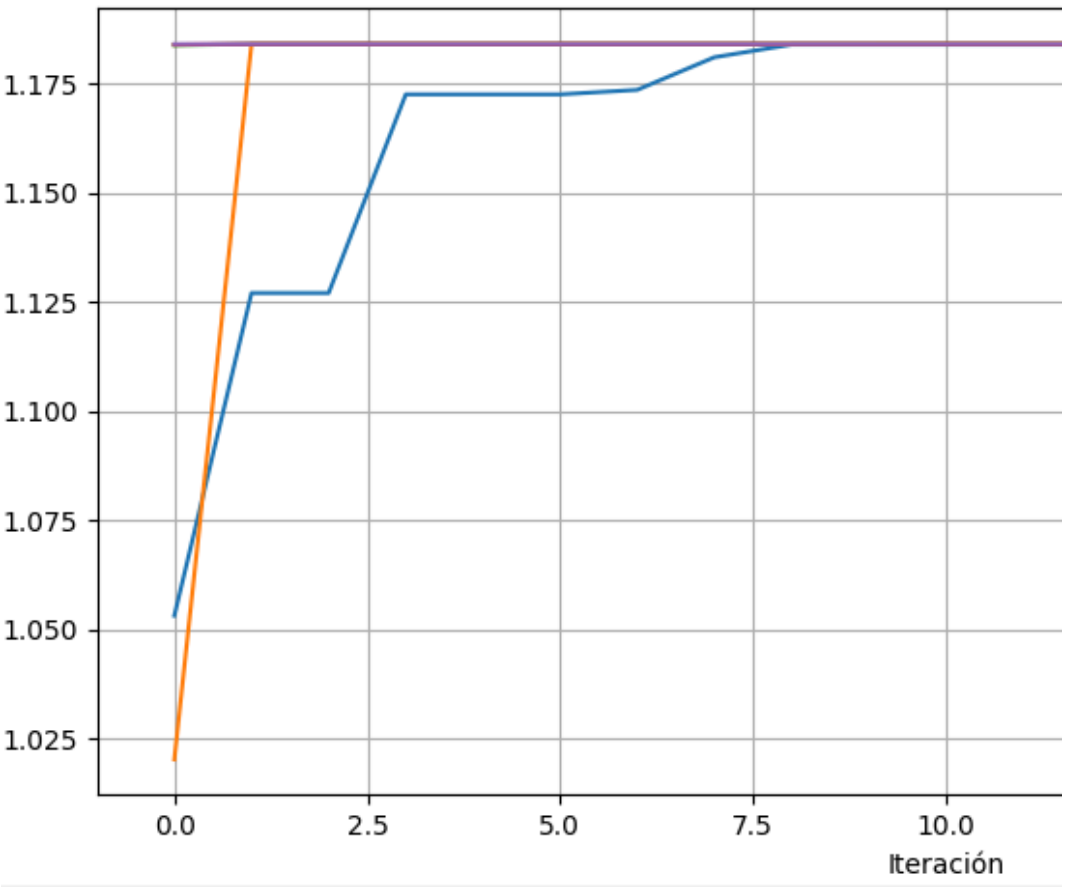
Ejercicio 1: https://github.com/freischarler/ceia_algevo/blob/main/tp2/1-main.py

1. Inicialización:
    - Se definen los parámetros del PSO: número máximo de iteraciones (max_iterations), coeficientes de aceleración (c1 y c2), y el factor de inercia (w).
    - Se inicializan las partículas con posiciones aleatorias dentro del intervalo [0,4] y velocidades iniciales a cero.
    - Se evalúa la función objetivo para cada partícula y se establece el mejor valor personal (personal_best) y el mejor valor global (global_best).
2. Bucle de Optimización:
    - Para cada iteración, se actualiza la velocidad y la posición de cada partícula utilizando las siguientes fórmulas:
        - v[i] = w * v[i] + c1 * r1 * (personal_best_x[i] - x[i]) + c2 * r2 * (global_best_x - x[i])
        - x[i] += v[i]

    - Las posiciones de las partículas se restringen al intervalo [0, 4].
    - Se evalúa la función objetivo para las nuevas posiciones y se actualizan los mejores valores personales y globales si se encuentra una mejor solución.
3. Almacenamiento de Resultados:
    - Se ejecuta el PSO para diferentes números de partículas (4, 10, 100, 200, 400) y se almacenan las historias de los mejores valores globales (gbest_history).
4. Visualización:
    - Se grafican los resultados, mostrando la función objetivo y el punto máximo encontrado, así como la evolución del mejor valor global en función de las iteraciones para diferentes números de partículas.


Optimización de la Función Objetivo usando PSO
Función Objetivo: $f(x) = 2\sin(x) - \frac{x^2}{2}$
Máximo Encontrado: x = 1.03, f(x) = 1.18
f(x)
x


Evolución de g_{best} en función de las iteraciones para diferentes números de partículas
g_{best}
Iteración
4 Partículas
10 Partículas
100 Partículas
200 Partículas
400 Partículas


Iteración

Según el gráfico de evolución de g best, a mayor número de partículas, se llega más rápido al valor global en función de las iteraciones, siendo de poca diferencia las cantidades mayores a 10 partículas.

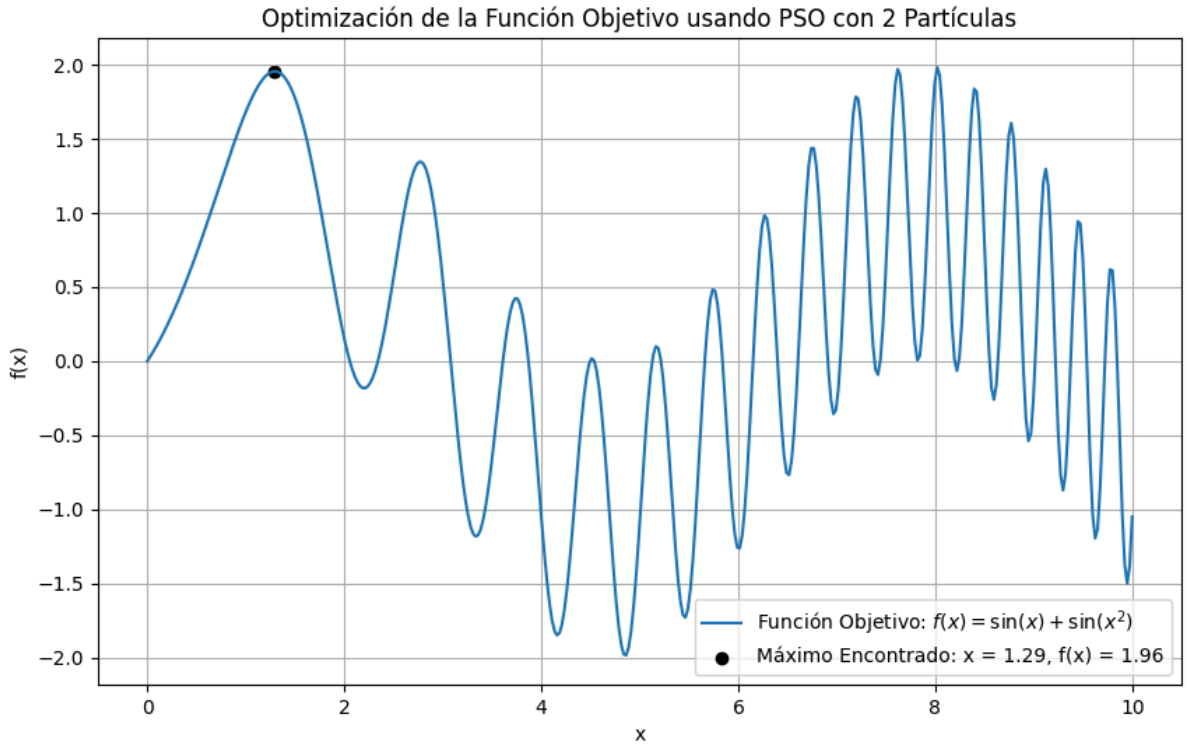
Ejercicio 2: https://github.com/freischarler/ceia_algevo/blob/main/tp2/2-main.py

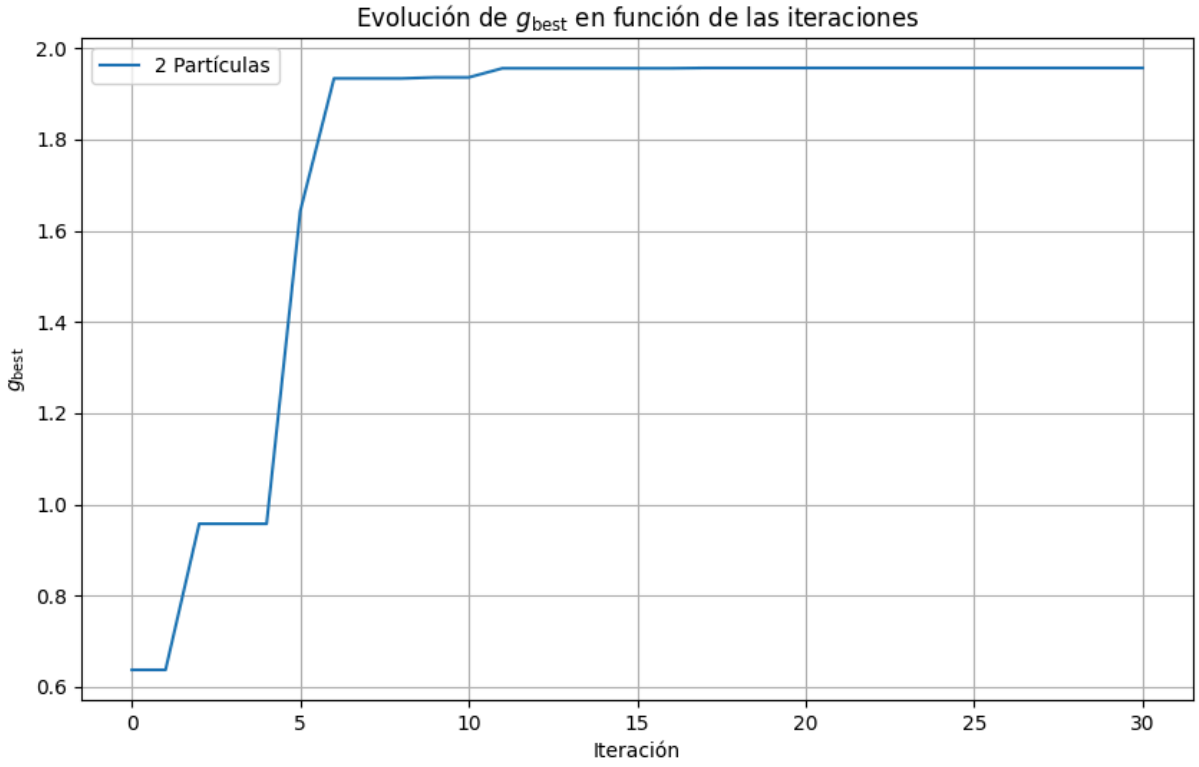
1. Definición de la Función Objetivo:
    - La función objetivo es f(x) = sin(x) + sin(x^2).
2. Inicialización de Parámetros:
    - Se definen los parámetros del PSO: número máximo de iteraciones (max_iterations), coeficientes de aceleración (c1 y c2), y el factor de inercia (w).
3. Inicialización de Partículas:
    - Se inicializan las posiciones (x) y velocidades (v) de las partículas de manera aleatoria.
    - Se almacenan las mejores posiciones personales (personal_best_x) y sus valores de fitness (personal_best_fitness).
    - Se determina la mejor posición global (global_best_x) y su valor de fitness (global_best_fitness).
4. Bucle de Optimización:
    - Para cada iteración, se actualizan las velocidades y posiciones de las partículas.
    - Las posiciones se restringen al intervalo [0, 10].
    - Se evalúa la función objetivo para cada partícula.
    - Se actualizan las mejores posiciones personales y, si es necesario, la mejor posición global.
    - Se almacena el historial de global_best_fitness.
5. Visualización de Resultados:
    - Se grafican la función objetivo y la evolución de global_best_fitness a lo largo de las iteraciones.
    - Se ejecuta el PSO con diferentes números de partículas y se muestran los resultados.

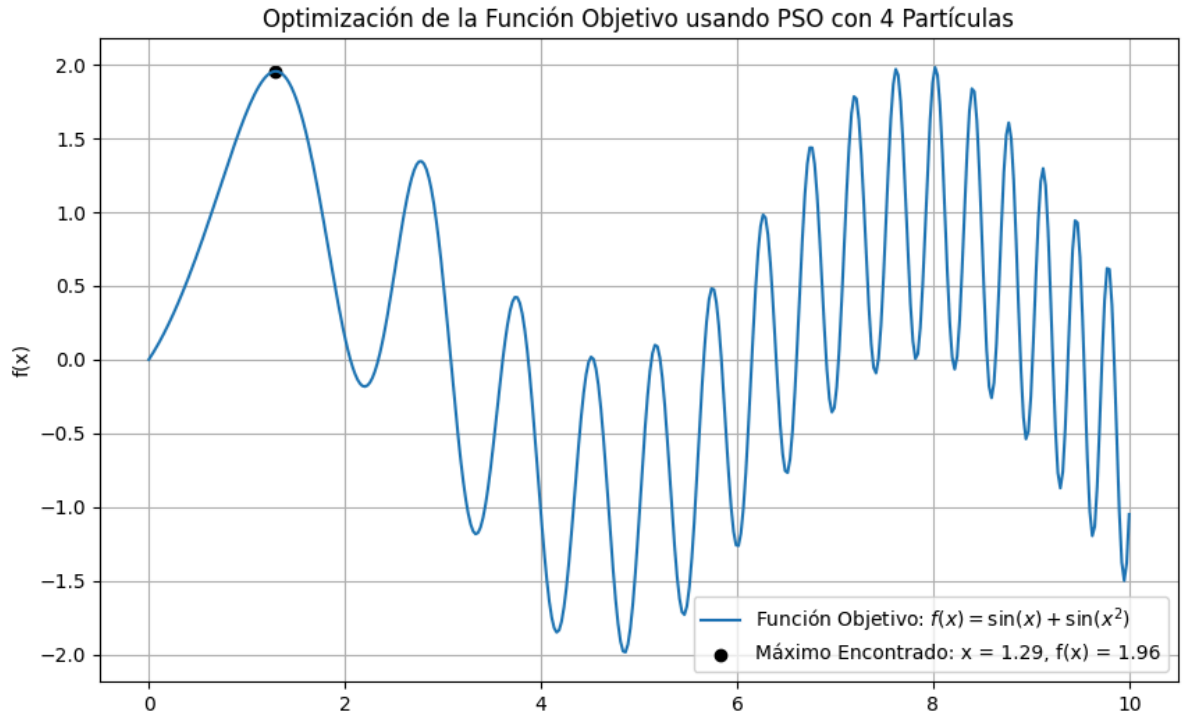
Resultados:

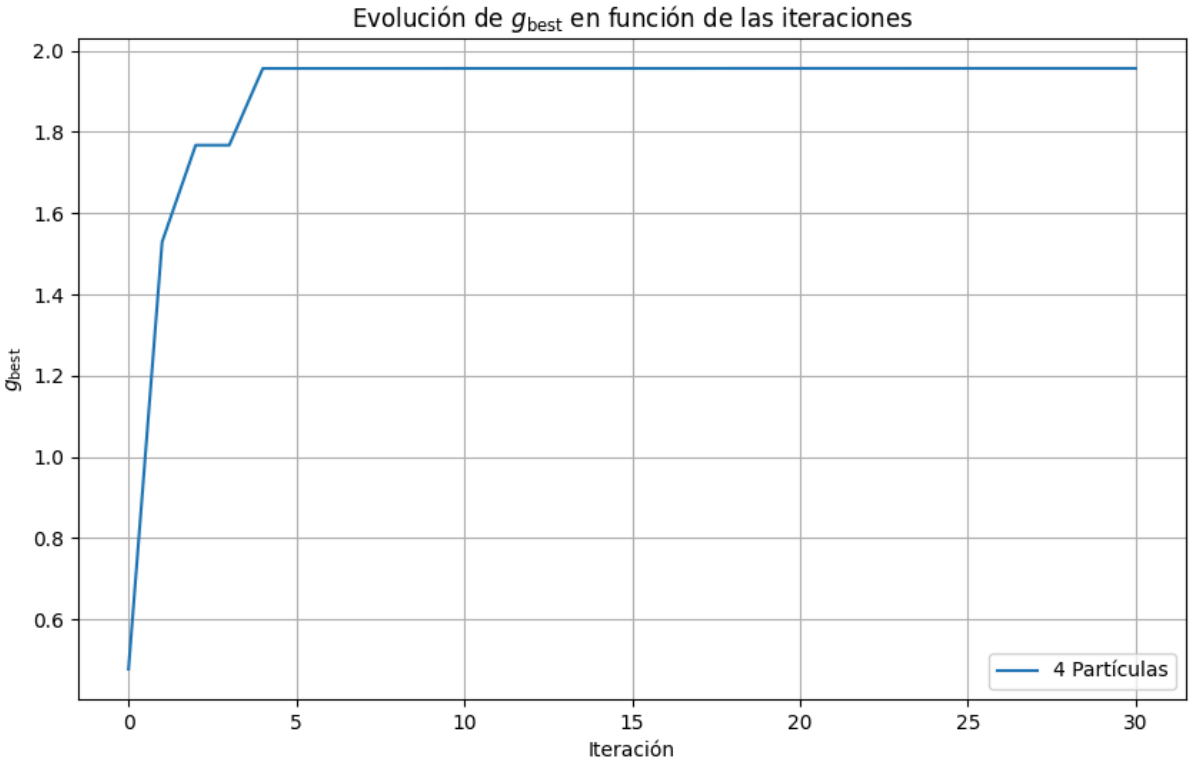
Número de partículas: 2
Solución óptima encontrada: x = 1.29
Valor objetivo óptimo: f(x) = 1.96
Número de partículas: 4
Solución óptima encontrada: x = 1.29
Valor objetivo óptimo: f(x) = 1.96
Número de partículas: 6
Solución óptima encontrada: x = 8.02
Valor objetivo óptimo: f(x) = 1.99

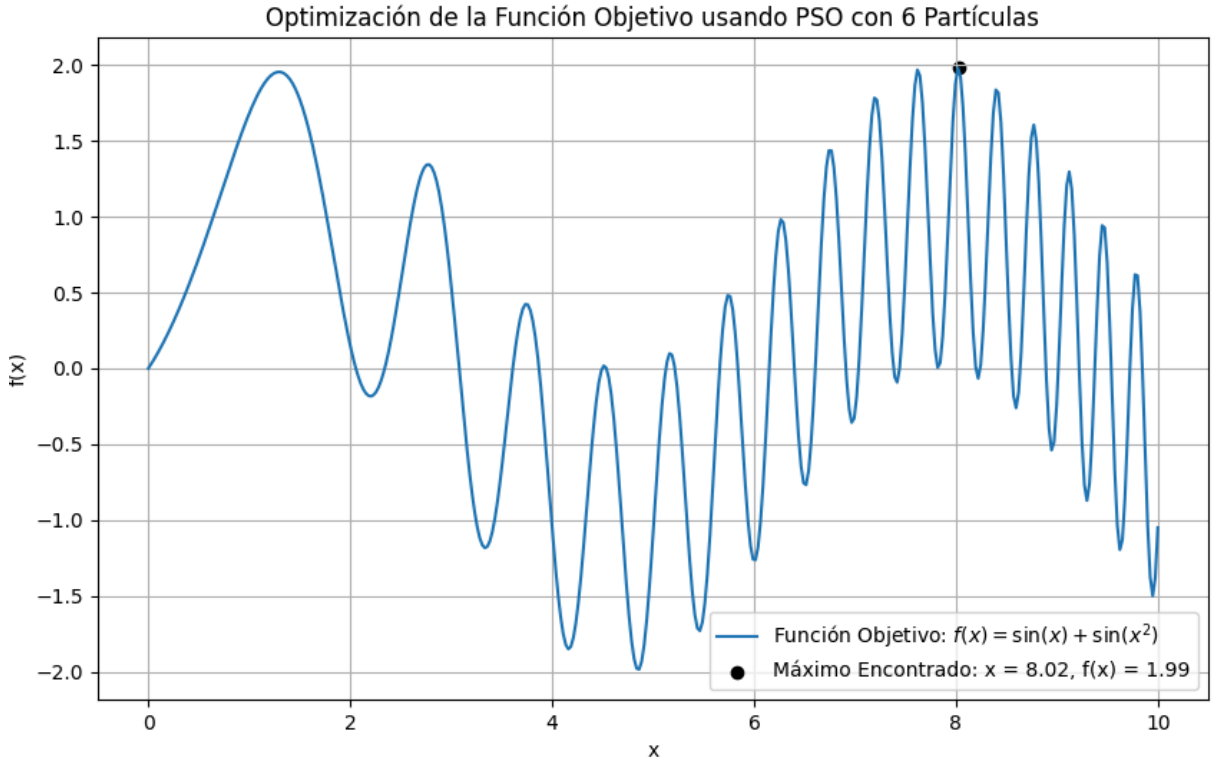
Número de partículas: 10
Solución óptima encontrada: x = 1.29
Valor objetivo óptimo: f(x) = 1.96

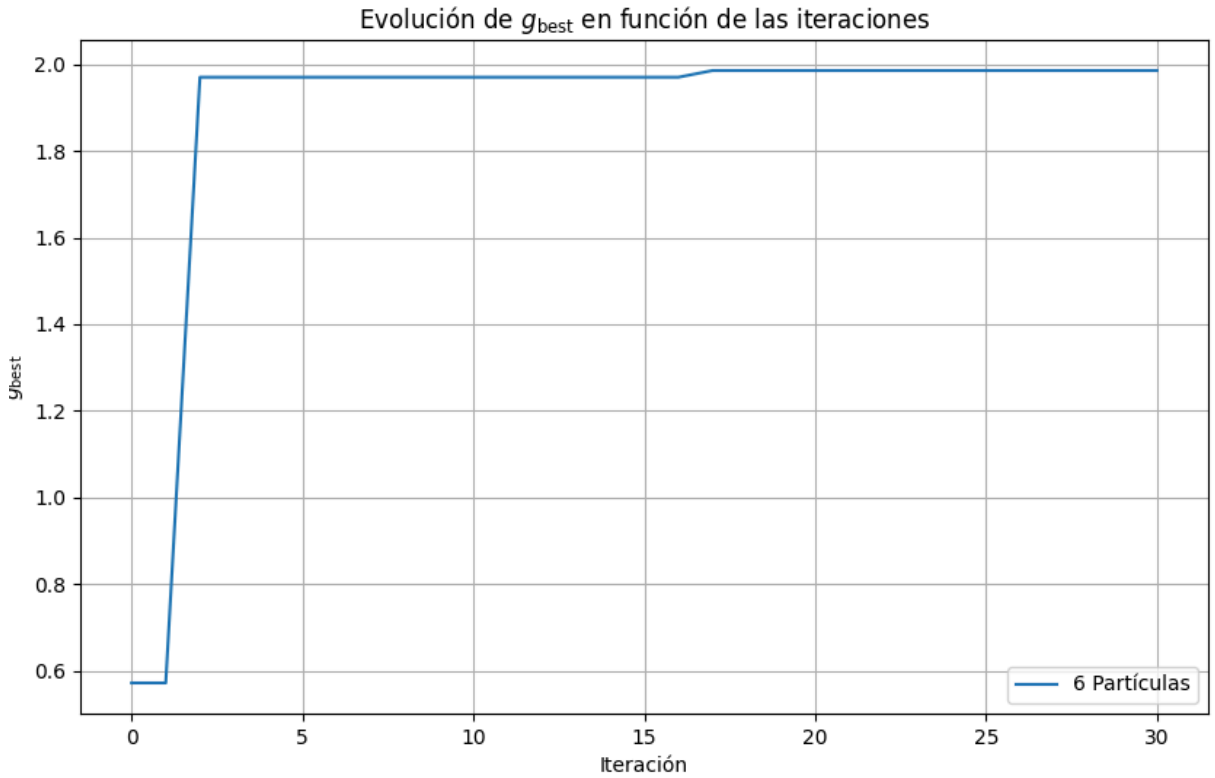


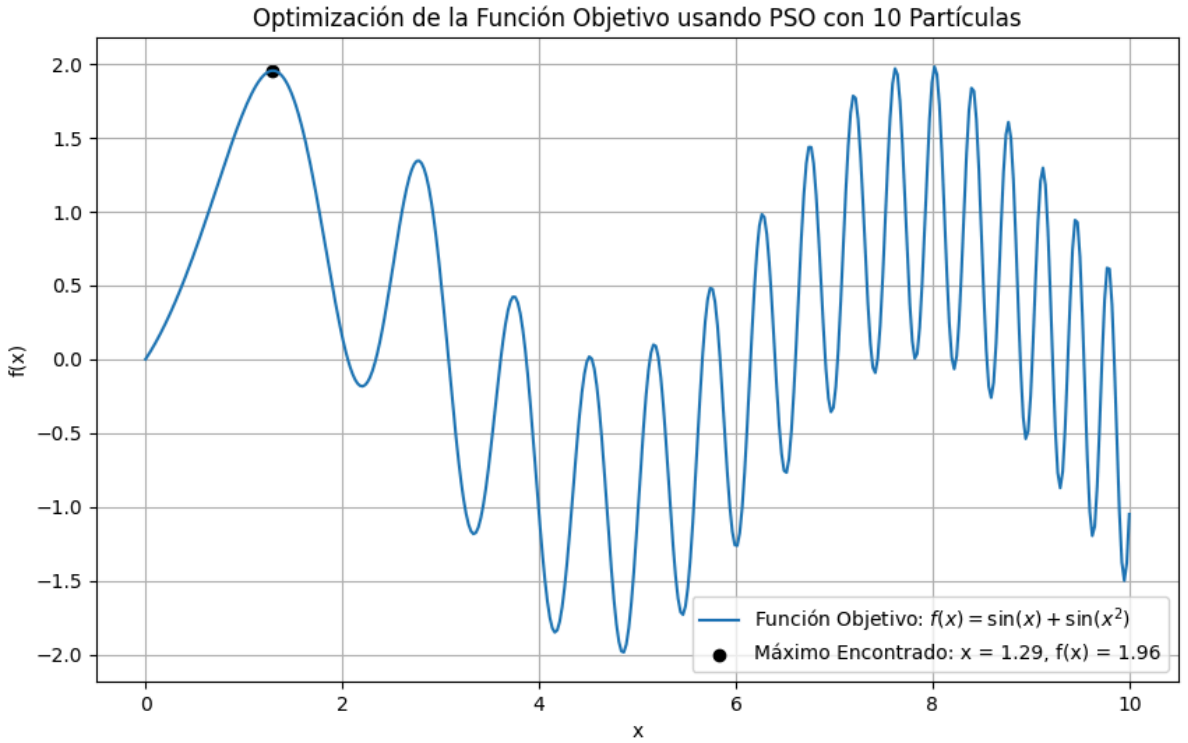


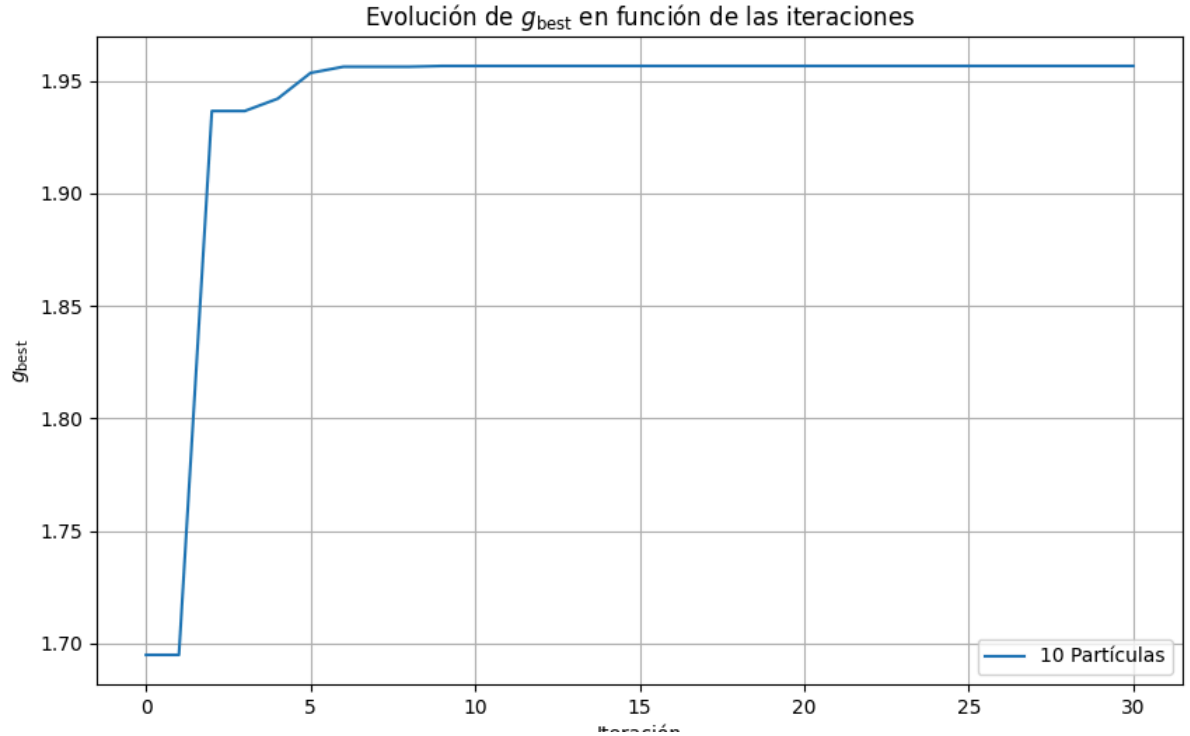

Optimización de la Función Objetivo usando PSO con 4 Partículas
f(x)
Función Objetivo: $f(x) = \sin(x) + \sin(x^2)$
Máximo Encontrado: x = 1.29, f(x) = 1.96


Evolución de g_{best} en función de las iteraciones
g_{best}
Iteración
4 Partículas


Optimización de la Función Objetivo usando PSO con 6 Partículas
f(x)
x
Función Objetivo: $f(x) = \sin(x) + \sin(x^2)$
Máximo Encontrado: x = 8.02, f(x) = 1.99


Evolución de g_{best} en función de las iteraciones
gbest
Iteración
6 Partículas



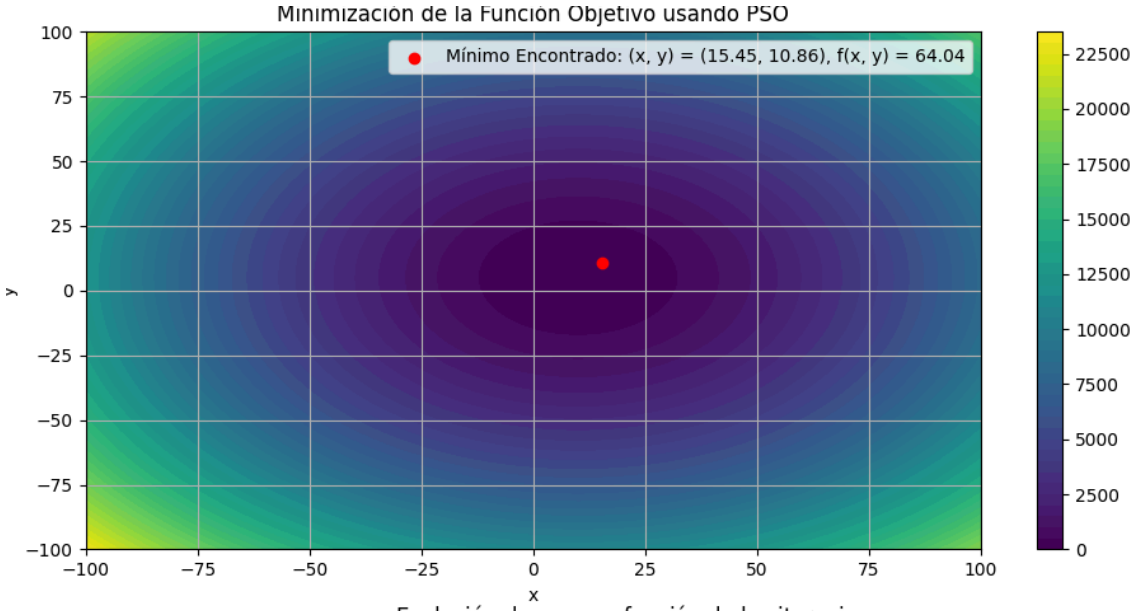


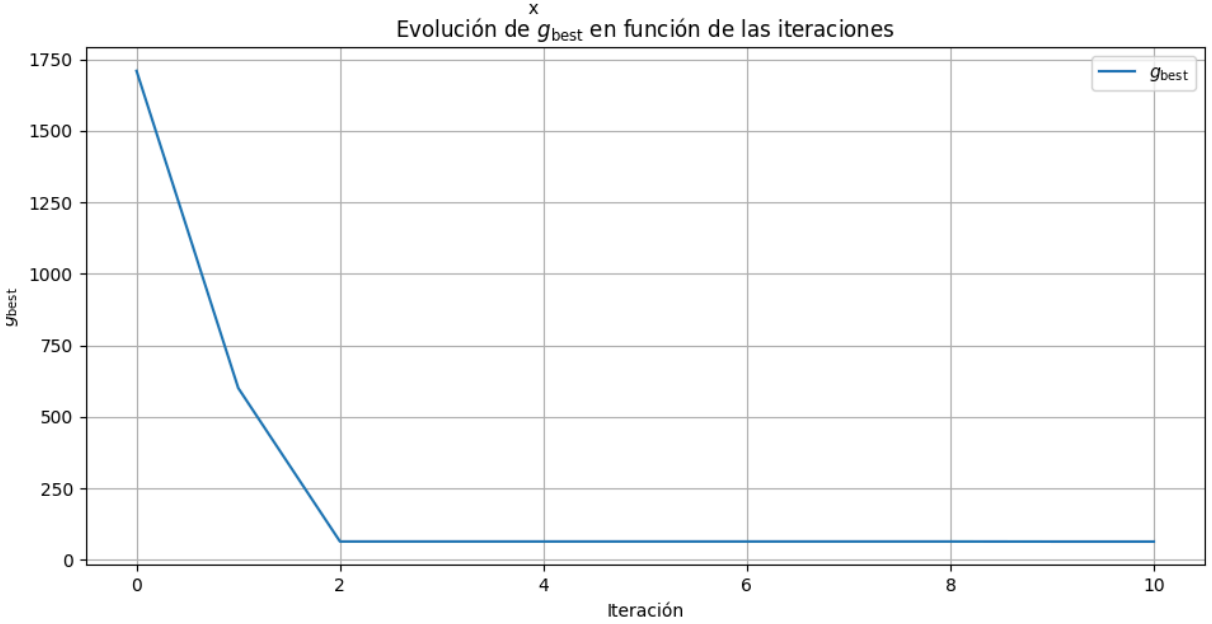
Como vemos con 6 partículas se encontró el máximo en el intervalo [0,10] en x=8.02 f(x)=1,99. En cambio en los demás casos se encontró el valor 1,96 en x=1,29.
La solución óptima y el valor objetivo se estabilizan a partir de 4 partículas, lo que sugiere que un número mayor de partículas no mejora significativamente la solución en este caso específico, además, puede no ser eficiente en tiempo de cómputo.

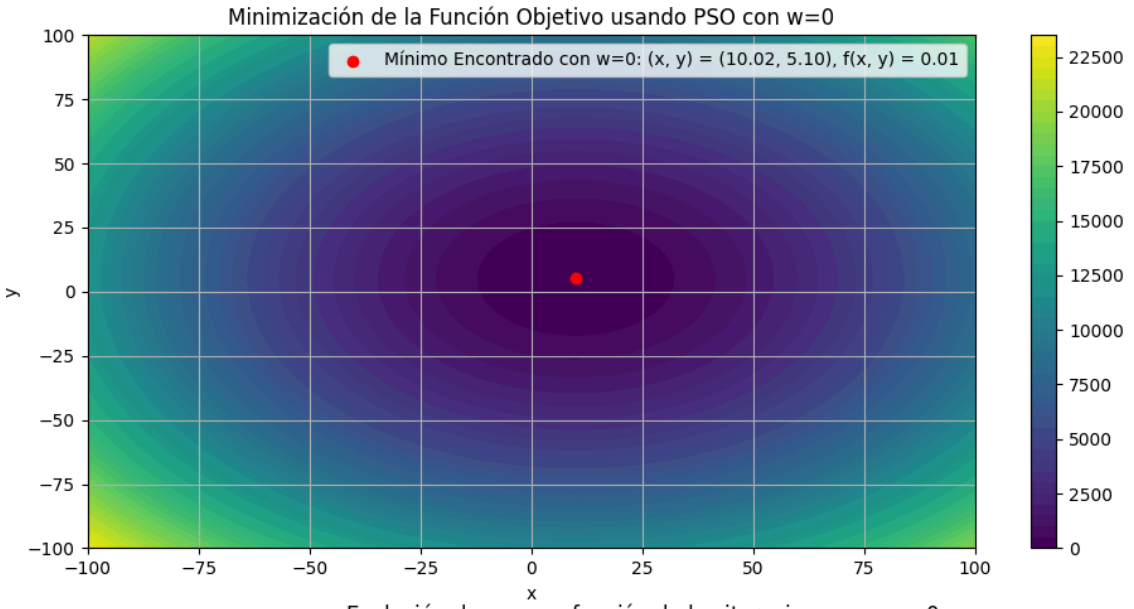
En cuanto al número de interacciones en cuestión de evolución del gbest, la mayoría llegó al máximo valor antes de la 5ta interacción, salvo con 2 partículas.

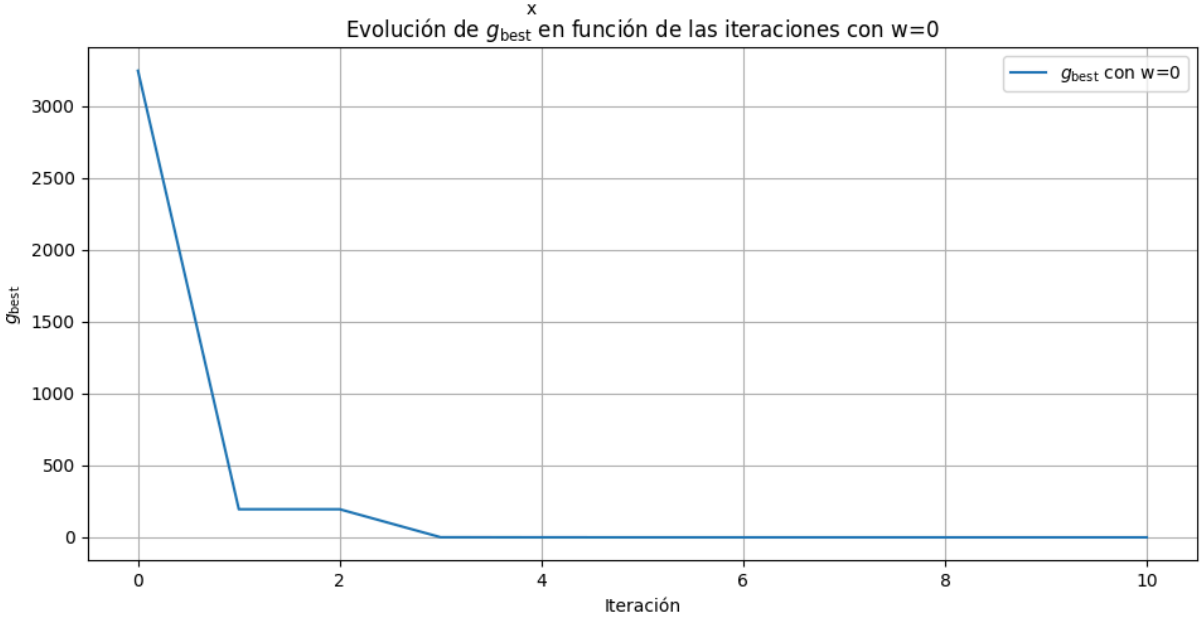
Ejercicio 3: https://github.com/freischarler/ceia_algevo/blob/main/tp2/3-main.py

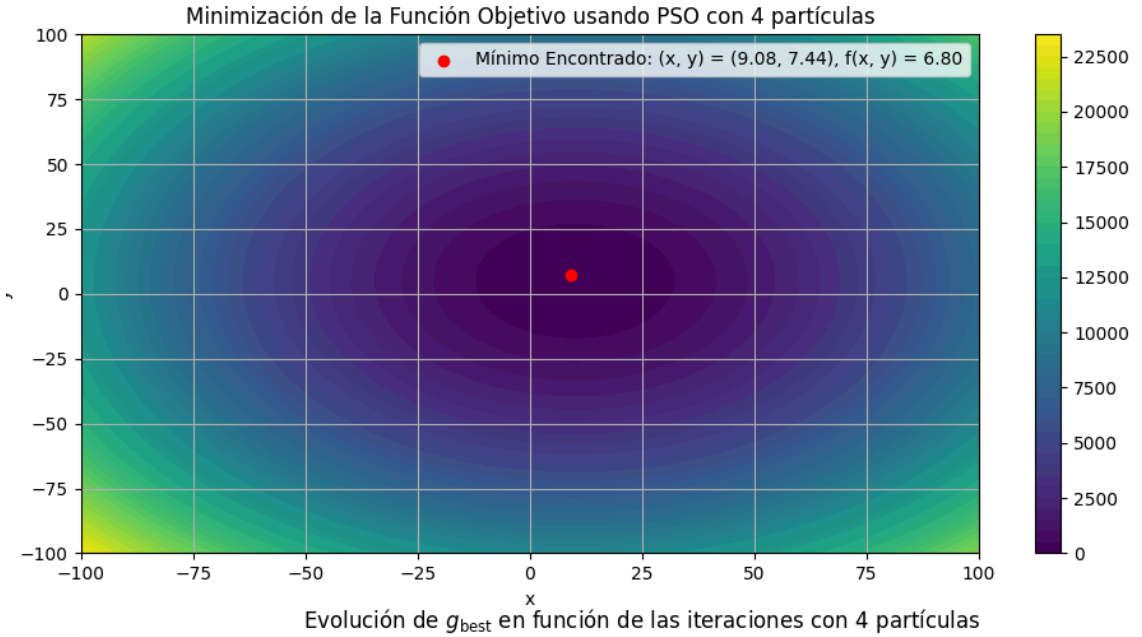
El valor de a (entre -50 y 50): 10
El valor de b (entre -50 y 50): -5
Solución óptima encontrada: (x, y) = (15.45, 10.86)
Valor objetivo óptimo: f(x, y) = 64.04
Solución óptima encontrada con w=0: (x, y) = (10.02, 5.10)
Valor objetivo óptimo con w=0: f(x, y) = 0.01
Solución óptima encontrada con 4 partículas: (x, y) = (9.08, 7.44)
Valor objetivo óptimo con 4 partículas: f(x, y) = 6.80
Solución óptima encontrada con 6 partículas: (x, y) = (7.94, -3.40)
Valor objetivo óptimo con 6 partículas: f(x, y) = 74.75
Solución óptima encontrada con 10 partículas: (x, y) = (9.75, 4.48)
Valor objetivo óptimo con 10 partículas: f(x, y) = 0.33


Minimización de la Función Objetivo usando PSO
Mínimo Encontrado: (x, y) = (15.45, 10.86), f(x, y) = 64.04
x
y

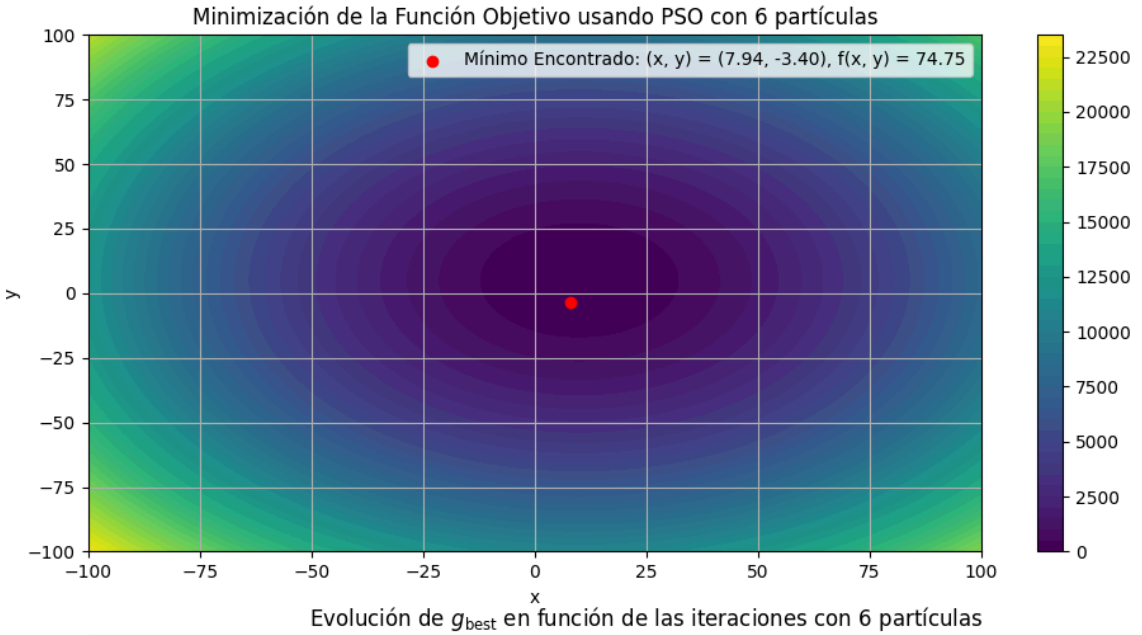

Evolución de g_{best} en función de las iteraciones
g_{best}
Iteración

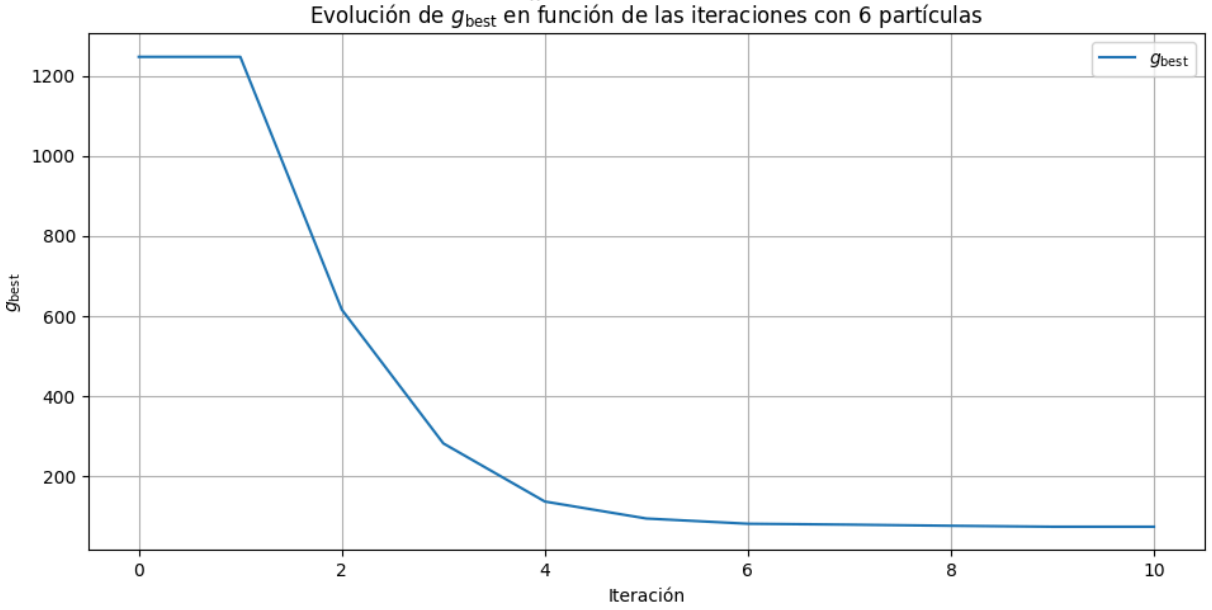

Minimización de la Función Objetivo usando PSO con w=0
Mínimo Encontrado con w=0: (x, y) = (10.02, 5.10), f(x, y) = 0.01
x
y

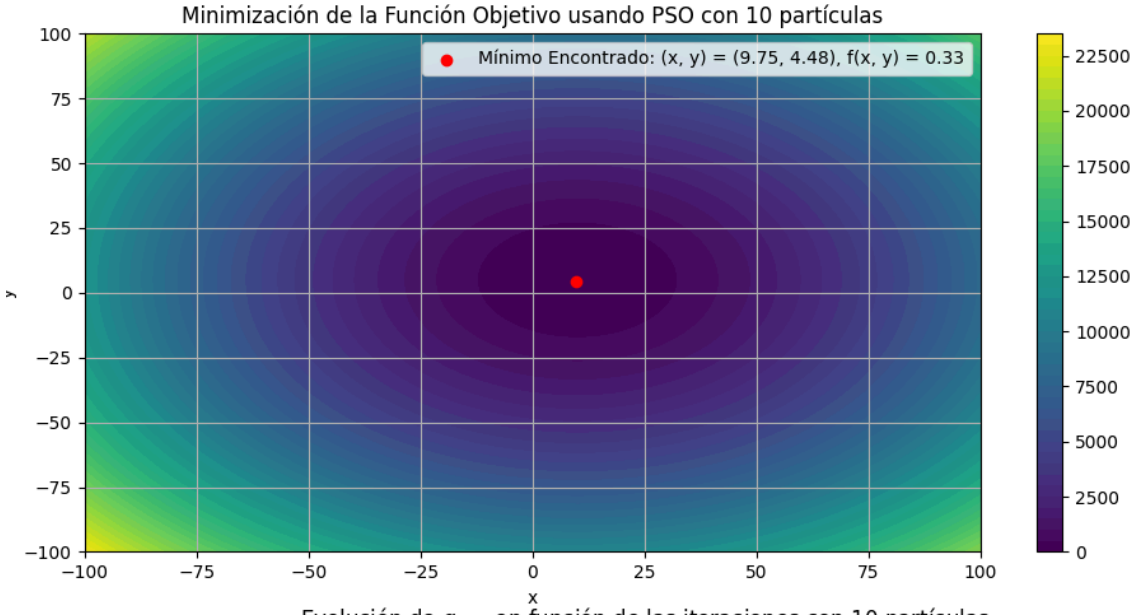

Evolución de g_{best} en función de las iteraciones con w=0
g_{best} con w=0
g_{best}
Iteración

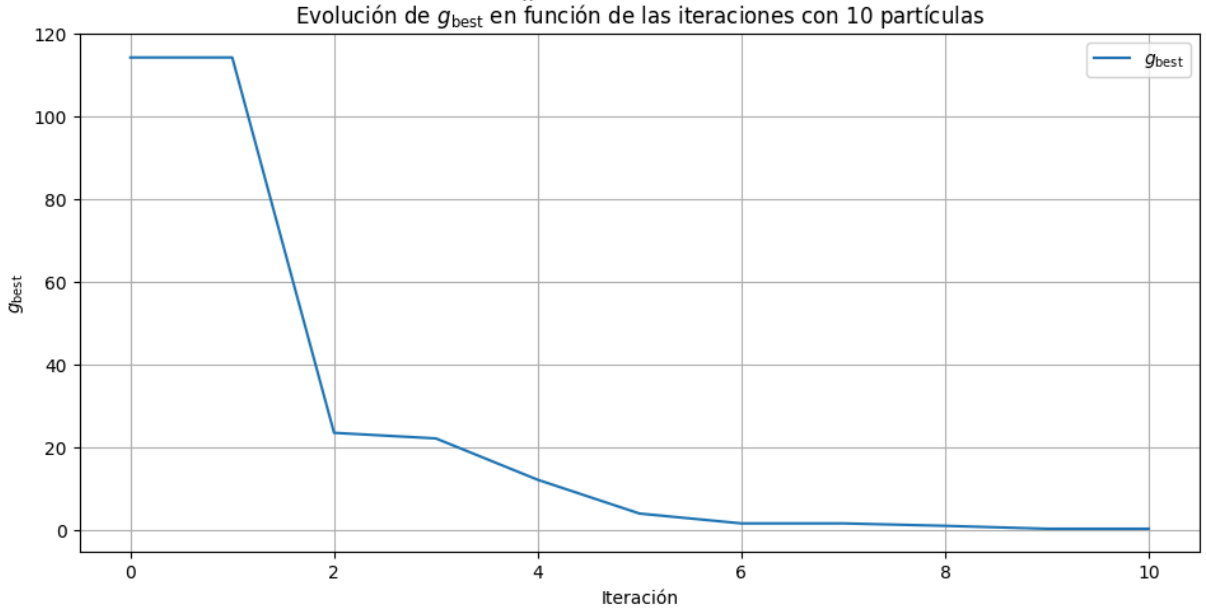

Minimización de la Función Objetivo usando PSO con 4 partículas
Mínimo Encontrado: (x, y) = (9.08, 7.44), f(x, y) = 6.80
100
75
50
25
0
−25
−50
−75
−100
−100
−75
−50
−25
0
25
50
75
100
x
y
22500
20000
17500
15000
12500
10000
7500
5000
2500
0


Evolución de g_{best} en función de las iteraciones con 4 partículas
g_{best}
140
120
100
80
60
40
20
0
0
2
4
6
8
10
Iteración
g_{best}


Minimización de la Función Objetivo usando PSO con 6 partículas
Mínimo Encontrado: (x, y) = (7.94, -3.40), f(x, y) = 74.75
x
y


Evolución de g_{best} en función de las iteraciones con 6 partículas
g_{best}
Iteración





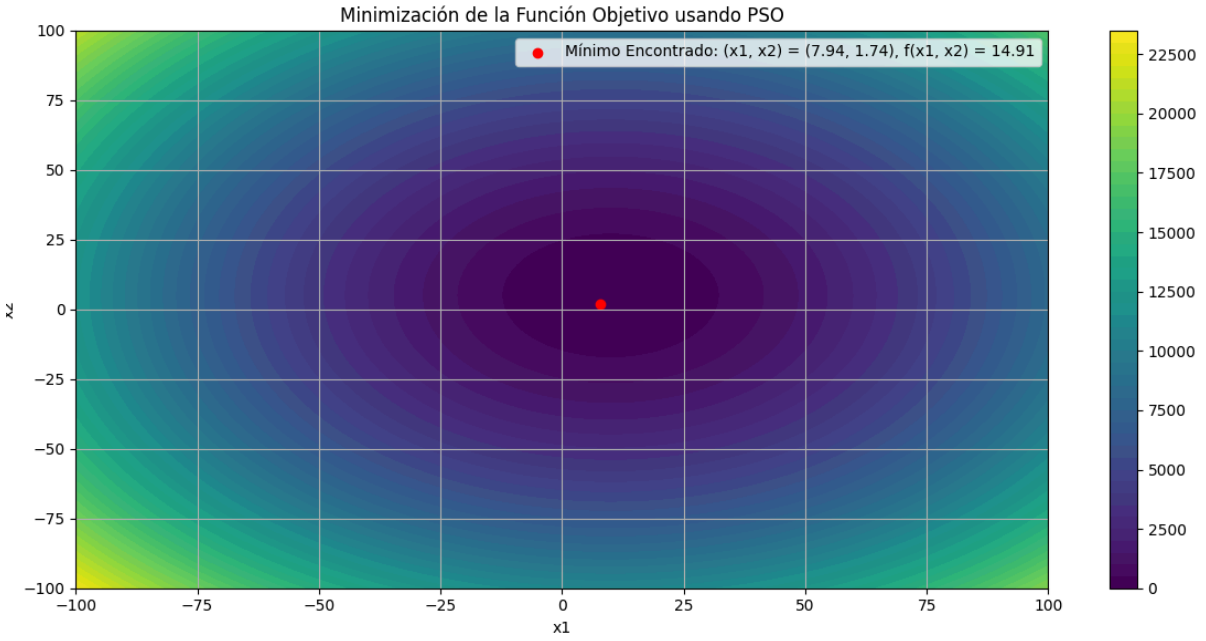
1. Solución Óptima General:
   - La solución óptima encontrada es (x, y) = (15.45, 10.86) con un valor objetivo de f(x, y) = 64.04.
   - Este resultado indica que el algoritmo ha encontrado un punto en el espacio de búsqueda que minimiza la función objetivo (x - a) ** 2 + (y + b) ** 2.
2. Solución con w=0:
   - Con w=0, la solución óptima encontrada es (x, y) = (10.02, 5.10) con un valor objetivo de f(x, y) = 0.01.
   - Este resultado es muy cercano al valor mínimo posible de 0, lo que sugiere que el algoritmo ha encontrado una solución casi perfecta.
3. Soluciones con Diferentes Cantidades de Partículas:
   - Con 4 partículas: (x, y) = (9.08, 7.44) y f(x, y) = 6.80.
   - Con 6 partículas: (x, y) = (7.94, -3.40) y f(x, y) = 74.75.
   - Con 10 partículas: (x, y) = (9.75, 4.48) y f(x, y) = 0.33.

- Eficiencia del Algoritmo:
  - La eficiencia del algoritmo varía significativamente con el número de partículas y el valor de w.
  - Con w=0, el algoritmo converge rápidamente a una solución casi óptima, lo que sugiere que la inercia no es necesaria en este caso específico.
  - Con 10 partículas, el algoritmo también encuentra una solución muy cercana al óptimo, lo que indica que un mayor número de partículas puede mejorar la exploración del espacio de búsqueda.
- Robustez del Algoritmo:
  - La robustez del algoritmo es evidente en su capacidad para encontrar soluciones razonablemente buenas con diferentes configuraciones de partículas.
  - Sin embargo, con 6 partículas, el algoritmo encontró una solución subóptima con un valor objetivo significativamente mayor, lo que sugiere que el número de partículas puede influir en la calidad de la solución.
  -

G) Utilizando Pyswarm

`Link:` https://github.com/freischarler/ceia_algevo/blob/main/tp2/3b-main.py

Stopping search: maximum iterations reached --> 10
Solución óptima encontrada: (x1, x2) = (7.94, 1.74)
Valor objetivo óptimo: f(x1, x2) = 14.91



Comparación entre modelos:

1. Consistencia de los Resultados:
   - Sin PySwarm:
     - Los resultados son menos consistentes y dependen fuertemente de los parámetros y el número de partículas.

    - Con PySwarm:
        - Los resultados son más consistentes, aunque no necesariamente los mejores en términos de valor objetivo.
2. Eficiencia Computacional:
    - Sin PySwarm:
        - La eficiencia puede variar, y se requiere ajustar manualmente los parámetros para obtener mejores resultados.
    - Con PySwarm:
        - PySwarm maneja automáticamente muchos de los parámetros, lo que puede simplificar el proceso y mejorar la eficiencia en términos de tiempo de cómputo.
3. Robustez del Algoritmo:
    - Sin PySwarm:
        - El algoritmo puede ser menos robusto y más sensible a los parámetros.
    - Con PySwarm:
        - PySwarm proporciona una implementación más robusta del PSO, lo que puede llevar a soluciones más estables

Ejercicio 4: https://github.com/freischarler/ceia_algevo/blob/main/tp2/4-main.py

(i) Los límites superior e inferior de x1 y x2 seleccionados fueron ([-5, 5]) seleccionados a prueba y error, dando un rango inicial para que el algoritmo PSO pueda buscar la solución óptima. La elección de estos límites puede depender del conocimiento previo del problema o de la necesidad de explorar un espacio de búsqueda suficientemente grande para encontrar una solución adecuada.

(ii) Se puede resolver un sistema de ecuaciones con n incógnitas realizando una suma de los cuadrados de las ecuaciones, lo que convierte el problema en una minimización de esta función objetivo:

Para el problema es: [f(x) = (3x1 + 2x2 - 9)^2 + (x1 - 5x2 - 4)^2]

El PSO busca minimizar esta función objetivo, encontrando así los valores de las incógnitas que satisfacen el sistema de ecuaciones.

(iii) Se resolvió el problema de la siguiente forma:

1. Se define la función objetivo que representa el sistema de ecuaciones.
2. Establecer los límites de búsqueda para las variables.
3. Configurar los parámetros del PSO, como el número de partículas, el número máximo de iteraciones, y los coeficientes de aceleración.
4. Ejecutar el algoritmo PSO para encontrar los valores óptimos de x1 y x2 que minimizan la función objetivo.

(iv) Los resultados obtenidos están directamente relacionados con los valores de los parámetros elegidos. El número de partículas, el número máximo de iteraciones, y los coeficientes de aceleración, influyen en la capacidad del algoritmo para explorar el espacio de búsqueda y converger a una solución óptima. Por ejemplo, aumentar el número de partículas o el número de iteraciones puede mejorar la precisión de la solución, mientras que valores inadecuados de los coeficientes de aceleración pueden llevar a una convergencia prematura o a una exploración insuficiente del espacio de búsqueda.

Partículas: 10, Iteraciones: 20, x1: 3.1424, x2: -0.1738, Valor objetivo: 6.4904e-03
Stopping search: Swarm best objective change less than 1e-08
Partículas: 10, Iteraciones: 50, x1: 10.0000, x2: -0.4138, Valor objetivo: 4.7203e+02
Stopping search: Swarm best objective change less than 1e-08
Partículas: 10, Iteraciones: 100, x1: 3.1176, x2: -0.1765, Valor objetivo: 7.0512e-09
Stopping search: maximum iterations reached --> 20
Partículas: 30, Iteraciones: 20, x1: 3.0776, x2: -0.1938, Valor objetivo: 2.6080e-02
Stopping search: Swarm best objective change less than 1e-08
Partículas: 30, Iteraciones: 50, x1: 3.1178, x2: -0.1765, Valor objetivo: 2.0816e-07
Stopping search: Swarm best objective change less than 1e-08
Partículas: 30, Iteraciones: 100, x1: 3.1175, x2: -0.1765, Valor objetivo: 2.2133e-07
Stopping search: maximum iterations reached --> 20
Partículas: 50, Iteraciones: 20, x1: 3.0984, x2: -0.1690, Valor objetivo: 5.0170e-03
Stopping search: maximum iterations reached --> 50
Partículas: 50, Iteraciones: 50, x1: 3.1176, x2: -0.1764, Valor objetivo: 2.7046e-08
Stopping search: Swarm best objective change less than 1e-08
Partículas: 50, Iteraciones: 100, x1: 3.1176, x2: -0.1765, Valor objetivo: 5.7775e-11

ANEXO ARCHIVOS DE FUENTE

Ejercicio 1

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

# Definición de la función objetivo
def objective_function(x):
    return (2 * np.sin(x)) - (x**2) / 2

# Parámetros del PSO
max_iterations = 20
c1 = 2
c2 = 2
w = 0.7

def pso(num_particles):
    # Inicialización de las partículas
    x = np.random.uniform(0, 4, num_particles)
    v = np.zeros(num_particles)
    personal_best_x = np.copy(x)
    personal_best_fitness = objective_function(x)
    global_best_x = x[np.argmax(personal_best_fitness)]
    global_best_fitness = np.max(personal_best_fitness)

    gbest_history = [global_best_fitness]

    # Bucle de optimización
    for iteration in range(max_iterations):
        for i in range(num_particles):
            # Actualización de la velocidad
            r1 = np.random.rand()
            r2 = np.random.rand()
            v[i] = (w * v[i] + c1 * r1 * (personal_best_x[i] - x[i]) +
                    c2 * r2 * (global_best_x - x[i]))

            # Actualización de la posición
            x[i] += v[i]

            # Restricción de las posiciones al intervalo [0, 4]
            x[i] = np.clip(x[i], 0, 4)

            # Evaluación de la función objetivo
            fitness = objective_function(x[i])

            # Actualización del mejor personal
            if fitness > personal_best_fitness[i]:
                personal_best_x[i] = x[i]
                personal_best_fitness[i] = fitness
```

```
        # Actualización del mejor global
        if np.max(personal_best_fitness) > global_best_fitness:
            global_best_x = personal_best_x[np.argmax(personal_best_fitness)]
            global_best_fitness = np.max(personal_best_fitness)

        gbest_history.append(global_best_fitness)

    return global_best_x, global_best_fitness, gbest_history

# Ejecutar PSO con 2 partículas y graficar la función objetivo
global_best_x, global_best_fitness, _ = pso(2)

# Números de partículas a probar
num_particles_list = [4, 10, 100, 200, 400]

# Ejecutar PSO para diferentes números de partículas y almacenar las historias de gbest
histories = {}
for num_particles in num_particles_list:
    _, _, gbest_history = pso(num_particles)
    histories[num_particles] = gbest_history

# Crear los subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto máximo encontrado
x_vals = np.linspace(0, 4, 400)
y_vals = objective_function(x_vals)

ax1.plot(x_vals, y_vals, label=r'Función Objetivo: $f(x) = 2\sin(x) - \frac{x^2}{2}$')
ax1.scatter(global_best_x, global_best_fitness, color='green', label=f'Máximo Encontrado:
x = {global_best_x:.2f}, f(x) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('f(x)')
ax1.set_title('Optimización de la Función Objetivo usando PSO')
ax1.legend()
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones
for num_particles, history in histories.items():
    ax2.plot(history, label=f'{num_particles} Partículas')

ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones para
diferentes números de partículas')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()
```

Ejercicio 2

```python
import numpy as np
import matplotlib.pyplot as plt

# Definición de la función objetivo
def objective_function(x):
    return np.sin(x) + np.sin(x ** 2)

# Parámetros del PSO
max_iterations = 30
c1 = 1.49
c2 = 1.49
w = 0.5

def pso(num_particles):
    # Inicialización de las partículas
    x = np.random.uniform(0, 10, num_particles)
    v = np.zeros(num_particles)
    personal_best_x = np.copy(x)
    personal_best_fitness = objective_function(x)
    global_best_x = x[np.argmax(personal_best_fitness)]
    global_best_fitness = np.max(personal_best_fitness)

    gbest_history = [global_best_fitness]

    # Bucle de optimización
    for iteration in range(max_iterations):
        for i in range(num_particles):
            # Actualización de la velocidad
            r1 = np.random.rand()
            r2 = np.random.rand()
            v[i] = (w * v[i] + c1 * r1 * (personal_best_x[i] - x[i]) +
                    c2 * r2 * (global_best_x - x[i]))

            # Actualización de la posición
            x[i] += v[i]

            # Restricción de las posiciones al intervalo [0, 10]
            x[i] = np.clip(x[i], 0, 10)

            # Evaluación de la función objetivo
            fitness = objective_function(x[i])

            # Actualización del mejor personal
            if fitness > personal_best_fitness[i]:
                personal_best_x[i] = x[i]
                personal_best_fitness[i] = fitness

        # Actualización del mejor global
        if np.max(personal_best_fitness) > global_best_fitness:
            global_best_x = personal_best_x[np.argmax(personal_best_fitness)]
```

```
            global_best_fitness = np.max(personal_best_fitness)

        gbest_history.append(global_best_fitness)

    return global_best_x, global_best_fitness, gbest_history

# Función para graficar los resultados
def plot_results(num_particles, global_best_x, global_best_fitness, gbest_history):
    # Graficar la función objetivo
    x_vals = np.linspace(0, 10, 400)
    y_vals = objective_function(x_vals)

    plt.figure(figsize=(10, 6))
    plt.plot(x_vals, y_vals, label=r'Función Objetivo: $f(x) = \sin(x) + \sin(x^2)$')
    plt.scatter(global_best_x, global_best_fitness, color='black', label=f'Máximo
Encontrado: x = {global_best_x:.2f}, f(x) = {global_best_fitness:.2f}')
    plt.xlabel('x')
    plt.ylabel('f(x)')
    plt.title(f'Optimización de la Función Objetivo usando PSO con {num_particles}
Partículas')
    plt.legend()
    plt.grid(True)
    plt.show()

    # Graficar gbest en función de las iteraciones
    plt.figure(figsize=(10, 6))
    plt.plot(gbest_history, label=f'{num_particles} Partículas')
    plt.xlabel('Iteración')
    plt.ylabel(r'$g_{\text{best}}$')
    plt.title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones')
    plt.legend()
    plt.grid(True)
    plt.show()

# Ejecutar PSO con diferentes números de partículas y mostrar resultados
particle_counts = [2, 4, 6, 10]

for num_particles in particle_counts:
    global_best_x, global_best_fitness, gbest_history = pso(num_particles)
    print(f'Número de partículas: {num_particles}')
    print(f'Solución óptima encontrada: x = {global_best_x:.2f}')
    print(f'Valor objetivo óptimo: f(x) = {global_best_fitness:.2f}')
    plot_results(num_particles, global_best_x, global_best_fitness, gbest_history)
```

Ejercicio 3

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

# Definición de la función objetivo
def objective_function(x, y, a, b):
```

```
    return (x - a) ** 2 + (y + b) ** 2

# Parámetros del PSO
num_particles = 20
max_iterations = 10
c1 = 2
c2 = 2
w = 0.7

def pso(a, b, w):
    # Inicialización de las partículas
    x = np.random.uniform(-100, 100, num_particles)
    y = np.random.uniform(-100, 100, num_particles)
    v_x = np.zeros(num_particles)
    v_y = np.zeros(num_particles)
    personal_best_x = np.copy(x)
    personal_best_y = np.copy(y)
    personal_best_fitness = objective_function(x, y, a, b)
    global_best_index = np.argmin(personal_best_fitness)
    global_best_x = personal_best_x[global_best_index]
    global_best_y = personal_best_y[global_best_index]
    global_best_fitness = personal_best_fitness[global_best_index]

    gbest_history = [global_best_fitness]

    # Bucle de optimización
    for iteration in range(max_iterations):
        for i in range(num_particles):
            # Actualización de la velocidad
            r1 = np.random.rand()
            r2 = np.random.rand()
            v_x[i] = (w * v_x[i] + c1 * r1 * (personal_best_x[i] - x[i]) +
                      c2 * r2 * (global_best_x - x[i]))
            v_y[i] = (w * v_y[i] + c1 * r1 * (personal_best_y[i] - y[i]) +
                      c2 * r2 * (global_best_y - y[i]))

            # Actualización de la posición
            x[i] += v_x[i]
            y[i] += v_y[i]

            # Restricción de las posiciones al intervalo [-100, 100]
            x[i] = np.clip(x[i], -100, 100)
            y[i] = np.clip(y[i], -100, 100)

            # Evaluación de la función objetivo
            fitness = objective_function(x[i], y[i], a, b)

            # Actualización del mejor personal
            if fitness < personal_best_fitness[i]:
                personal_best_x[i] = x[i]
                personal_best_y[i] = y[i]
                personal_best_fitness[i] = fitness
```

```
        # Actualización del mejor global
        if np.min(personal_best_fitness) < global_best_fitness:
            global_best_index = np.argmin(personal_best_fitness)
            global_best_x = personal_best_x[global_best_index]
            global_best_y = personal_best_y[global_best_index]
            global_best_fitness = personal_best_fitness[global_best_index]

        gbest_history.append(global_best_fitness)

    return global_best_x, global_best_y, global_best_fitness, gbest_history

# Entrada de valores para a y b
# Función para solicitar un valor dentro del rango
def solicitar_valor(mensaje):
    while True:
        try:
            valor = float(input(mensaje))
            if -50 <= valor <= 50:
                return valor
            else:
                print("El valor debe estar entre -50 y 50. Intente de nuevo.")
        except ValueError:
            print("Entrada inválida. Por favor, ingrese un número.")

# Entrada de valores para a y b
a = solicitar_valor("Ingrese el valor de a (entre -50 y 50): ")
b = solicitar_valor("Ingrese el valor de b (entre -50 y 50): ")

# Ejecutar PSO con los parámetros dados
global_best_x, global_best_y, global_best_fitness, gbest_history = pso(a, b, w)

# Graficar la función objetivo y el punto mínimo encontrado
x_vals = np.linspace(-100, 100, 400)
y_vals = np.linspace(-100, 100, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = objective_function(X, Y, a, b)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto mínimo encontrado
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(global_best_x, global_best_y, color='red', label=f'Mínimo Encontrado: (x, y) =
({global_best_x:.2f}, {global_best_y:.2f}), f(x, y) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Minimización de la Función Objetivo usando PSO')
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones
```

```
ax2.plot(gbest_history, label=r'$g_{\text{best}}$')
ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# Mostrar la solución óptima encontrada
print(f'Solución óptima encontrada: (x, y) = ({global_best_x:.2f}, {global_best_y:.2f})')
print(f'Valor objetivo óptimo: f(x, y) = {global_best_fitness:.2f}')

# Establecer el coeficiente de inercia w en 0 y ejecutar el algoritmo nuevamente
w = 0
global_best_x, global_best_y, global_best_fitness, gbest_history = pso(a, b, w)

# Graficar la función objetivo y el punto mínimo encontrado con w = 0
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto mínimo encontrado con w = 0
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(global_best_x, global_best_y, color='red', label=f'Mínimo Encontrado con w=0:
(x, y) = ({global_best_x:.2f}, {global_best_y:.2f}), f(x, y) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Minimización de la Función Objetivo usando PSO con w=0')
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones con w = 0
ax2.plot(gbest_history, label=r'$g_{\text{best}}$ con w=0')
ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones con w=0')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

# Mostrar la solución óptima encontrada con w = 0
print(f'Solución óptima encontrada con w=0: (x, y) = ({global_best_x:.2f},
{global_best_y:.2f})')
print(f'Valor objetivo óptimo con w=0: f(x, y) = {global_best_fitness:.2f}')

num_particles = 4

# Ejecutar PSO con los parámetros dados
global_best_x, global_best_y, global_best_fitness, gbest_history = pso(a, b, w)
```

```
# Mostrar la solución óptima encontrada
print(f'Solución óptima encontrada con 4 partículas: (x, y) = ({global_best_x:.2f},
{global_best_y:.2f})')
print(f'Valor objetivo óptimo con 4 partículas: f(x, y) = {global_best_fitness:.2f}')

# Graficar la función objetivo y el punto mínimo encontrado
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto mínimo encontrado
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(global_best_x, global_best_y, color='red', label=f'Mínimo Encontrado: (x, y) =
({global_best_x:.2f}, {global_best_y:.2f}), f(x, y) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Minimización de la Función Objetivo usando PSO con 4 partículas')
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones
ax2.plot(gbest_history, label=r'$g_{\text{best}}$')
ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones con 4
partículas')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

num_particles = 6

# Ejecutar PSO con los parámetros dados
global_best_x, global_best_y, global_best_fitness, gbest_history = pso(a, b, w)

# Mostrar la solución óptima encontrada
print(f'Solución óptima encontrada con 6 partículas: (x, y) = ({global_best_x:.2f},
{global_best_y:.2f})')
print(f'Valor objetivo óptimo con 6 partículas: f(x, y) = {global_best_fitness:.2f}')

# Graficar la función objetivo y el punto mínimo encontrado
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto mínimo encontrado
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(global_best_x, global_best_y, color='red', label=f'Mínimo Encontrado: (x, y) =
({global_best_x:.2f}, {global_best_y:.2f}), f(x, y) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Minimización de la Función Objetivo usando PSO con 6 partículas')
```

```
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones
ax2.plot(gbest_history, label=r'$g_{\text{best}}$')
ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones con 6
partículas')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()

num_particles = 10

# Ejecutar PSO con los parámetros dados
global_best_x, global_best_y, global_best_fitness, gbest_history = pso(a, b, w)

# Mostrar la solución óptima encontrada
print(f'Solución óptima encontrada con 10 partículas: (x, y) = ({global_best_x:.2f},
{global_best_y:.2f})')
print(f'Valor objetivo óptimo con 10 partículas: f(x, y) = {global_best_fitness:.2f}')

# Graficar la función objetivo y el punto mínimo encontrado
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))

# Primer subplot: Función objetivo y el punto mínimo encontrado
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(global_best_x, global_best_y, color='red', label=f'Mínimo Encontrado: (x, y) =
({global_best_x:.2f}, {global_best_y:.2f}), f(x, y) = {global_best_fitness:.2f}')
ax1.set_xlabel('x')
ax1.set_ylabel('y')
ax1.set_title('Minimización de la Función Objetivo usando PSO con 10 partículas')
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

# Segundo subplot: Evolución de g_best en función de las iteraciones
ax2.plot(gbest_history, label=r'$g_{\text{best}}$')
ax2.set_xlabel('Iteración')
ax2.set_ylabel(r'$g_{\text{best}}$')
ax2.set_title(r'Evolución de $g_{\text{best}}$ en función de las iteraciones con 10
partículas')
ax2.legend()
ax2.grid(True)

plt.tight_layout()
plt.show()
```

Ejercicio 3B

```python
from pyswarm import pso
import numpy as np
import matplotlib.pyplot as plt

# Definición de la función objetivo
def objective_function(x):
    x1, x2 = x
    return (x1 - a) ** 2 + (x2 + b) ** 2

# Entrada de valores para a y b
def solicitar_valor(mensaje):
    while True:
        try:
            valor = float(input(mensaje))
            if -50 <= valor <= 50:
                return valor
            else:
                print("El valor debe estar entre -50 y 50. Intente de nuevo.")
        except ValueError:
            print("Entrada inválida. Por favor, ingrese un número.")

a = solicitar_valor("Ingrese el valor de a (entre -50 y 50): ")
b = solicitar_valor("Ingrese el valor de b (entre -50 y 50): ")

# Definición de los límites para las variables
lb = [-100, -100]  # límites inferiores para x1 y x2
ub = [100, 100]    # límites superiores para x1 y x2

# Parámetros del PSO
num_particles = 20
max_iterations = 10
c1 = 2
c2 = 2
w = 0.7

# Ejecutar PSO usando pyswarm
xopt, fopt = pso(objective_function, lb, ub, swarmsize=num_particles,
maxiter=max_iterations, debug=False)

# Mostrar la solución óptima encontrada
print(f'Solución óptima encontrada: (x1, x2) = ({xopt[0]:.2f}, {xopt[1]:.2f})')
print(f'Valor objetivo óptimo: f(x1, x2) = {fopt:.2f}')

# Graficar la función objetivo y el punto mínimo encontrado
x_vals = np.linspace(-100, 100, 400)
y_vals = np.linspace(-100, 100, 400)
X, Y = np.meshgrid(x_vals, y_vals)
Z = objective_function([X, Y])
```

```
fig, ax1 = plt.subplots(figsize=(12, 6))

# Función objetivo y el punto mínimo encontrado
contour = ax1.contourf(X, Y, Z, levels=50, cmap='viridis')
ax1.scatter(xopt[0], xopt[1], color='red', label=f'Mínimo Encontrado: (x1, x2) =
({xopt[0]:.2f}, {xopt[1]:.2f}), f(x1, x2) = {fopt:.2f}')
ax1.set_xlabel('x1')
ax1.set_ylabel('x2')
ax1.set_title('Minimización de la Función Objetivo usando PSO')
ax1.legend()
fig.colorbar(contour, ax=ax1)
ax1.grid(True)

plt.tight_layout()
plt.show()
```

Ejercicio 4

```
import numpy as np
from pyswarm import pso

# Definición de la función objetivo
def objective_function(x):
    x1, x2 = x
    eq1 = 3 * x1 + 2 * x2 - 9
    eq2 = x1 - 5 * x2 - 4
    return eq1**2 + eq2**2

# Definir los límites de búsqueda para x1 y x2
lb = [-5, -5]  # Límite inferior para x1 y x2
ub = [5, 5]    # Límite superior para x1 y x2


# Parámetros del PSO
c1 = 2.0          # Coeficiente de aceleración personal
c2 = 2.0          # Coeficiente de aceleración global
w = 0.5           # Peso de inercia
max_iter = 50     # Número máximo de iteraciones
num_particles = 30 # Número de partículas

# Ejecución del PSO
xopt, fopt = pso(objective_function, lb, ub, swarmsize=num_particles, maxiter=max_iter,
omega=w, phip=c1, phig=c2)

# Mostrar los resultados
print(f'Solución óptima encontrada: x1 = {xopt[0]:.4f}, x2 = {xopt[1]:.4f}')
print(f'Valor objetivo óptimo: {fopt:.4e}')
```