CARRERA DE ESPECIALIZACIÓN EN INTELIGENCIA ARTIFICIAL

Algoritmos Evolutivos

16Co2024

Trabajo Práctico N°2

Tema: *PSO*

- DEGANO, MYRNA LORENA N° SIU a1618 myrna.l.degano@gmail.com
- RIVAS, GUSTAVO JULIÁN N° SIU a1620 gus.j.rivas@gmail.com

11 de agosto de 2024

Índice

Ejercicio 1 2
Ejercicio 2 14
Ejercicio 3 32
Ejercicio 4 48
Anexo – Código fuente 55
Bibliografía 55

Ejercicio 1

Escribir un algoritmo PSO para la maximización de la función:

$$f(x) = 2\sin(x) - \frac{x^2}{2}$$

En el intervalo de $0 \leq x \leq 4$ y que cumpla con las siguientes consignas:

A. Transcribir el algoritmo en Python con los siguientes parámetros: número de partículas = 2, máximo número de iteraciones 80, coeficientes de aceleración c1 = c2 = 2, peso de inercia w = 0.7.

```
##########################################################################
# CEIA - 16Co2024 - Algoritmos Evolutivos - TP2 - Ejercicio 1
# Gustavo J. Rivas (a1620) | Myrna L. Degano (a1618)
##########################################################################
# Algoritmo PSO para maximizar función.
##########################################################################
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate


# Obtener parámetros de ejecución
def get_params():


    num_p = input("Número de partículas (DEFAULT: 2): ").strip()
    num_p = int(num_p) if num_p else 2


    num_i = input("Número de iteraciones (DEFAULT: 80): ").strip()
    num_i = int(num_i) if num_i else 80


    c1 = input("Coeficiente de aceleración - Componente cognitivo (DEFAULT:
2): ").strip()
```

```
    c1 = float(c1) if c1 else 2.0


    c2 = input("Coeficiente de aceleración  - Componente social (DEFAULT: 2):
").strip()
    c2 = float(c2) if c2 else 2.0


    w = input("Coeficiente de inercia (DEFAULT: 0.7): ").strip()
    w = float(w) if w else 0.7


    # Límites inferior y suprior de x
    l = 0
    u = 4


    return num_p, num_i, c1, c2, w, l, u



# Función objetivo
def f_obj(x):
    return 2*np.sin(x) - (x**2)/2



######################################################################
# Desarrollo del algoritmo
######################################################################
def custom_pso(particles, iterations, c1, c2, w, lb, ub):
    result = [["Iteración #", "GBest - Valor de x", "GBest - Valor de f(x)"]]


    # Inicializar enjambre de partículas (posiciones y velocidades)
    swarm = np.random.uniform(lb, ub, particles)
    velocity = np.zeros(particles)


    # Personal Best inicial para cada partícula
```

```
    pbest = swarm.copy()
    f_pbest = [f_obj(swarm[i]) for i in range(particles)]

    # Global Best inicial
    gbest = pbest[np.argmax(f_pbest)]
    f_gbest = np.max(f_pbest)

    # Búsqueda del óptimo
    for i in range(iterations): # Máximo de iteraciones

        for p in range(particles):  # Iteración por partícula

            r1, r2 = np.random.rand(), np.random.rand()  # Aleatorios por cada partícula/iteración

            # Actualizar velocidad y posición de la partícula dentro de los límites del espacio de búsqueda
            velocity[p] = (w * velocity[p] + c1 * r1 * (pbest[p] - swarm[p]) + c2 * r2 * (gbest - swarm[p]))
            swarm[p] = np.clip(swarm[p] + velocity[p], lb, ub)

            # Evaluar la función objetivo para la nueva posición de la partícula
            fitness = f_obj(swarm[p])

            # Actualizar el mejor personal
            if fitness > f_pbest[p]:
                f_pbest[p] = fitness
                pbest[p] = swarm[p].copy()

                # Actualizar del mejor global
                if fitness > f_gbest:
                    f_gbest = fitness
```

```
                    gbest = swarm[p].copy()


        # Resultados de la iteración
        result.append([i+1, gbest, f_gbest])


    return result, f_gbest, gbest


########################################################################


# Obtener parámetros de ejecución
# Cantidad de partículas, máximo de iteraciones
# Coeficientes de aceleración e incercia (c1, c2, w)
print("\nINGRESE LOS PARÁMETROS PARA LA EJECUCIÓN DEL ALGORITMO (O <ENTER>
PARA TOMAR LOS VALORES POR DEFAULT)\n")
particles, iterations, c1, c2, w, lb, ub = get_params()
result, f_gbest, gbest = custom_pso (particles, iterations, c1, c2, w, lb, ub)


# Impresión de resultados
print(tabulate(result, headers="firstrow", tablefmt="grid"))
print(f"\nEl valor óptimo es: {f_gbest} para x= {gbest}")


# Gráfico de la función objetivo
x = np.linspace(lb, ub, 400)
y = 2*np.sin(x) - (x**2)/2


plt.figure(figsize=(10, 5))
plt.plot(x, y, label='y = 2*sin(x) - x^2 / 2', color='blue')


# Añadir el punto verde en el valor máximo
plt.scatter(gbest, f_gbest, color='green', zorder=5, label='Valor máximo')
plt.annotate(f'GBest\nx={gbest}\nf(x)={f_gbest}', (gbest, f_gbest),
textcoords="offset points", xytext=(0,-40), ha='center', zorder=10)
```

```
plt.title('Gráfica de la función objetivo')
plt.xlabel('x')
plt.ylabel('y')
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()

# Gráfico de GBest en función de las iteraciones
x = np.arange(1, iterations + 1)
y = np.array(result[1:])[:, 2]

plt.figure(figsize=(10, 5))
plt.plot(x, y, label='GBest en cada iteración', color='blue', linewidth=2)
plt.title("Gráfica de Global Best en cada Iteración")
plt.xlabel('Iteración #')
plt.ylabel('Global Best')
plt.scatter(iterations, f_gbest,
            color='green', zorder=5,
            label=f'Valor máximo luego de {iterations} iteraciones =
{f_gbest}')

step = iterations // 10 if iterations > 30 else 1
for i in range(0, len(x), step):
    plt.annotate(round(y[i], 2),
                 (x[i], y[i]),
                 fontsize=8,
                 textcoords="offset points",
                 xytext=(0, -15),
                 ha='right',
                 arrowprops=dict(facecolor='blue', shrink=0, width=0.5,
headwidth=5, headlength=5))
```

```
plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()

# Gráfico de comparación de ejecuciones
num_part = [4, 10, 100, 200 , 400]
# mismos parámetros ingresados - iterations, c1, c2, w, lb, ub

comparative = []
for k in num_part:
    result, f_gbest, gbest = custom_pso (k, iterations, c1, c2, w, lb, ub)
    comparative.append ([k, result, f_gbest, gbest])

# Definir el rango de valores para x
x = np.arange(1, iterations + 1)

y = []
y_min = 100
y_max = 0
for j in comparative:
    jbest = np.array ([sublist[-1] for sublist in j[1][1:]])
    y.append([j[0], jbest, j[2]])
    y_min = np.min(jbest) if np.min(jbest) < y_min else y_min
    y_max = np.max(jbest) if np.max(jbest) > y_max else y_max

plt.figure(figsize=(10, 5))

plt.plot(x, y[0][1], label=f'{y[0][0]} partículas - GBest={y[0][2]}',
color='blue')
```

```
plt.plot(x, y[1][1], label=f'{y[1][0]} partículas - GBest={y[1][2]}',
color='red')

plt.plot(x, y[2][1], label=f'{y[2][0]} partículas - GBest={y[2][2]}',
color='green')

plt.plot(x, y[3][1], label=f'{y[3][0]} partículas - GBest={y[3][2]}',
color='purple')

plt.plot(x, y[4][1], label=f'{y[4][0]} partículas - GBest={y[4][2]}',
color='orange')


plt.title('Comparativa con diferente tamaño de enjambre')

plt.xlabel('x')

plt.ylabel('y')

plt.grid(True, linestyle='--', alpha=0.7)


print (f"{y_min} {y_max}")

plt.yticks(np.arange(y_min, y_max, step=(y_max-y_min)/10))


plt.legend()

plt.show()
```

B. Transcribir la solución óptima encontrada (dominio) y el valor objetivo óptimo (imagen).

```
Número de iteraciones: 80
Coeficiente de aceleración - Componente cognitivo: 2
Coeficiente de aceleración - Componente social: 2
Coeficiente de inercia: 0.7

+---------------+----------------------+-------------------------+
|   Iteración # |   GBest - Valor de x |   GBest - Valor de f(x) |
+===============+======================+=========================+
|             1 |              1.34554 |                 1.04423 |
+---------------+----------------------+-------------------------+
|             2 |              1.27316 |                 1.10159 |
+---------------+----------------------+-------------------------+
|             3 |              1.2104  |                 1.13898 |
+---------------+----------------------+-------------------------+
|             4 |              1.16646 |                 1.15841 |
+---------------+----------------------+-------------------------+
|             5 |              1.09248 |                 1.17879 |
+---------------+----------------------+-------------------------+
|             6 |              1.03525 |                 1.18411 |
+---------------+----------------------+-------------------------+
```

7	1.03525	1.18411
8	1.03525	1.18411
9	1.02678	1.18414
10	1.02678	1.18414
11	1.02678	1.18414
12	1.02678	1.18414
13	1.02678	1.18414
14	1.02678	1.18414
15	1.0325	1.18414
16	1.0325	1.18414
17	1.0325	1.18414
18	1.02781	1.18414
19	1.02781	1.18414
20	1.03064	1.18415
21	1.03064	1.18415
22	1.02968	1.18415
23	1.02991	1.18415
24	1.02991	1.18415
25	1.02991	1.18415
26	1.02991	1.18415
27	1.02991	1.18415
28	1.02991	1.18415
29	1.02991	1.18415
30	1.02986	1.18415
31	1.02986	1.18415
32	1.02986	1.18415
33	1.02986	1.18415
34	1.02986	1.18415

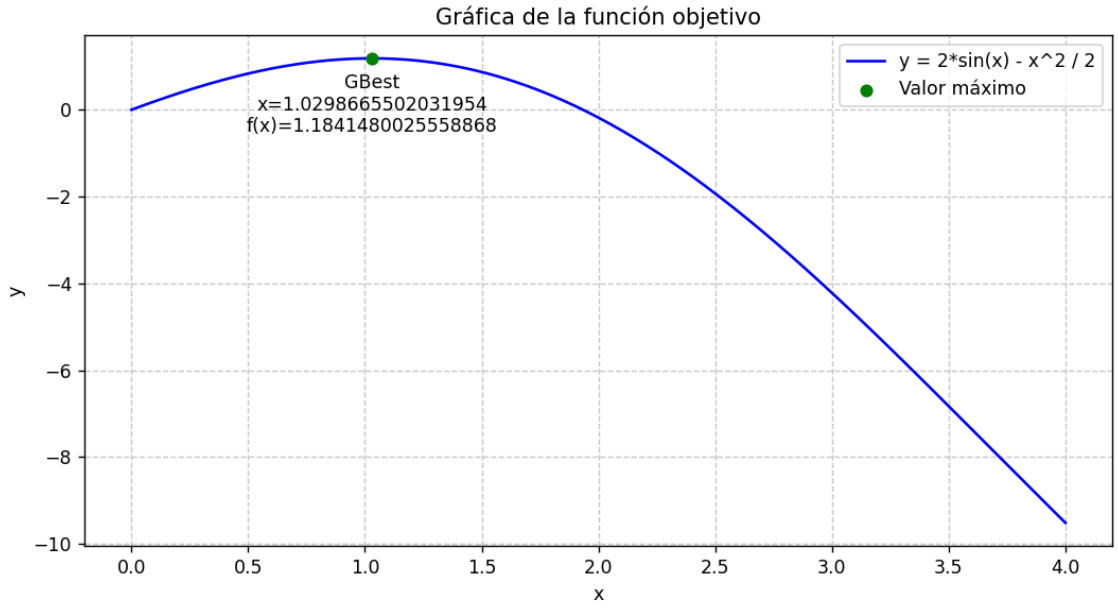
35	1.02986	1.18415
36	1.02986	1.18415
37	1.02986	1.18415
38	1.02986	1.18415
39	1.02986	1.18415
40	1.02986	1.18415
41	1.02986	1.18415
42	1.02986	1.18415
43	1.02986	1.18415
44	1.02986	1.18415
45	1.02986	1.18415
46	1.02986	1.18415
47	1.02986	1.18415
48	1.02986	1.18415
49	1.02986	1.18415
50	1.02986	1.18415
51	1.02986	1.18415
52	1.02986	1.18415
53	1.02986	1.18415
54	1.02986	1.18415
55	1.02986	1.18415
56	1.02986	1.18415
57	1.02986	1.18415
58	1.02986	1.18415
59	1.02986	1.18415
60	1.02986	1.18415
61	1.02986	1.18415
62	1.02987	1.18415

```
|              63 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              64 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              65 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              66 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              67 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              68 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              69 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              70 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              71 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              72 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              73 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              74 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              75 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              76 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              77 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              78 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              79 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+
|              80 |                1.02987 |                    1.18415 |
+-----------------+------------------------+----------------------------+


El valor óptimo es: 1.1841480025558868 para x= 1.0298665502031954
```

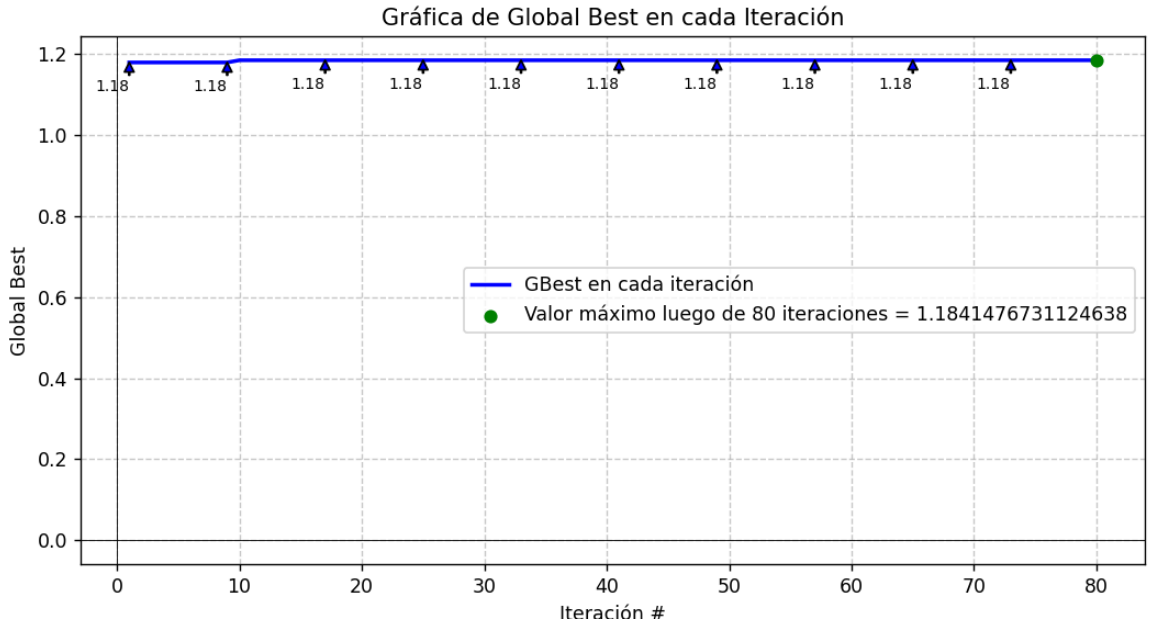
C. Indicar la URL del repositorio en donde se encuentra el algoritmo PSO.

https://github.com/gusjrivas/TP2_AEv/blob/main/ae_tp2_e1.py

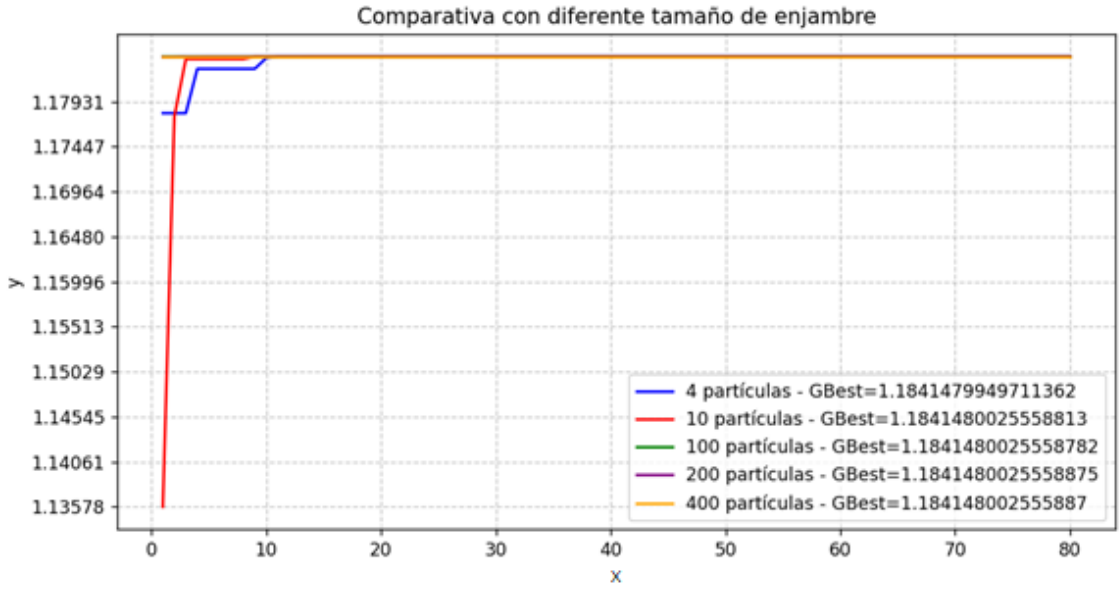
D. Graficar usando matplotlib la función objetivo y agregar un punto verde en donde el algoritmo haya encontrado el valor máximo. El gráfico debe contener etiquetas en los ejes, leyenda y un título.



E. Realizar un gráfico de línea que muestre gbest en función de las iteraciones realizadas.



F. Sobre el gráfico anterior superponer (con colores diferentes) 5 gráficos de línea de gbest en función de las iteraciones realizadas para ejecuciones con 4, 10, 100, 200 y 400 partículas respectivamente.



G. Realizar observaciones/comentarios/conclusiones sobre los resultados obtenidos en el ítem anterior.

- **Convergencia**: El algoritmo es capaz de encontrar un valor cercano al máximo global de la función en el intervalo dado, mostrando la efectividad de la PSO con los parámetros establecidos.
- **Influencia del Tamaño del Enjambre**: Aumentar el número de partículas generalmente mejora la precisión del óptimo global encontrado, pero a costa de un mayor tiempo de computación.
- **Importancia de los Parámetros**: Los valores de c1, c2, y w influyen directamente en la exploración y explotación del espacio de búsqueda. Un balance adecuado entre ellos permite al enjambre explorar nuevas áreas mientras converge hacia el mejor valor.
- **Visualización**: Los gráficos proporcionan una excelente manera de entender el comportamiento del algoritmo, mostrando tanto la evolución del valor global óptimo como la comparación entre diferentes tamaños de enjambre.

Ejercicio 2

Escribir un algoritmo PSO para la maximización de la función:

$$y = \sin(x) + \sin(x^2)$$

En el intervalo de $0 \leq x \leq 10$ y que cumpla con las siguientes consignas:

A. Transcribir el algoritmo en Python con los siguientes parámetros: número de partículas = 2, máximo número de iteraciones = 30, coeficientes de aceleración c1 = c2 = 1.49, peso de inercia w = 0.5.

```
########################################################################
# CEIA - 16Co2024 - Algoritmos Evolutivos - TP2 - Ejercicio 2
# Gustavo J. Rivas (a1620) | Myrna L. Degano (a1618)
########################################################################
# Algoritmo PSO para maximizar función.
########################################################################
import numpy as np
import matplotlib.pyplot as plt
from tabulate import tabulate

# Obtener parámetros de ejecución
def get_params():

    num_p = input("Número de partículas (DEFAULT: 2): ").strip()
    num_p = int(num_p) if num_p else 2

    num_i = input("Número de iteraciones (DEFAULT: 30): ").strip()
    num_i = int(num_i) if num_i else 30

    c1 = input("Coeficiente de aceleración - Componente cognitivo (DEFAULT:
1.49): ").strip()
```

```
    c1 = float(c1) if c1 else 1.49


    c2 = input("Coeficiente de aceleración  - Componente social (DEFAULT:
1.49): ").strip()
    c2 = float(c2) if c2 else 1.49


    w = input("Coeficiente de inercia (DEFAULT: 0.5): ").strip()
    w = float(w) if w else 0.5


    l = 0
    u = 10


    return num_p, num_i, c1, c2, w, l, u


# Función objetivo
def f_obj(x):
    return np.sin(x) + np.sin(x**2)



######################################################################
# Desarrollo del algoritmo
######################################################################
result = [["Iteración #", "GBest - Valor de x", "GBest - Valor de f(x)"]]


print("\nINGRESE LOS PARÁMETROS PARA LA EJECUCIÓN DEL ALGORITMO (O <ENTER>
PARA TOMAR LOS VALORES POR DEFAULT)\n")


# Obtener parámetros de ejecución
# Cantidad de partículas, máximo de iteraciones
# Coeficientes de aceleración e incercia (c1, c2, w)
# Límites del espacio (inferior lb, superior ub)
particles, iterations, c1, c2, w, lb, ub = get_params()
```

```
# Inicializar enjambre de partículas (posiciones y velocidades)
swarm = np.random.uniform(lb, ub, particles)
velocity = np.zeros(particles)

# Personal Best inicial para cada partícula
pbest = swarm.copy()
f_pbest = [f_obj(swarm[i]) for i in range(particles)]

# Global Best inicial
gbest = pbest[np.argmax(f_pbest)]
f_gbest = np.max(f_pbest)

# Búsqueda del óptimo
for i in range(iterations): # Máximo de iteraciones

    for p in range(particles):  # Iteración por partícula

        r1, r2 = np.random.rand(), np.random.rand()  # Aleatorios por cada
partícula/iteración

        # Actualizar velocidad de la partícula
        velocity[p] = (w * velocity[p] + c1 * r1 * (pbest[p] - swarm[p]) + c2
* r2 * (gbest - swarm[p]))

        # Actualizar posición de la partícula
        # # manteniéndola dentro de los limites del espacio de búsqueda
        swarm[p] = np.clip(swarm[p] + velocity[p], lb, ub)

        # Evaluar la función objetivo para la nueva posición de la partícula
        fitness = f_obj(swarm[p])
```

```
        # Actualizar el mejor personal
        if fitness > f_pbest[p]:
            f_pbest[p] = fitness
            pbest[p] = swarm[p].copy()

            # Actualizar del mejor global
            if fitness > f_gbest:
                f_gbest = fitness
                gbest = swarm[p].copy()

    # Resultados de la iteración
    result.append([i+1, gbest, f_gbest])



# Impresión de resultados
print(tabulate(result, headers="firstrow", tablefmt="grid"))
print(f"\nEl valor óptimo es: {f_gbest} para x= {gbest}")

# Gráfico de la función objetivo
x = np.linspace(lb, ub, 400)
y = np.sin(x) + np.sin(x**2)

plt.figure(figsize=(10, 5))
plt.plot(x, y, label='y = sin(x) + sin(x^2)', color='blue')

# Añadir el punto negro en el valor máximo
plt.scatter(gbest, f_gbest, color='black', zorder=5, label='Valor máximo')
plt.annotate(f'GBest\nx={gbest}\nf(x)={f_gbest}', (gbest, f_gbest),
textcoords="offset points", xytext=(0,-40), ha='center', zorder=10)

plt.title('Gráfica de la función objetivo')
plt.xlabel('x')
```

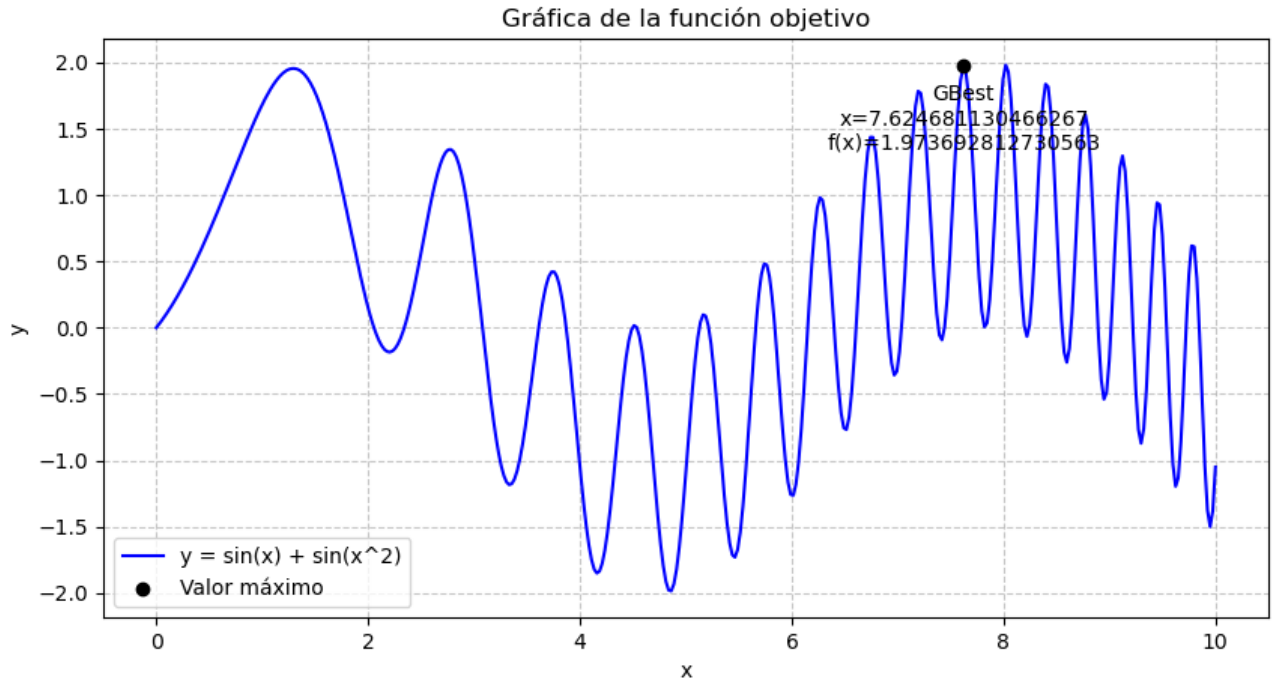
```
plt.ylabel('y')
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()


# Gráfico de GBest en función de las iteraciones
x = np.arange(1, iterations + 1)
y = np.array(result[1:])[:, 2]


plt.figure(figsize=(10, 5))
plt.plot(x, y, label='GBest en cada iteración', color='blue', linewidth=2)
plt.title("Gráfica de Global Best en cada Iteración")
plt.xlabel('Iteración #')
plt.ylabel('Global Best')
plt.scatter(iterations, f_gbest,
            color='black', zorder=5,
            label=f'Valor máximo luego de {iterations} iteraciones =
{f_gbest}')


step = iterations // 10 if iterations > 30 else 1
for i in range(0, len(x), step):
    plt.annotate(round(y[i], 2),
                 (x[i], y[i]),
                 fontsize=8,
                 textcoords="offset points",
                 xytext=(0, -15),
                 ha='right',
                 arrowprops=dict(facecolor='blue', shrink=0, width=0.5,
headwidth=5, headlength=5))


plt.axhline(0, color='black', linewidth=0.5)
plt.axvline(0, color='black', linewidth=0.5)
```

```
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()
plt.show()
```

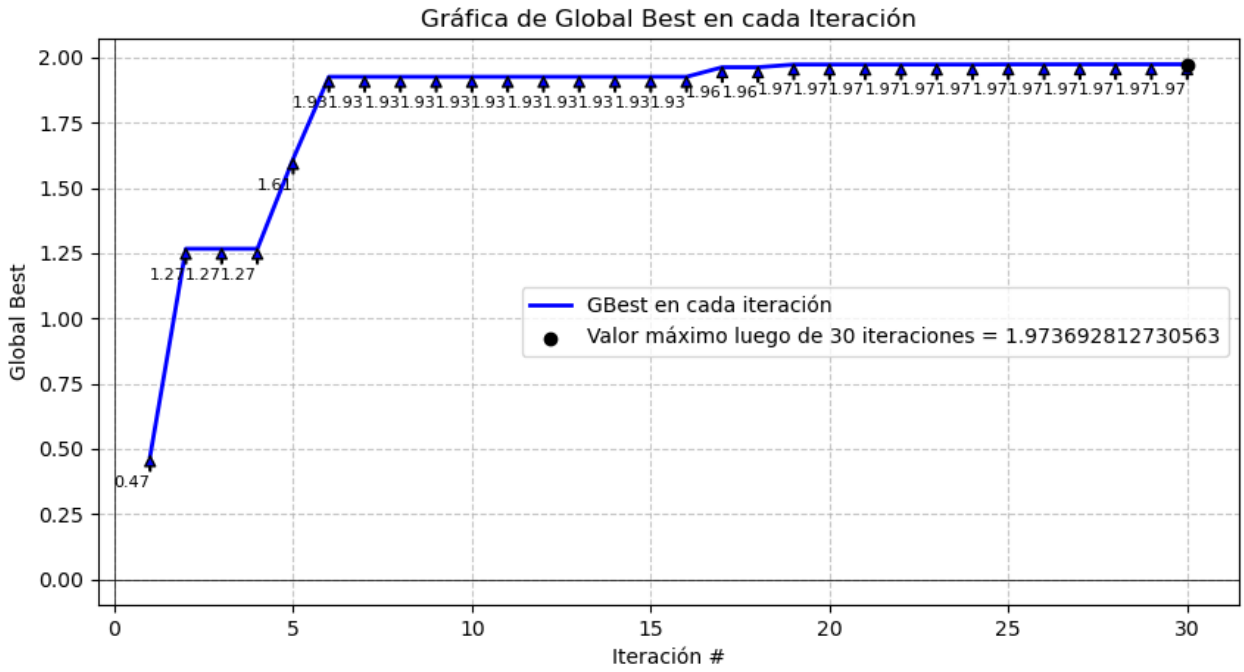
B. Indicar la URL del repositorio en donde se encuentra el algoritmo PSO.

https://github.com/gusjrivas/TP2_AEv/blob/main/ae_tp2_e2.py

C. Graficar usando matplotlib la función objetivo y agregar un punto negro en donde el algoritmo haya encontrado el valor máximo. El gráfico debe contener etiquetas en los ejes, leyenda y un título.



D. Realizar un gráfico de línea que muestre gbest en función de las iteraciones realizadas.



E. Transcribir la solución óptima encontrada (dominio) y el valor objetivo óptimo (imagen).

```
Número de partículas (DEFAULT: 2):
Número de iteraciones (DEFAULT: 30):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 1.49):
Coeficiente de aceleración - Componente social (DEFAULT: 1.49):
Coeficiente de inercia (DEFAULT: 0.5):
+---------------+----------------------+-------------------------+
|   Iteración # |   GBest - Valor de x |   GBest - Valor de f(x) |
+===============+======================+=========================+
|             1 |              5.73366 |                 0.47147 |
+---------------+----------------------+-------------------------+
|             2 |              6.80011 |                 1.26645 |
+---------------+----------------------+-------------------------+
|             3 |              6.80011 |                 1.26645 |
+---------------+----------------------+-------------------------+
|             4 |              6.80011 |                 1.26645 |
```

5	7.16009	1.611
6	7.64493	1.92574
7	7.64493	1.92574
8	7.64493	1.92574
9	7.64493	1.92574
10	7.64493	1.92574
11	7.64493	1.92574
12	7.64493	1.92574
13	7.64493	1.92574
14	7.64493	1.92574
15	7.64493	1.92574
16	7.64493	1.92574
17	7.61471	1.96232
18	7.61471	1.96232
19	7.62179	1.97278
20	7.62179	1.97278
21	7.62179	1.97278
22	7.62179	1.97278
23	7.62179	1.97278

```
|             24 |               7.62179 |                   1.97278 |
+----------------+-----------------------+---------------------------+
|             25 |               7.6261  |                   1.97343 |
+----------------+-----------------------+---------------------------+
|             26 |               7.6261  |                   1.97343 |
+----------------+-----------------------+---------------------------+
|             27 |               7.62369 |                   1.9736  |
+----------------+-----------------------+---------------------------+
|             28 |               7.62444 |                   1.97369 |
+----------------+-----------------------+---------------------------+
|             29 |               7.62444 |                   1.97369 |
+----------------+-----------------------+---------------------------+
|             30 |               7.62468 |                   1.97369 |
+----------------+-----------------------+---------------------------+

El valor óptimo es: 1.973692812730563 para x= 7.624681130466267
```

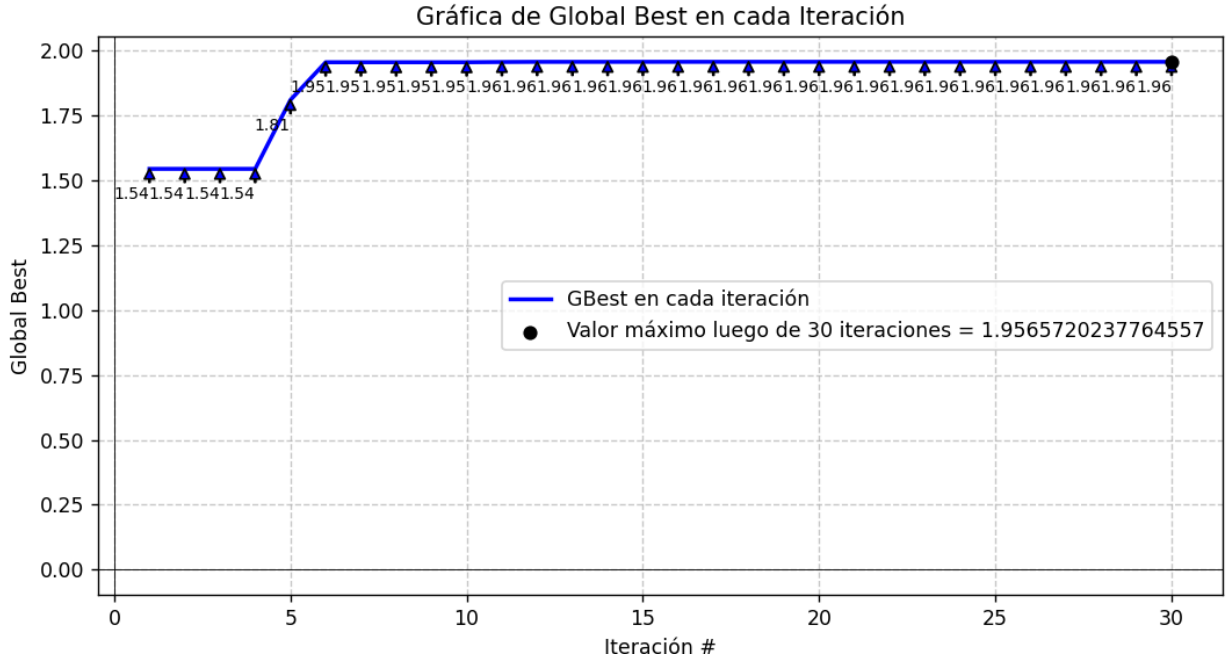
F. Incrementar el número de partículas a 4, ejecutar la rutina, transcribir la solución óptima encontrada, transcribir el valor objetivo óptimo y realizar nuevamente los gráficos solicitados en C y D.

```
Número de partículas (DEFAULT: 2): 4
Número de iteraciones (DEFAULT: 30):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 1.49):
Coeficiente de aceleración - Componente social (DEFAULT: 1.49):
Coeficiente de inercia (DEFAULT: 0.5):

+----------------+-----------------------+---------------------------+
|   Iteración # |   GBest - Valor de x |   GBest - Valor de f(x) |
+================+=======================+===========================+
|              1 |               0.919792 |                   1.54412 |
+----------------+-----------------------+---------------------------+
|              2 |               0.919792 |                   1.54412 |
+----------------+-----------------------+---------------------------+
|              3 |               0.919792 |                   1.54412 |
+----------------+-----------------------+---------------------------+
|              4 |               0.919792 |                   1.54412 |
+----------------+-----------------------+---------------------------+
|              5 |               1.08635  |                   1.80959 |
+----------------+-----------------------+---------------------------+
|              6 |               1.31513  |                   1.95492 |
+----------------+-----------------------+---------------------------+
|              7 |               1.31513  |                   1.95492 |
```

8	1.31513	1.95492
9	1.31513	1.95492
10	1.31513	1.95492
11	1.3083	1.95584
12	1.28909	1.95645
13	1.28909	1.95645
14	1.28909	1.95645
15	1.29039	1.9565
16	1.29538	1.95657
17	1.29538	1.95657
18	1.29538	1.95657
19	1.29538	1.95657
20	1.29457	1.95657
21	1.29457	1.95657
22	1.29457	1.95657
23	1.29457	1.95657
24	1.29457	1.95657
25	1.29457	1.95657
26	1.29477	1.95657
27	1.29477	1.95657
28	1.29467	1.95657
29	1.29467	1.95657
30	1.29467	1.95657

El valor óptimo es: 1.9565720237764557 para x= 1.2946716706873767


Gráfica de la función objetivo
GBest
x=1.2946716706873767
f(x)=1.9565720237764557
y = sin(x) + sin(x^2)
Valor máximo
x
y


Gráfica de Global Best en cada Iteración
1.54 1.54 1.54 1.54 1.81 1.95 1.95 1.95 1.95 1.95 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96 1.96
GBest en cada iteración
Valor máximo luego de 30 iteraciones = 1.9565720237764557
Global Best
Iteración #

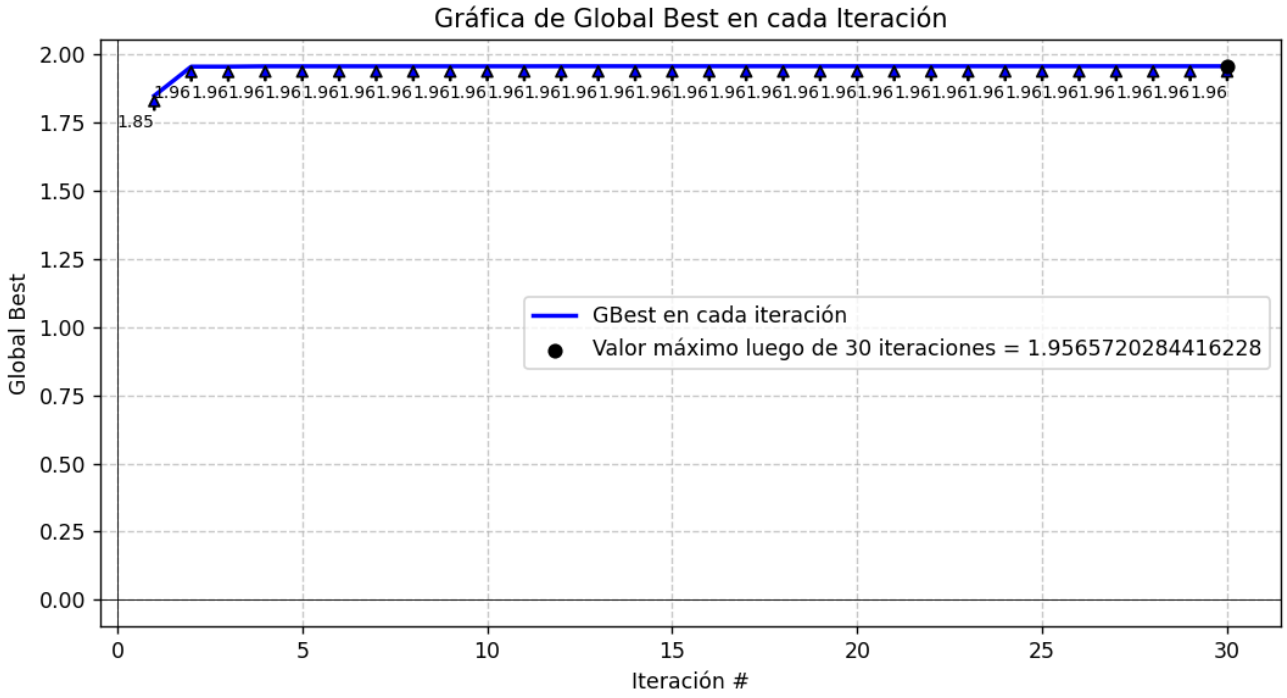
G. Incrementar el número de partículas a 6, ejecutar la rutina, transcribir la solución óptima encontrada, transcribir el valor objetivo óptimo y realizar nuevamente los gráficos solicitados en C y D.

```
Número de partículas (DEFAULT: 2): 6
Número de iteraciones (DEFAULT: 30):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 1.49):
Coeficiente de aceleración - Componente social (DEFAULT: 1.49):
Coeficiente de inercia (DEFAULT: 0.5):
```

Iteración #	GBest - Valor de x	GBest - Valor de f(x)
1	1.4547	1.84821
2	1.27507	1.95508
3	1.27507	1.95508
4	1.30244	1.95634
5	1.30244	1.95634
6	1.30244	1.95634
7	1.30244	1.95634
8	1.30244	1.95634
9	1.30244	1.95634
10	1.30244	1.95634
11	1.30244	1.95634
12	1.29562	1.95657
13	1.29562	1.95657
14	1.29562	1.95657
15	1.29417	1.95657
16	1.29417	1.95657
17	1.29417	1.95657
18	1.29417	1.95657
19	1.2944	1.95657
20	1.29483	1.95657

21	1.29483	1.95657
22	1.29483	1.95657
23	1.29472	1.95657
24	1.29472	1.95657
25	1.29472	1.95657
26	1.29472	1.95657
27	1.29472	1.95657
28	1.29472	1.95657
29	1.2947	1.95657
30	1.2947	1.95657

El valor óptimo es: 1.9565720284416228 para x= 1.2947017527974902



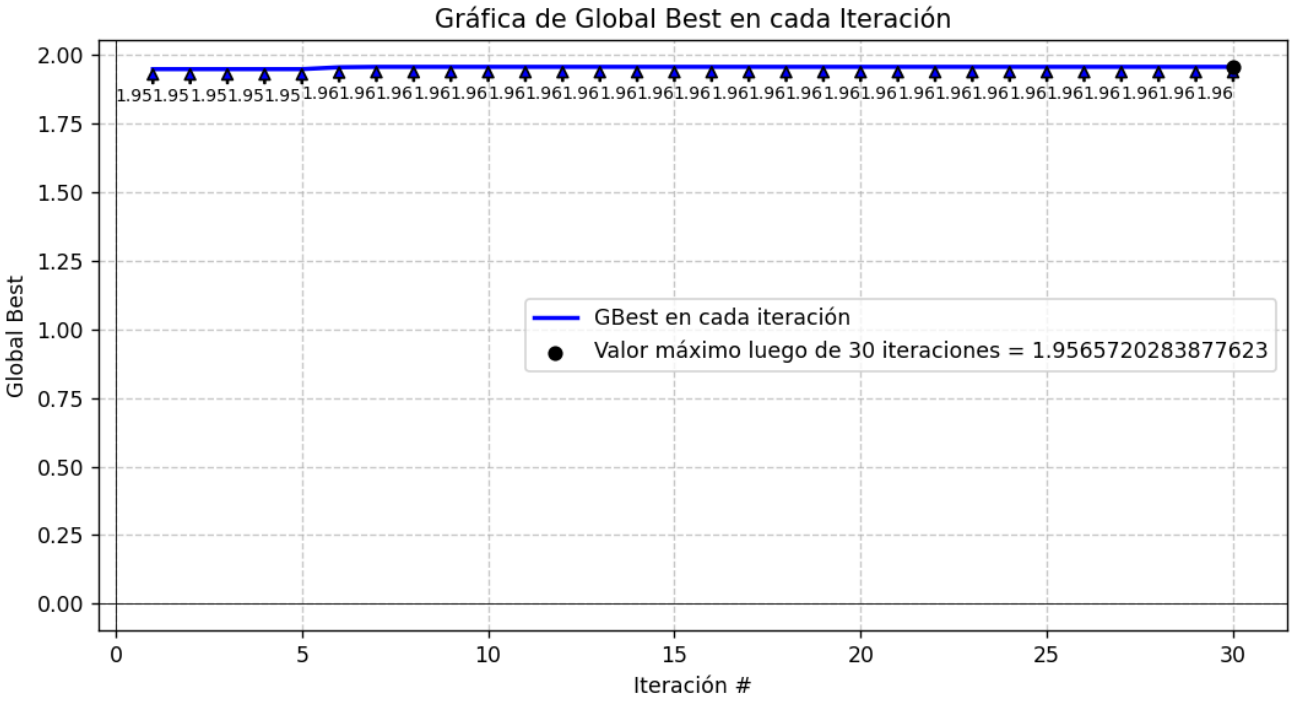
H. Incrementar el número de partículas a 10, ejecutar la rutina, transcribir la solución óptima encontrada, transcribir el valor objetivo óptimo y realizar nuevamente los gráficos solicitados en C y D.

```
Número de partículas (DEFAULT: 2): 10
Número de iteraciones (DEFAULT: 30):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 1.49):
Coeficiente de aceleración - Componente social (DEFAULT: 1.49):
Coeficiente de inercia (DEFAULT: 0.5):

+---------------+----------------------+--------------------------+
|   Iteración # |   GBest - Valor de x |   GBest - Valor de f(x)  |
+===============+======================+==========================+
|             1 |              1.34032 |                  1.94821 |
+---------------+----------------------+--------------------------+
|             2 |              1.34032 |                  1.94821 |
+---------------+----------------------+--------------------------+
|             3 |              1.34032 |                  1.94821 |
+---------------+----------------------+--------------------------+
|             4 |              1.34032 |                  1.94821 |
+---------------+----------------------+--------------------------+
|             5 |              1.34032 |                  1.94821 |
+---------------+----------------------+--------------------------+
|             6 |              1.31425 |                  1.95506 |
+---------------+----------------------+--------------------------+
|             7 |              1.2982  |                  1.95652 |
```

8	1.2982	1.95652
9	1.2944	1.95657
10	1.2944	1.95657
11	1.2944	1.95657
12	1.2944	1.95657
13	1.2944	1.95657
14	1.2944	1.95657
15	1.2944	1.95657
16	1.2949	1.95657
17	1.2949	1.95657
18	1.2949	1.95657
19	1.2949	1.95657
20	1.2949	1.95657
21	1.2949	1.95657
22	1.29453	1.95657
23	1.29467	1.95657
24	1.29467	1.95657
25	1.2947	1.95657
26	1.2947	1.95657
27	1.2947	1.95657
28	1.2947	1.95657
29	1.2947	1.95657
30	1.2947	1.95657

El valor óptimo es: 1.9565720283877623 para x= 1.2947004753613698


Gráfica de la función objetivo
GBest
x=1.2947004753613698
f(x)=1.9565720283877623
y = sin(x) + sin(x^2)
Valor máximo
x
y


Gráfica de Global Best en cada Iteración
GBest en cada iteración
Valor máximo luego de 30 iteraciones = 1.9565720283877623
Iteración #
Global Best

I. Realizar observaciones/comentarios/conclusiones sobre los resultados obtenidos.

Comparación de resultados

Nro. de partículas	Valor óptimo	Valor de x	Observaciones
2	1.97369	7.62468	Con sólo 2 partículas, el algoritmo no logra encontrar el valor óptimo cercano al mínimo real. La solución es significativamente diferente del valor esperado y el resultado es menos preciso.
4	1.95657	1.29467	Con 4 partículas, el algoritmo empieza a encontrar una solución más cercana al valor óptimo real. La función objetivo está cerca del valor óptimo y la precisión ha mejorado considerablemente en comparación con el caso de 2 partículas.
6	1.95657	1.29470	La solución es similar a la de 4 partículas, indicando que el aumento del número de partículas ha ayudado a mejorar la precisión, pero los resultados se estabilizan a un valor muy cercano al óptimo real. El algoritmo ha encontrado una solución precisa.
10	1.95657	1.2947	Con 10 partículas, el resultado es prácticamente el mismo que el obtenido con 4 y 6 partículas. Esto sugiere que, aunque el aumento del número de partículas mejora la convergencia y la estabilidad, después de cierto punto, añadir más partículas tiene un impacto marginal en la precisión de la solución.

Conclusiones generales:

- **Mejora con el número de partículas**: A medida que se incrementa el número de partículas, el algoritmo mejora en la precisión y rapidez de convergencia hacia el valor óptimo. Con 2 partículas, el algoritmo no logra encontrar un valor cercano al óptimo real. A partir de 4 partículas, la precisión aumenta considerablemente.
- **Estabilización**: Con 6 y 10 partículas, los resultados son muy similares y se estabilizan en valores muy cercanos al óptimo real. Esto indica que después de un número adecuado de partículas (en este caso, alrededor de 4 a 6), la precisión adicional que se obtiene al añadir más partículas es marginal.
- **Eficiencia**: Para equilibrar precisión y eficiencia, un número de partículas en el rango de 4 a 6 parece ser adecuado. Utilizar más partículas no mejora significativamente la solución y puede aumentar el tiempo de cómputo sin beneficios adicionales sustanciales.

En resumen, se observa que, aumentar el número de partículas mejora la precisión del algoritmo hasta cierto punto, después del cual se estabiliza y añadir más partículas no aporta mejoras significativas.

Ejercicio 3

Dada la siguiente función perteneciente a un paraboloide elíptico de la forma:

$$f(x, y) = (x - a)^2 + (y + b)^2 \qquad (1)$$

donde, las constantes a y b son valores reales ingresados por el usuario a través de la consola, con intervalos de:

$$-100 \leq x \leq 100; x \in \mathbb{R}$$

$$-100 \leq y \leq 100; y \in \mathbb{R}$$

$$-50 \leq a \leq 50; a \in \mathbb{R}$$

$$-50 \leq b \leq 50; b \in \mathbb{R}$$

escribir en Python un algoritmo PSO para la minimización de la función (1) que cumpla con las siguientes consignas:

A. Transcribir el algoritmo utilizando los siguientes parámetros: número de partículas = 20, máximo número de iteraciones = 10, coeficientes de aceleración c1 = c2 = 2, peso de inercia w = 0.7.

```
######################################################################
# CEIA - 16Co2024 - Algoritmos Evolutivos - TP2 - Ejercicio 3
# Gustavo J. Rivas (a1620) | Myrna L. Degano (a1618)
######################################################################
# Algoritmo PSO para minimizar paraboloide elíptico.
######################################################################
import numpy as np
import matplotlib.pyplot as plt
import io
import sys
from tabulate import tabulate
from pyswarm import pso

# Obtener parámetros de ejecución
```

```
def get_params():

    num_p = input("Número de partículas (DEFAULT: 20): ").strip()
    num_p = int(num_p) if num_p else 20

    num_i = input("Número de iteraciones (DEFAULT: 10): ").strip()
    num_i = int(num_i) if num_i else 20

    c1 = input("Coeficiente de aceleración - Componente cognitivo (DEFAULT: 2):
").strip()
    c1 = float(c1) if c1 else 2.0

    c2 = input("Coeficiente de aceleración  - Componente social (DEFAULT: 2):
").strip()
    c2 = float(c2) if c2 else 2.0

    w = input("Coeficiente de inercia (DEFAULT: 0.7): ").strip()
    w = float(w) if w else 0.7

    l = input("Límite inferior para las variables x e y (DEFAULT: -100):
").strip()
    l = float(l) if l else -100.0

    u = input("Límite superior para las variables x e y (DEFAULT: +100):
").strip()
    u = float(u) if u else 100.0

    return num_p, num_i, c1, c2, w, l, u

# Obtener inputs
def get_inputs(min=-50, max=50):
    while True:
        try:
            print("\nIngrese los valores de \"a\" y \"b\"")
            print(f"Deben ser valores reales entre {min} y {max} (ambos
incluidos)")

            a = float(input("a = ").strip())
            b = float(input("b = ").strip())
```

```
            if min <= a <= max and min <= b <= max:
                return a, b
            else:
                print("Los valores ingresados no son válidos.")

        except ValueError:
            print("(!) Entrada no válida.")

# Función objetivo (Paraboloide elíptico)
def f_obj(xy):
    global a
    global b

    x, y = xy

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

# Gráfico de la función objetivo 3D
def graph_obj_3d(gX, gY, gZ, s1, s2, redP, redV):

    global a
    global b

    fig = plt.figure(figsize=(s1, s2))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(gX, gY, gZ, cmap='viridis')

    # Punto rojo en el mínimo encontrado
    ax.scatter(redP[0], redP[1], redV, color='red', marker='o', s=100)

    ax.set_title(f"Gráfico de la función objetivo\nf(x, y) = (x - {a})^2 + (y +
{b})^2")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.grid(True)
    plt.show()

# Gráfico de la función objetivo 2D
```

```
def graph_obj_2d(gX, gY, gZ, s1, s2, redP, redV):
    global a
    global b

    plt.figure(figsize=(s1, s2))
    contour = plt.contourf(gX, gY, gZ, levels=50, cmap='viridis')
    plt.colorbar(contour)
    plt.scatter([redP[0]], [redP[1]], color='red', zorder=5)
    plt.title(f"Gráfico de la función objetivo\nf(x, y) = (x - {a})^2 + (y +
{b})^2")
    plt.xlabel('x')
    plt.ylabel('y')
    plt.legend([f"Mínimo encontrado\n({redP[0]}, {redP[1]})"])
    plt.grid(True)
    plt.show()


# Gráfico de línea para GBest por iteración
def line_graph(gx, gy, s1, s2, bestV):
    global iterations

    plt.figure(figsize=(s1, s2))

    plt.plot(gx, gy, label='Global Best', color='blue', linestyle='-',
linewidth=2, marker='o')
    plt.xticks(range(min(gx), max(gx) + 1))

    for i in range(len(gx)):
        plt.annotate(round(gy[i], 2), (gx[i], gy[i]), fontsize=8,
textcoords="offset points", xytext=(0, 10),
                     ha='right')

    # Configurar el eje x para que muestre números enteros
    plt.gca().xaxis.set_major_locator(plt.MaxNLocator(integer=True))

    plt.xlabel("Iteración #")
    plt.ylabel("GBest")
    plt.title("Gráfico de Global Best por iteración")
    plt.legend(
```

```
        loc='upper right',
        title=f"Luego de {iterations} iteraciones \n{bestV}",
        fontsize='small',
        frameon=True,  # Mostrar el marco
        edgecolor='black',  # Color del borde del marco
    )
    # plt.legend(f"Global Best luego de {iterations} iteraciones\n{f_gbest}")
    plt.grid(True)
    plt.show()

class CaptureOutput(io.StringIO):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.records = []

    def write(self, data):
        super().write(data)
        try:
            self.records.append(data)
        except ValueError:
            pass

######################################################################
# Desarrollo del algoritmo
######################################################################
result = [["Iteración #", "GBest - Valor de x", "GBest - Valor de y", "GBest -
Valor de f(x, y)"]]
best_values = []

print("\nINGRESE LOS PARÁMETROS PARA LA EJECUCIÓN DEL ALGORITMO (O <ENTER> PARA
TOMAR LOS VALORES POR DEFAULT)\n")

# Dimensiones (x, y)
dim = 2

# Obtener parámetros de ejecución
# Cantidad de partículas, máximo de iteraciones
# Coeficientes de aceleración e incercia (c1, c2, w)
# Límites del espacio (inferior lb, superior ub)
```

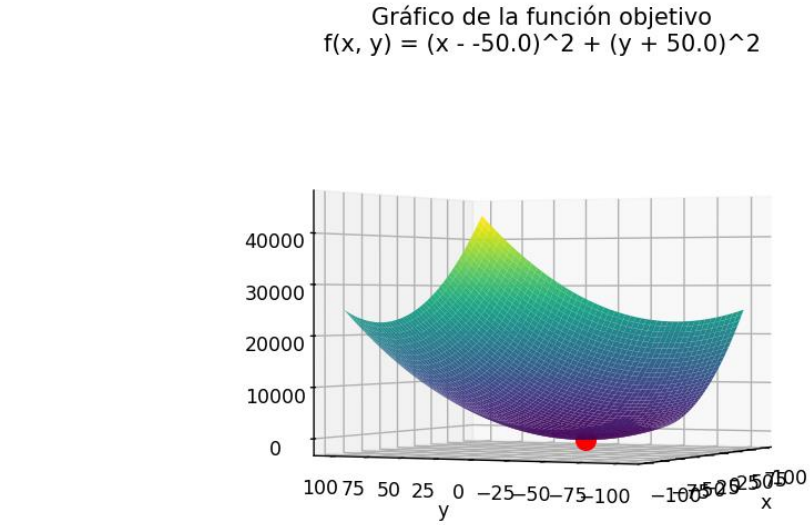
```
particles, iterations, c1, c2, w, lb, ub = get_params()

# Obtener inputs a y b
a, b = get_inputs()

# Inicializar enjambre de partículas (posiciones y velocidades)
swarm = np.random.uniform(lb, ub, (particles, dim))
velocity = np.zeros((particles, dim))

# Personal Best inicial para cada partícula
pbest = swarm.copy()
f_pbest = [f_obj([swarm[i][0], swarm[i][1]]) for i in range(particles)]

# Global Best inicial
gbest = pbest[np.argmin(f_pbest)]
f_gbest = np.min(f_pbest)

# Búsqueda del óptimo
for i in range(iterations): # Máximo de iteraciones

    for p in range(particles):  # Iteración por partícula

        r1, r2 = np.random.rand(), np.random.rand()   # Aleatorios por cada
partícula/iteración

        # Em cada dimensión x, y
        for d in range(dim):
            # Actualizar velocidad de la partícula en cada dimensión
            velocity[p][d] = (w * velocity[p][d] + c1 * r1 * (pbest[p][d] -
swarm[p][d]) + c2 * r2 * (gbest[d] - swarm[p][d]))

            # Actualizar posición de la partícula
            # # manteniéndola dentro de los limites del espacio de búsqueda
            swarm[p][d] = np.clip(swarm[p][d] + velocity[p][d], lb, ub)

        # Evaluar la función objetivo para la nueva posición de la partícula
        fitness = f_obj([swarm[p][0], swarm[p][1]])

        # Actualizar el mejor personal
```

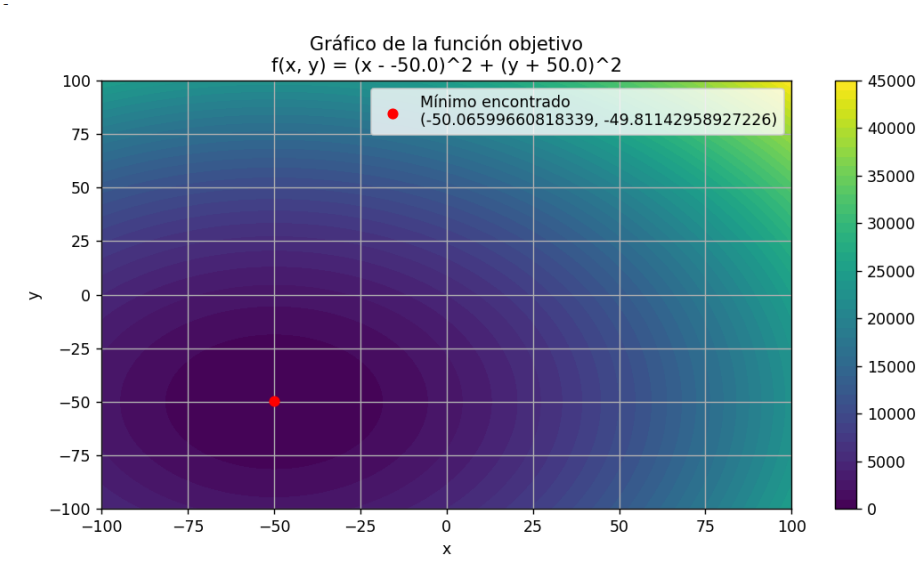
```
        if fitness < f_pbest[p]:
            f_pbest[p] = fitness
            pbest[p] = swarm[p].copy()

            # Actualizar del mejor global
            if fitness < f_gbest:
                f_gbest = fitness
                gbest = swarm[p].copy()

    # Resultados de la iteración
    result.append([i+1, gbest[0], gbest[1], f_gbest])


# Impresión de resultados
print(tabulate(result, headers="firstrow", tablefmt="grid"))
print(f"\nEl valor óptimo es: {f_gbest} para los valores (x, y): {gbest}")


# Gráficos
X, Y = np.meshgrid(np.linspace(lb, ub, 400), np.linspace(lb, ub, 400))
Z = f_obj([X, Y])

x = np.arange(1, iterations + 1)
y = np.array(result[1:])[:, 3]

# Gráfico de la función objetivo en 3D
graph_obj_3d (X, Y, Z, 10, 5, gbest, f_gbest)

# Gráfico de dispersión para la función objetivo
graph_obj_2d (X, Y, Z, 10, 5, gbest, f_gbest)

# Gráfico de GBest por iteración
line_graph (x, y, 10, 5, f_gbest)
```

B. Indicar la URL del repositorio en donde se encuentra el algoritmo PSO.

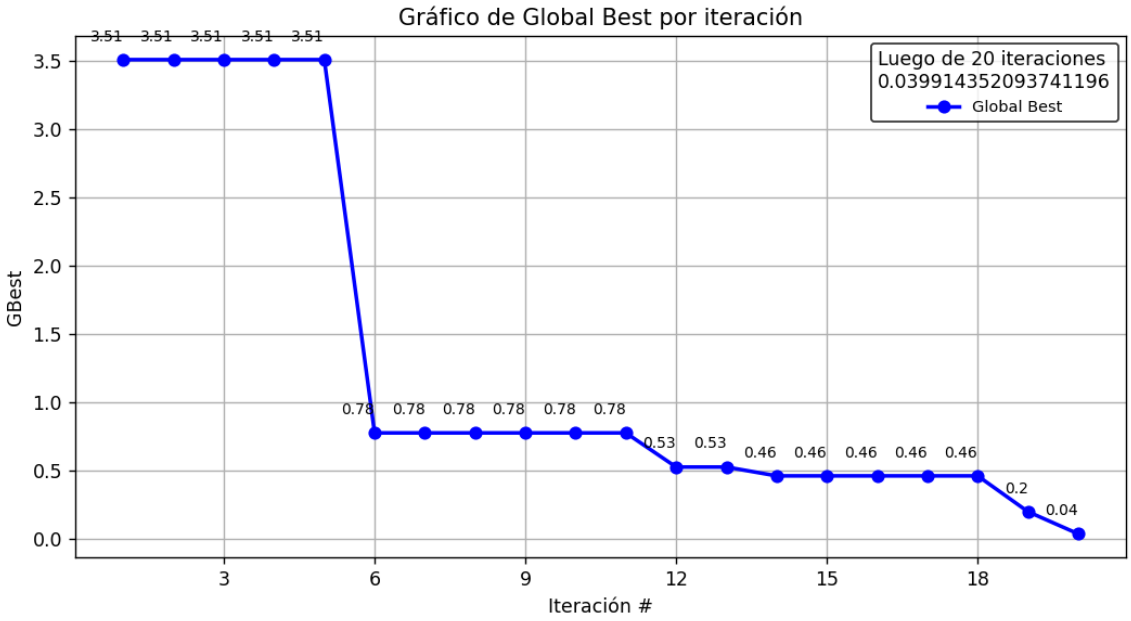
https://github.com/gusjrivas/TP2_AEv/blob/main/ae_tp2_e3.py

C. Graficar usando matplotlib la función objetivo f(x, y) y agregar un punto rojo en donde el algoritmo haya encontrado el valor mínimo. El gráfico debe contener etiquetas en los ejes, leyenda y un título.





D. Realizar un gráfico de línea que muestre gbest en función de las iteraciones realizadas.



E. Transcribir la solución óptima encontrada (dominio) y el valor objetivo óptimo (imagen).

```
Número de partículas (DEFAULT: 20):
Número de iteraciones (DEFAULT: 10):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 2):
Coeficiente de aceleración - Componente social (DEFAULT: 2):
Coeficiente de inercia (DEFAULT: 0.7):
Límite inferior para las variables x e y (DEFAULT: -100):
Límite superior para las variables x e y (DEFAULT: +100):

Ingrese los valores de "a" y "b"
Deben ser valores reales entre -50 y 50 (ambos incluidos)
a = -50
b = 50
```

Iteración #	GBest - Valor de x	GBest - Valor de y	GBest - Valor de f(x, y)
1	-50.9734	-48.4006	3.50567
2	-50.9734	-48.4006	3.50567
3	-50.9734	-48.4006	3.50567
4	-50.9734	-48.4006	3.50567
5	-50.9734	-48.4006	3.50567
6	-50.0668	-50.8794	0.777831
7	-50.0668	-50.8794	0.777831
8	-50.0668	-50.8794	0.777831
9	-50.0668	-50.8794	0.777831
10	-50.0668	-50.8794	0.777831

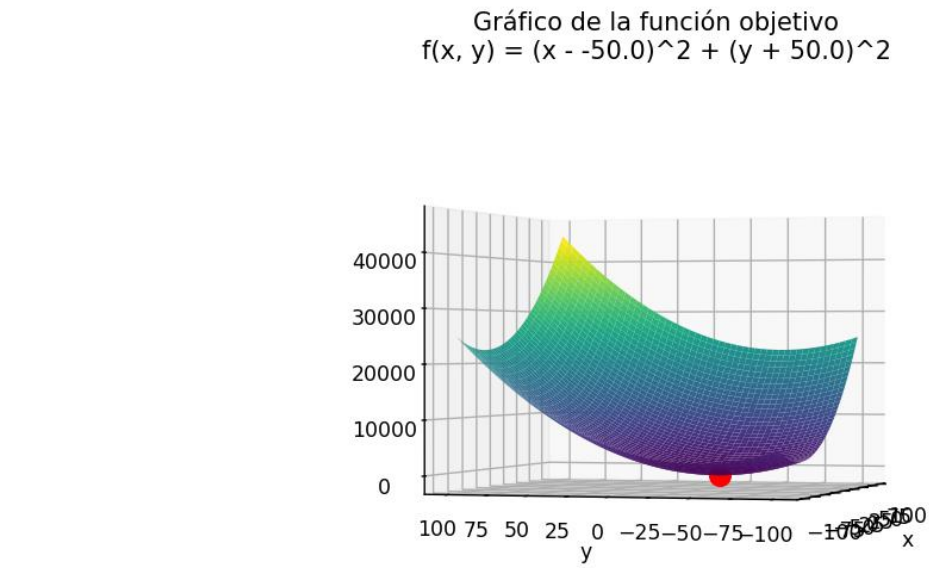
```
+---------------+----------------------+----------------------+----------------------------+
|            11 |             -50.0668 |             -50.8794 |                   0.777831 |
+---------------+----------------------+----------------------+----------------------------+
|            12 |             -49.7848 |             -50.6943 |                   0.528283 |
+---------------+----------------------+----------------------+----------------------------+
|            13 |             -49.7848 |             -50.6943 |                   0.528283 |
+---------------+----------------------+----------------------+----------------------------+
|            14 |             -50.4412 |             -49.4813 |                   0.463762 |
+---------------+----------------------+----------------------+----------------------------+
|            15 |             -50.4412 |             -49.4813 |                   0.463762 |
+---------------+----------------------+----------------------+----------------------------+
|            16 |             -50.4412 |             -49.4813 |                   0.463762 |
+---------------+----------------------+----------------------+----------------------------+
|            17 |             -50.4412 |             -49.4813 |                   0.463762 |
+---------------+----------------------+----------------------+----------------------------+
|            18 |             -50.4412 |             -49.4813 |                   0.463762 |
+---------------+----------------------+----------------------+----------------------------+
|            19 |             -49.6314 |             -50.2573 |                   0.202017 |
+---------------+----------------------+----------------------+----------------------------+
|            20 |             -50.066  |             -49.8114 |                   0.0399144 |
+---------------+----------------------+----------------------+----------------------------+


El valor óptimo es: 0.039914352093741196 para los valores (x, y): [-50.06599661 -49.81142959]
```

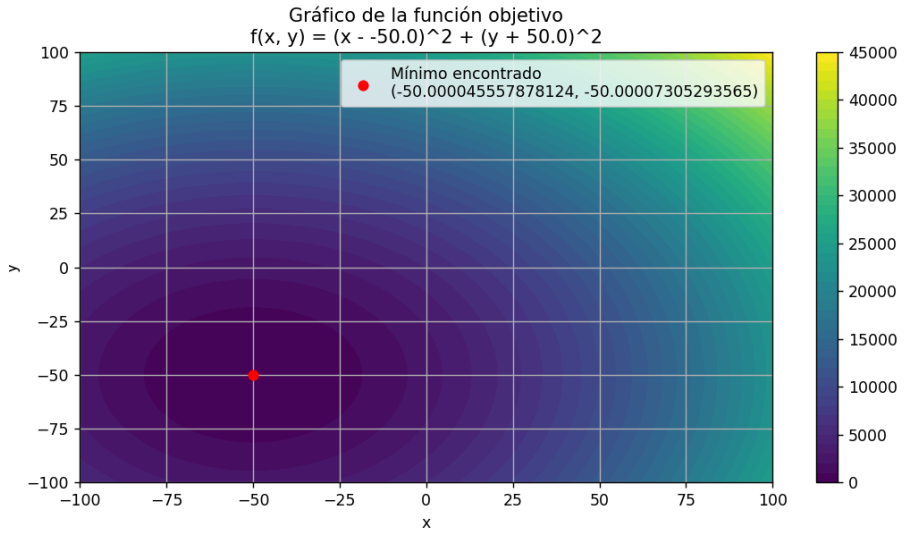
F. Establecer el coeficiente de inercia w en 0, ejecutar el algoritmo y realizar observaciones/comentarios/conclusiones sobre los resultados observados.

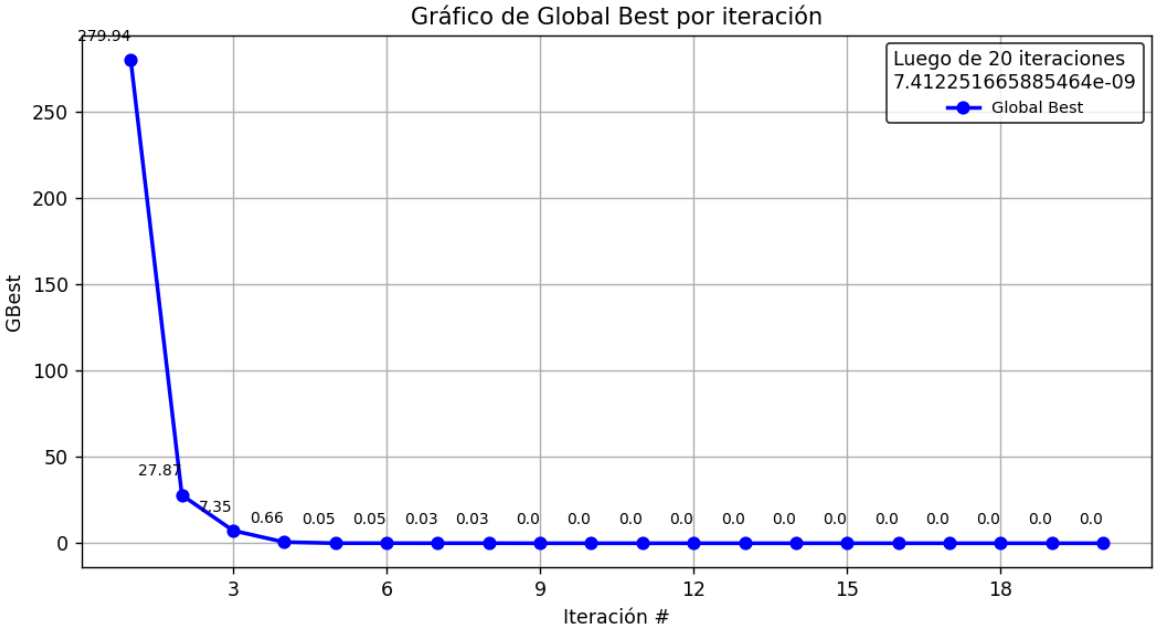
```
Número de partículas (DEFAULT: 20):
Número de iteraciones (DEFAULT: 10):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 2):
Coeficiente de aceleración - Componente social (DEFAULT: 2):
Coeficiente de inercia (DEFAULT: 0.7): 0
Límite inferior para las variables x e y (DEFAULT: -100):
Límite superior para las variables x e y (DEFAULT: +100):

Ingrese los valores de "a" y "b"
Deben ser valores reales entre -50 y 50 (ambos incluidos)
a = -50
b = 50
+---------------+----------------------+----------------------+----------------------------+
|   Iteración # |   GBest - Valor de x |   GBest - Valor de y |   GBest - Valor de f(x, y) |
+===============+======================+======================+============================+
|             1 |             -33.4348 |             -52.3532 |                    279.945 |
+---------------+----------------------+----------------------+----------------------------+
|             2 |             -48.5231 |             -44.932  |                    27.8659 |
+---------------+----------------------+----------------------+----------------------------+
|             3 |             -50.4444 |             -47.3252 |                    7.35207 |
+---------------+----------------------+----------------------+----------------------------+
|             4 |             -50.773  |             -49.7438 |                    0.663193 |
+---------------+----------------------+----------------------+----------------------------+
|             5 |             -50.0085 |             -49.7871 |                    0.0454069 |
+---------------+----------------------+----------------------+----------------------------+
|             6 |             -50.0085 |             -49.7871 |                    0.0454069 |
+---------------+----------------------+----------------------+----------------------------+
|             7 |             -50.1062 |             -49.8533 |                    0.0327918 |
+---------------+----------------------+----------------------+----------------------------+
|             8 |             -50.1049 |             -49.8525 |                    0.0327767 |
+---------------+----------------------+----------------------+----------------------------+
|             9 |             -50.0123 |             -49.9953 |                    0.000173073 |
+---------------+----------------------+----------------------+----------------------------+
|            10 |             -50.0012 |             -50.0013 |                    3.08426e-06 |
+---------------+----------------------+----------------------+----------------------------+
|            11 |             -49.9997 |             -50.0009 |                    9.03982e-07 |
+---------------+----------------------+----------------------+----------------------------+
```

12	-49.9996	-50.0007	6.05641e-07
13	-49.9996	-50.0006	5.56294e-07
14	-50.0001	-50.0005	2.60801e-07
15	-50.0001	-50.0005	2.60801e-07
16	-50.0002	-50.0005	2.50643e-07
17	-50.0002	-50	3.50515e-08
18	-50.0002	-50	3.46984e-08
19	-50.0002	-50	2.87266e-08
20	-50	-50.0001	7.41225e-09

```
El valor óptimo es: 7.412251665885464e-09
para los valores (x, y): [-50.00004556 -50.00007305]
```







Conclusiones:

Al poner en cero el coeficiente de inercia estamos eliminando por completo el término que indica cuánto de la velocidad (o dirección) anterior de la partícula se conserva en la nueva velocidad.

En consecuencia, se observa:

- **Pérdida de velocidad de inercia**: La velocidad de una partícula ya no conservará ninguna parte de su velocidad anterior. La velocidad se actualizará únicamente en función de la diferencia entre la mejor posición conocida por la partícula y su posición actual, y la diferencia entre la mejor posición global y la posición actual.
- **Convergencia rápida pero riesgosa**: En los gráficos de GBest por iteración vemos que la convergencia fue más rápida, pero dependiendo del problema, esto podría ser riesgoso. Las partículas pueden converger rápidamente a una solución local sin explorar adecuadamente otras posibles soluciones. Esto se debe a que no hay un componente de inercia que permita un movimiento más suave y gradual a través del espacio de búsqueda.

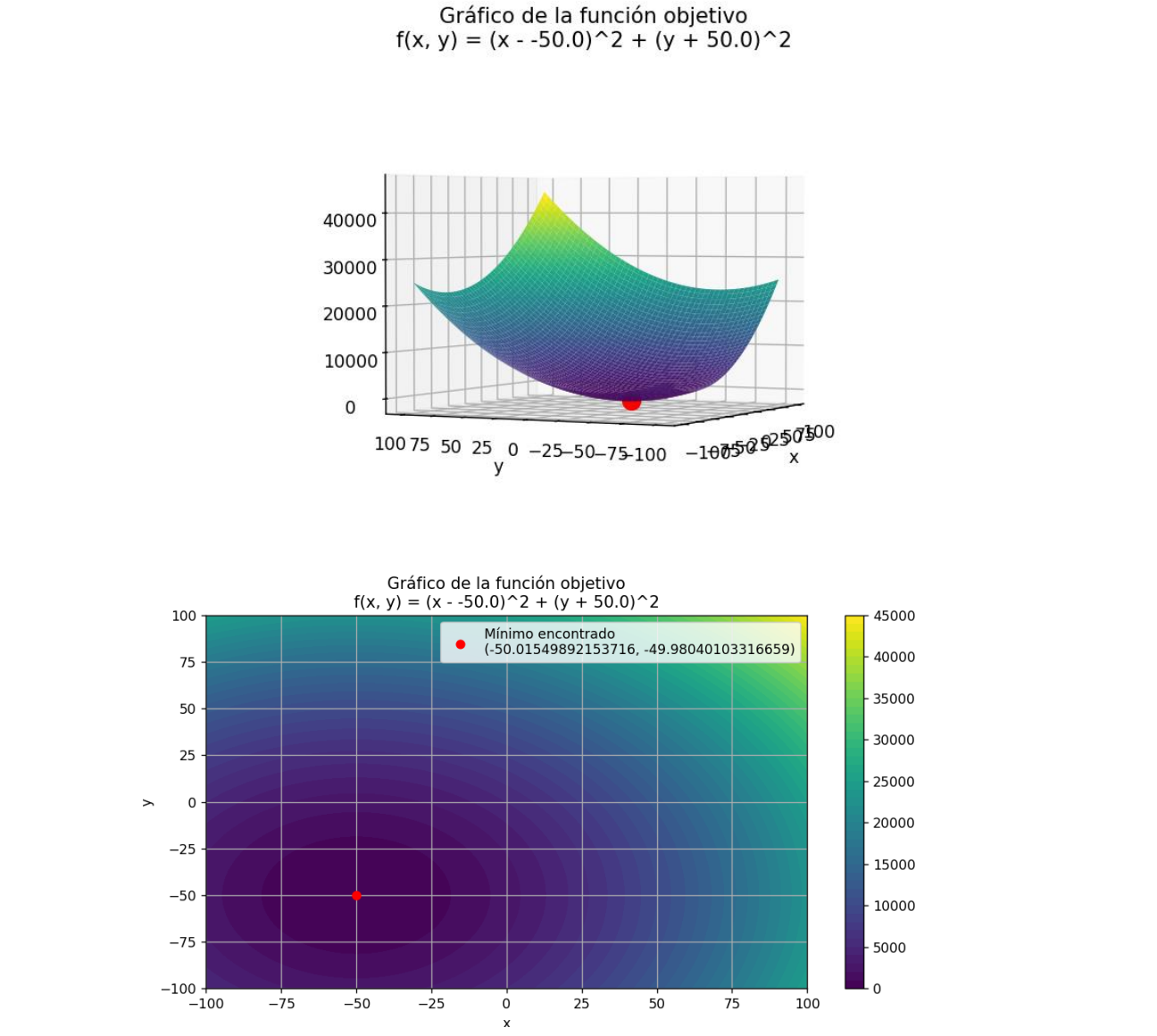
Por lo tanto, cambiar el coeficiente de inercia a 0 en un algoritmo PSO elimina la influencia de la velocidad anterior en la actualización de la velocidad, lo que puede llevar a una exploración menos efectiva del espacio de búsqueda y una mayor posibilidad de quedarse atrapado en óptimos locales. Es fundamental encontrar un valor adecuado para el coeficiente de inercia que permita un equilibrio entre exploración y explotación.

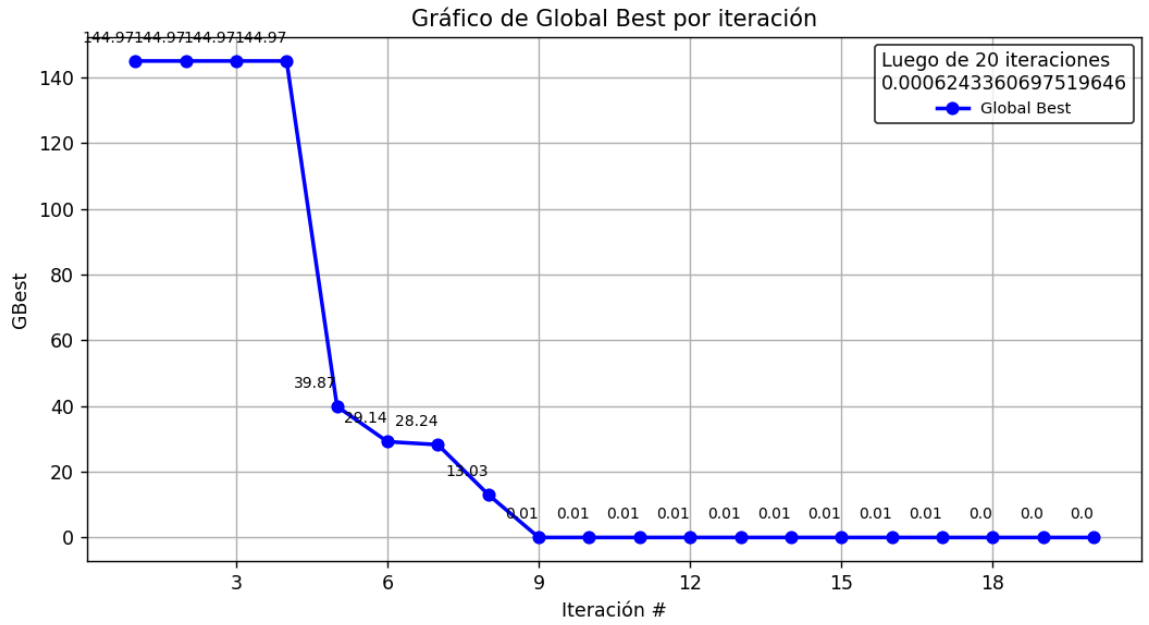
G. Reescribir el algoritmo PSO para que cumpla nuevamente con los ítems A hasta F pero usando la biblioteca pyswarm (from pyswarm import pso).

(continúa del código en el punto A.)

```
#########################################################################
# Resultados utilizando PYSWARM
#########################################################################

# Redirigir la salida estándar para capturar los mensajes de PSO
original_stdout = sys.stdout
capture_output = CaptureOutput()
sys.stdout = capture_output

# Ejecutar PSO en modo Debug
best_pos, best_val = pso(
    f_obj, # Función objetivo
    [lb, lb], # Límites inferiores
    [ub, ub], # Límites superiores
    swarmsize=particles, # Tamaño del enjambre
    maxiter=iterations, # Número máximo de iteraciones
    debug=True # Modo debug
)

# Restaurar la salida estándar
sys.stdout = original_stdout

# PSO Debug
pso_log_arr = capture_output.records
pso_log_str = ''.join(pso_log_arr)
print(pso_log_str)

print(f"El valor óptimo utilizando pyswarm es: {best_val} para los valores (x,
y): {best_pos}")

# Gráfico de la función objetivo en 3D
graph_obj_3d (X, Y, Z, 10, 5, best_pos, best_val)
```

```
# Gráfico de dispersión para la función objetivo
graph_obj_2d (X, Y, Z, 10, 5, best_pos, best_val)

# Gráfico de GBest por iteración
pso_log_lines = [s for s in pso_log_arr if s.startswith('Best after iteration')]
for v in pso_log_lines:
    vals = v.split()
    best_values.append(float(vals[-1]))

line_graph (x, best_values, 10, 5, best_val)
```






```
Best after iteration 1: [-61.06085247 -45.24339159] 144.96778093472452
Best after iteration 2: [-61.06085247 -45.24339159] 144.96778093472452
Best after iteration 3: [-61.06085247 -45.24339159] 144.96778093472452
Best after iteration 4: [-61.06085247 -45.24339159] 144.96778093472452
New best for swarm at iteration 5: [-44.62368523 -53.31211718] 39.87488069533987
Best after iteration 5: [-44.62368523 -53.31211718] 39.87488069533987
New best for swarm at iteration 6: [-55.34086898 -49.21263644] 29.144822832215123
Best after iteration 6: [-55.34086898 -49.21263644] 29.144822832215123
New best for swarm at iteration 7: [-49.38249983 -44.72218532] 28.236634235335412
Best after iteration 7: [-49.38249983 -44.72218532] 28.236634235335412
New best for swarm at iteration 8: [-45.49125853 -50.51605451] 20.595061931392454
New best for swarm at iteration 8: [-47.28485787 -52.3794396 ] 13.033729589662308
Best after iteration 8: [-47.28485787 -52.3794396 ] 13.033729589662308
New best for swarm at iteration 9: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 9: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 10: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 11: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 12: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 13: [-50.08758101 -49.96241773] 0.009082860100751204
Best after iteration 14: [-50.08758101 -49.96241773] 0.009082860100751204
New best for swarm at iteration 15: [-50.03836714 -50.05966947] 0.005032483054626554
Best after iteration 15: [-50.03836714 -50.05966947] 0.005032483054626554
Best after iteration 16: [-50.03836714 -50.05966947] 0.005032483054626554
Best after iteration 17: [-50.03836714 -50.05966947] 0.005032483054626554
New best for swarm at iteration 18: [-50.01549892 -49.98040103] 0.0006243360697519646
Best after iteration 18: [-50.01549892 -49.98040103] 0.0006243360697519646
Best after iteration 19: [-50.01549892 -49.98040103] 0.0006243360697519646
Best after iteration 20: [-50.01549892 -49.98040103] 0.0006243360697519646
Stopping search: maximum iterations reached --> 20

El valor óptimo utilizando pyswarm es: 0.0006243360697519646
para los valores (x, y): [-50.01549892 -49.98040103]
```

H. Realizar observaciones/comentarios/conclusiones comparando los resultados obtenidos sin pyswarm y con pyswarm.

Comparación de Resultados

Algoritmo	**Mejor valor encontrado**	**Posición**
Sin Pyswarm con w=0.7	0.0399143520937411960	[−50.06599661,−49.81142959]
Sin Pyswarm con w=0	7.412251665885464e-09	[−50.00004556,−50.00007305]
Con Pyswarm	0.0006243360697519646	[−50.01549892,−49.98040103]

Observaciones

Precisión y estabilidad:

- Pyswarm muestra una alta precisión en la optimización, con un valor final muy cercano a cero. Esto indica que el método utilizado es altamente efectivo en encontrar una solución muy precisa al problema.
- Con w=0, también se logró una alta precisión, con un valor muy cercano al mínimo teórico, lo que indica que la convergencia rápida fue efectiva en este caso específico.

Convergencia:

- Pyswarm mostró una mejora constante a lo largo de las iteraciones y alcanzó un valor óptimo con precisión, a pesar de que el valor final no es exactamente cero, es extremadamente bajo.
- Con w=0.7, la convergencia fue más lenta y el valor final fue mayor en comparación con los otros métodos, indicando que una exploración más amplia puede ser beneficiosa en ciertos escenarios.

En resumen, **Pyswarm** parece ser una opción robusta para la optimización por enjambre de partículas, ofreciendo una alta precisión en la convergencia al mínimo global en comparación con configuraciones manuales.

Ejercicio 4

Mediante PSO es posible resolver **en forma aproximada** un sistema de ***n*** ecuaciones con ***n*** incógnitas clásico del tipo:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases} \quad (2)$$

Por ejemplo, el siguiente es un sistema de 2 ecuaciones con 2 incógnitas (x_1 y x_2) que puede ser resuelto con PSO:

$$\begin{cases} 3x_1 + 2x_2 = 9 \\ x_1 - 5x_2 = 4 \end{cases} \quad (3)$$

Utilizando la biblioteca pyswarm:

A. Escribir un algoritmo PSO con parámetros a elección (c1, c2, w, número de partículas, máximo número de iteraciones) que encuentre x1 y x2 para el sistema de ecuaciones anterior (3). Transcribir el código fuente.

```
######################################################################
# CEIA - 16Co2024 - Algoritmos Evolutivos - TP2 - Ejercicio 4
# Gustavo J. Rivas (a1620) | Myrna L. Degano (a1618)
######################################################################
# Sistema de 2 ecuaciones con 2 incógnitas resuelto con PSO.
######################################################################

from pyswarm import pso

# Obtener parámetros de ejecución
def get_params():
```

```
    num_p = input("Número de partículas (DEFAULT: 20): ").strip()
    num_p = int(num_p) if num_p else 20

    num_i = input("Número de iteraciones (DEFAULT: 50): ").strip()
    num_i = int(num_i) if num_i else 50

    c1 = input("Coeficiente de aceleración - Componente cognitivo (DEFAULT:
1.5): ").strip()
    c1 = float(c1) if c1 else 1.5

    c2 = input("Coeficiente de aceleración  - Componente social (DEFAULT: 1.5):
").strip()
    c2 = float(c2) if c2 else 1.5

    w = input("Coeficiente de inercia (DEFAULT: 0.5): ").strip()
    w = float(w) if w else 0.5

    l = input("Límite inferior para las variables x e y (DEFAULT: -100):
").strip()
    l = float(l) if l else -100.0

    u = input("Límite superior para las variables x e y (DEFAULT: +100):
").strip()
    u = float(u) if u else 100.0

    return num_p, num_i, c1, c2, w, l, u

# Función objetivo (Suma de cuadrados de las ecuaciones -> Error a minimizar)
def f_obj(x1x2):

    x1, x2 = x1x2

    # Sistema de ecuaciones
    # 3*x1 + 2*x2 - 9 = 0 => f1 (x1, x2) = 3*x1 + 2*x2 - 9
    # x1 - 5*x2 - 4 = 0 => f1 (x1, x2) = x1 - 5*x2 - 4

    # Función Objetivo a minimizar: f1^2 + f2^2
    return (3*x1 + 2*x2 - 9)**2 + (x1 - 5*x2 - 4)**2
```

```
#########################################################################
# Desarrollo del algoritmo
#########################################################################

print("\nINGRESE LOS PARÁMETROS PARA LA EJECUCIÓN DEL ALGORITMO (O <ENTER> PARA
TOMAR LOS VALORES POR DEFAULT)\n")

# Obtener parámetros de ejecución
# Cantidad de partículas, máximo de iteraciones
# Coeficientes de aceleración e incercia (c1, c2, w)
# Límites del espacio (inferior lb, superior ub)
particles, iterations, c1, c2, w, lb, ub = get_params()

# Ejecutar PSO en modo Debug
best_pos, best_val = pso(
    f_obj, # Función objetivo
    [lb, lb], # Límites inferiores
    [ub, ub], # Límites superiores
    swarmsize=particles, # Tamaño del enjambre
    maxiter=iterations, # Número máximo de iteraciones
    debug=True # Modo debug
)

print(f"\nLos valores aproximados encontrados para resolver el sistema de
ecuaciones son:\n x1, x2 = {best_pos}")
```

B. Transcribir los valores de x_1 y x_2 encontrados por el algoritmo.

```
Número de partículas (DEFAULT: 20):
Número de iteraciones (DEFAULT: 50):
Coeficiente de aceleración - Componente cognitivo (DEFAULT: 1.5):
Coeficiente de aceleración  - Componente social (DEFAULT: 1.5):
Coeficiente de inercia (DEFAULT: 0.5):
Límite inferior para las variables x e y (DEFAULT: -100):
Límite superior para las variables x e y (DEFAULT: +100):

No constraints given.
Best after iteration 1: [-1.69215576e+01  6.71883144e-06] 4009.527476794138
New best for swarm at iteration 2: [-10.21788671   5.21069763] 2476.3089431526514
Best after iteration 2: [-10.21788671   5.21069763] 2476.3089431526514
```

```
Best after iteration 3: [-10.21788671   5.21069763] 2476.3089431526514
Best after iteration 4: [-10.21788671   5.21069763] 2476.3089431526514
New best for swarm at iteration 5: [-1.04019873 -2.14428149] 301.53663704081174
Best after iteration 5: [-1.04019873 -2.14428149] 301.53663704081174
Best after iteration 6: [-1.04019873 -2.14428149] 301.53663704081174
New best for swarm at iteration 7: [ 3.29140994 -0.66576839] 7.074849534500977
Best after iteration 7: [ 3.29140994 -0.66576839] 7.074849534500977
Best after iteration 8: [ 3.29140994 -0.66576839] 7.074849534500977
Best after iteration 9: [ 3.29140994 -0.66576839] 7.074849534500977
New best for swarm at iteration 10: [ 2.83164773 -0.38175863] 2.157532890883734
Best after iteration 10: [ 2.83164773 -0.38175863] 2.157532890883734
Best after iteration 11: [ 2.83164773 -0.38175863] 2.157532890883734
New best for swarm at iteration 12: [ 3.04407143 -0.35668946] 1.0225395463918803
Best after iteration 12: [ 3.04407143 -0.35668946] 1.0225395463918803
New best for swarm at iteration 13: [ 3.07640035 -0.02194271] 0.6967524213994106
Best after iteration 13: [ 3.07640035 -0.02194271] 0.6967524213994106
Best after iteration 14: [ 3.07640035 -0.02194271] 0.6967524213994106
New best for swarm at iteration 15: [ 3.07516477 -0.13643746] 0.06112293508106646
Best after iteration 15: [ 3.07516477 -0.13643746] 0.06112293508106646
New best for swarm at iteration 16: [ 3.08733703 -0.15472419] 0.021582978185898112
Best after iteration 16: [ 3.08733703 -0.15472419] 0.021582978185898112
Best after iteration 17: [ 3.08733703 -0.15472419] 0.021582978185898112
Best after iteration 18: [ 3.08733703 -0.15472419] 0.021582978185898112
New best for swarm at iteration 19: [ 3.09813399 -0.19585899] 0.015465643344260871
Best after iteration 19: [ 3.09813399 -0.19585899] 0.015465643344260871
Best after iteration 20: [ 3.09813399 -0.19585899] 0.015465643344260871
New best for swarm at iteration 21: [ 3.13433855 -0.19384172] 0.010957086965882396
New best for swarm at iteration 21: [ 3.08809052 -0.17191111] 0.009069240947218019
New best for swarm at iteration 21: [ 3.09911452 -0.18508593] 0.005906380157935086
New best for swarm at iteration 21: [ 3.09457865 -0.17498494] 0.005316980173148961
Best after iteration 21: [ 3.09457865 -0.17498494] 0.005316980173148961
Best after iteration 22: [ 3.09457865 -0.17498494] 0.005316980173148961
New best for swarm at iteration 23: [ 3.10746087 -0.17323158] 0.001275842405922753
Best after iteration 23: [ 3.10746087 -0.17323158] 0.001275842405922753
New best for swarm at iteration 24: [ 3.11353823 -0.17487345] 0.00022967506455642448
Best after iteration 24: [ 3.11353823 -0.17487345] 0.00022967506455642448
New best for swarm at iteration 25: [ 3.11618256 -0.17637429] 2.1434497421284428e-05
Best after iteration 25: [ 3.11618256 -0.17637429] 2.1434497421284428e-05
Best after iteration 26: [ 3.11618256 -0.17637429] 2.1434497421284428e-05
Best after iteration 27: [ 3.11618256 -0.17637429] 2.1434497421284428e-05
New best for swarm at iteration 28: [ 3.1167611  -0.17624978] 8.87190821237231e-06
Best after iteration 28: [ 3.1167611  -0.17624978] 8.87190821237231e-06
New best for swarm at iteration 29: [ 3.11699276 -0.17628476] 5.0392751749073156e-06
Best after iteration 29: [ 3.11699276 -0.17628476] 5.0392751749073156e-06
Best after iteration 30: [ 3.11699276 -0.17628476] 5.0392751749073156e-06
New best for swarm at iteration 31: [ 3.11760219 -0.1765655 ] 2.8986966291420137e-07
Best after iteration 31: [ 3.11760219 -0.1765655 ] 2.8986966291420137e-07
Best after iteration 32: [ 3.11760219 -0.1765655 ] 2.8986966291420137e-07
New best for swarm at iteration 33: [ 3.11768147 -0.1765338 ] 1.2338519011004643e-07
Best after iteration 33: [ 3.11768147 -0.1765338 ] 1.2338519011004643e-07
Best after iteration 34: [ 3.11768147 -0.1765338 ] 1.2338519011004643e-07
New best for swarm at iteration 35: [ 3.11762791 -0.17653282] 1.1836496401331196e-07
Stopping search: Swarm best objective change less than 1e-08
```

Los valores aproximados encontrados para resolver el sistema de ecuaciones son:

x1, x2 = [3.11762791 -0.17653282]

C. Indicar la URL del repositorio en donde se encuentra el algoritmo PSO.

https://github.com/gusjrivas/TP2_AEv/blob/main/ae_tp2_e4.py

D. Realizar observaciones/comentarios/conclusiones sobre:

(i) ¿Cómo eligió los limites superior e inferior de x_1 y x_2?.

Observaciones:

- Los límites superior e inferior para las variables x_1y x_2 fueron establecidos por ingresos del usuario, con valores predeterminados de -100 y 100, respectivamente, si no se proporcionaban valores específicos.
- Estos límites definen el rango dentro del cual el algoritmo PSO busca la solución óptima. Elegir límites más amplios puede permitir al algoritmo explorar un rango más amplio de posibles soluciones, pero también puede hacer que el algoritmo tarde más en converger o que sea menos preciso si el rango es demasiado amplio en relación con el problema.

Comentario:

- Para problemas específicos como el sistema de ecuaciones planteado, si se conoce el rango probable donde se encuentran las soluciones, se pueden ajustar los límites para mejorar la eficiencia de la búsqueda. En el ejemplo, los límites elegidos parecen adecuados dado que se encontró una solución muy cercana al valor esperado, pero podrían ser ajustados con base en el conocimiento previo del problema.

(ii) ¿PSO puede resolver un sistema de n ecuaciones con n incógnitas no lineal?. Demostrar.

Observaciones:

- El PSO es un algoritmo de optimización que se basa en la búsqueda en el espacio de soluciones a través de un enjambre de partículas, y puede resolver problemas de optimización con restricciones y objetivos complejos, incluyendo sistemas de ecuaciones no lineales.
- En el caso del sistema de ecuaciones proporcionado, se construyó una función objetivo que mide el error cuadrático de las ecuaciones dadas. Minimizar esta función

objetivo efectivamente busca encontrar los valores de x_1 y x_2 que hacen que las ecuaciones sean cero.

Demostración:

Para resolver un sistema de n ecuaciones con n incógnitas no lineales utilizando PSO, podemos reformular el problema de manera que sea adecuado para la optimización por enjambre de partículas.

Dado un sistema de n ecuaciones no lineales:

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) = 0 \\ f_2(x_1, x_2, \ldots, x_n) = 0 \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) = 0 \end{cases}$$

Es posible definir una función de costo que mida la magnitud del error de cada ecuación:

$$C(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} f_i(x_1, x_2, \ldots, x_n)^2$$

Esta función de costo C es la suma de los cuadrados de las ecuaciones, que será cero si y solo si el sistema de ecuaciones es satisfecho.

De esta forma, convertimos el sistema de ecuaciones en un problema de optimización minimizando una función objetivo que puede ser resuelto aplicando PSO.

(iii) ¿Cómo logró resolver el ejercicio?.

Observaciones:

- El ejercicio se resolvió usando PSO para minimizar una función objetivo construida a partir del sistema de ecuaciones. La función objetivo es la suma de los cuadrados de las ecuaciones del sistema, de modo que, al minimizar esta suma, se aproximan las ecuaciones a cero, es decir, se resuelven las ecuaciones.

- Se ajustaron varios parámetros del PSO, incluyendo el número de partículas, el número de iteraciones, los coeficientes de aceleración y el coeficiente de inercia.

Comentario:

- PSO utiliza un enfoque de optimización basado en la búsqueda y ajuste iterativo de las posiciones de las partículas en el espacio de soluciones, guiadas por la mejor posición global encontrada hasta el momento. La función objetivo utilizada en este caso es adecuada para convertir el problema de resolución de ecuaciones en un problema de minimización.

(iv) ¿Los resultados obtenidos guardan relación directa con los valores de los parámetros elegidos?. Demostrar.

Observaciones:

- Los resultados obtenidos dependen de los parámetros del algoritmo PSO, tales como el número de partículas, el número de iteraciones, los coeficientes de aceleración (cognitivo y social) y el coeficiente de inercia. Estos parámetros afectan la capacidad del algoritmo para explorar y explotar el espacio de soluciones.

Demostración:

- En el ejemplo proporcionado, se observa que el mejor valor obtenido para la función objetivo mejora con el aumento de iteraciones y ajustes en los parámetros del algoritmo. Un número mayor de iteraciones y un ajuste fino de los parámetros pueden llevar a una mejor aproximación de la solución.
- Por ejemplo, el valor de la función objetivo mejora de manera significativa a medida que el algoritmo avanza en las iteraciones, lo que indica que los parámetros y la cantidad de iteraciones influyen directamente en la calidad de la solución.
- Ajustar los coeficientes de aceleración y el coeficiente de inercia también impacta el comportamiento del algoritmo en términos de exploración y explotación, lo cual puede afectar la precisión y rapidez con la que se encuentra una solución óptima.

Ejemplos:

Nro. de partículas	Nro. de iteraciones	C1	C2	W	Límites de x_1 y x_2	x_1	x_2	$f(x_1, x_2)$	Best After Iteration
20	50	1.5	1.5	0.5	[-100..100]	3.1176	-0.1765	0.00000005	33
2	50	1.5	1.5	0.5	[-100..100]	-4.0837	-13.3013	5703,20016353	50
10	50	1.5	1.5	0.5	[-100..100]	3.1176	-0.1765	0.00000005	40
10	50	1.2	1.2	0.3	[-100..100]	3.1176	-0.1765	0.00000005	9
2	200	1.5	1.5	0.5	[-100..100]	2.8646	-0.0901	0.8125	200

Conclusión:

- Los resultados del PSO están íntimamente relacionados con los parámetros elegidos. Ajustar estos parámetros permite controlar el equilibrio entre la exploración del espacio de soluciones y la explotación de las mejores soluciones encontradas, lo cual es crucial para obtener resultados óptimos en la resolución de problemas complejos.

Anexo – Código fuente

El código fuente correspondiente a los algoritmos desarrollados se encuentra en el siguiente repositorio de *GitHub:*
https://github.com/gusjrivas/TP2_AEv/

- Ejercicio 1: **`ae_tp2_e1.py`**
- Ejercicio 2: **`ae_tp2_e2.py`**
- Ejercicio 3: **`ae_tp2_e3.py`**
- Ejercicio 4: **`ae_tp2_e4.py`**

Bibliografía

https://campusposgrado.fi.uba.ar/
CEIA - Algoritmos Evolutivos - Material teórico práctico
*4. Optimización por enjambre de partículas (PSO) (*uba-ceia-ae-04.pdf*)*