INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE COMPUTO

2024

SESIÓN DE LABORATORIO 6: ALGORITMO DE OPTIMIZACIÓN POR ENJAMBRE DE PARTÍCULAS

ALGORITMOS BIOINSPIRADOS

Elaborado por: Flores Estopier Rodrigo

Boleta: 2021630260

Profesor: Rosas Trigueros Jorge Luis

Grupo: 6CV2

Fecha de realización: 25/10/2024

Fecha de entrega: 29/10/2024

Semestre: 25-1

Sesión de Laboratorio 6: Algoritmo de optimización por enjambre de partículas

Contenido

1. MARCO TEÓRICO

La optimización por enjambres de partículas (PSO por su siglas en ingles) es un tipo de algoritmo bioinspirados, es una forma simplificada de encontrar una solución optima en un espacio de búsqueda establecido. Se diferencia de otros algoritmos de optimización, ya que solo necesita de la funcion objetivo y no requiere de una forma diferencial o gradiente del objetivo. Además, cuenta con pocos hiper parámetros.

Este algoritmo fue propuesto por Kennedy y Eberhart en 1995. Quienes tomaron como base el comportamiento de un grupo de aves, quienes se pueden beneficiar entre ella al momento de buscar alimento, donde una de esas aves podría compartir el descubrimiento de comida y liderar a todo el grupo hacia ella.

Conceptos Clave de PSO

1. **Partículas**: Cada partícula es un vector en el espacio de búsqueda, que representa una solución candidata. Su posición y velocidad son actualizadas en cada iteración para acercarse al óptimo de la función.
2. **Velocidad y Posición**: La velocidad define el cambio en la posición de la partícula en cada iteración. Las actualizaciones de posición y velocidad permiten a las partículas explorar diferentes áreas del espacio de búsqueda.
3. **Mejor posición individual (pBest)**: Es la mejor posición que ha encontrado una partícula en función del valor de la función objetivo.
4. **Mejor posición global (gBest)**: Es la mejor posición encontrada por cualquiera de las partículas hasta el momento. Las partículas usan esta posición como referencia para mejorar su posición.
5. **Parámetros del PSO**:
    - **Inercia**: Controla cuánto mantiene la partícula de su velocidad anterior, lo que afecta su capacidad de exploración.
    - **Componentes cognitivo y social**: Fomentan la exploración de la experiencia individual y del grupo, respectivamente. Estos parámetros permiten que la partícula se enfoque en su mejor experiencia y en la del grupo.

Función de Rosenbrock

La función de Rosenbrock, también conocida como el "valle de Rosenbrock" o la "función de banana", es una función no convexa utilizada comúnmente como banco de pruebas para algoritmos de optimización. Es particularmente desafiante debido a su valle curvado, que lleva al mínimo global, haciendo difícil para los algoritmos de optimización converger al mínimo exacto.

El mínimo global de la función de Rosenbrock en n-dimensiones se encuentra en x=(1,1,…,1), donde su valor es 0.

Función de Schwefel

La función de Schwefel es otra función de prueba de optimización común, conocida por su amplio espacio de búsqueda y por tener múltiples mínimos locales que complican la búsqueda de un óptimo global.

El mínimo global se encuentra x=(420.9687,420.9687,…,420.9687), donde el valor de la función es cercano a 0.

2. DESARROLLO

Se utilizará el ejemplo proporcionado en clase para realizar algoritmos PSO que nos permitan optimizar las funciones de Rosenbrock y Schwefel.

CODIFICACIÓN

Optimización de la funcion de Rosenbrock

Se utilizaron las siguientes librerías para la realización del algoritmo.

```
import ipywidgets as widgets
from IPython import display as display
import matplotlib.pyplot as plt
import numpy as np
```

- Ipwidgets, IPython: Se utilizará para crear un botón que nos permita avanzar entre las iteraciones del algoritmo.
- Matplotlib.pyplot: Nos permitirá visualizar de forma grafica la funcion de Rosenbrock y como interactúan las partículas.

Establecemos los valores del espacio de búsqueda, en este caso, el limite inferior es -2 y el límite superior es 2. Además establecemos un numero de 10 partículas, las cuales estarán diseñadas para interactuar en un espacio de 3 dimensiones.

```
lower_limit = -2
upper_limit = 2

n_particles = 10
n_dimensions = 3
```

Posteriormente, definimos nuestra funcion objetivo, en este caso la funcion de Rosenbrock.

```
# Definir la función de Rosenbrock en 3D
def rosenbrock(X):
    x = X[0]
    y = X[1]
    z = X[2]
    return (x-1)**2 + 100 * (y - x**2)**2 + (y-1)**2 + 100 * (z - y**2)**2
```

Debemos inicializar las posiciones de las partículas y establecer su velocidad inicial. El siguiente código crea una matriz donde cada fila representa una partícula y las columnas son su valor de posición, en este caso 3 columnas para una funcion de 3 dimensiones. Se establece que las posiciones estén dentro del espacio de búsqueda, y estas se asignan de forma aleatorio y uniforme.

También asignamos de forma uniforme y aleatoria las velocidades iniciales de las partículas, estas velocidades también se guardan en una matriz.

```
# Inicializar posiciones y velocidades de partículas
X = lower_limit + (upper_limit - lower_limit) *
np.random.rand(n_particles, n_dimensions)
V = -(upper_limit - lower_limit) / 2 + (upper_limit - lower_limit) *
np.random.rand(n_particles, n_dimensions)
```

Una vez creadas las particulas, inicializamos nuestras variables para establecer la mejor posición local y su valor de aptitud, y, la mejor posición global y su aptitud.

Al iniciar el algorimto, establecemos al mejor valor global como infinito, debido a que en este caso queremos encontrar un mínimo.

Posteriormente, con un ciclo for encontramos el mejor valor global hasta ahora y guardamos su posición.

```
# Inicializar las mejores posiciones
fitness_gbest = np.inf
fitness_lbest = fitness_gbest * np.ones(n_particles)
X_lbest = 1 * X
X_gbest = 1 * X_lbest[0]

for I in range(0, n_particles):
    if rosenbrock(X_lbest[I]) < rosenbrock(X_gbest):
        X_gbest = 1 * X_lbest[I]
```

Ahora, se explicara el funcionamiento del algoritmo en la funcion iteration, la cual realiza la actualización de velocidades y posiciones de todas las partículas, estableciendo cada vez las mejores partículas.

Primero inicializamos la funcion, estableciendo las variables globales de las particulas, sus velocidades, la posición de los mejores locales, y la posiciones de la mejor particula global.

Ademas establecemos los parámetros del algoritmo, en este caso:

- Weight (inercia): 0.8
- Coeficiente cognitivo (C1): 0.3
- Coeficiente social (C2): 0.4

```
def iteration(b):
    global count
    global X, X_lbest, X_gbest, V

    weight = 0.8
    C1 = 0.3
    C2 = 0.4
```

En cada iteración, realizaremos la impresión de la posición de la mejor particula global y su valor de aptitud.

```
count += 1
    print(count, "Best particle in:", X_gbest, " gbest: ",
rosenbrock(X_gbest))
```

La parte principal del algoritmo se realiza en el siguiente ciclo for, donde para cada partícula y para cada valor de dimensión, se realiza lo siguiente.

Para cada valor de dimensión, se generan dos números aleatorios entre 0 y 1, los cuales de darán un peso a los coeficientes cognitivo y social. Posteriormente, se calcula la velocidad siguiente la formula mostrada.

Por último, la velocidad calculada se suma al valor actual de dimensión, para actualizar la posición de la partícula.

Después de calcular todas las dimensiones de una partícula, se verifica si su valor de aptitud es el mejor local, por último, se verifica si la posición es la mejor global.

```
# Actualizar velocidad y posición de las partículas
   for I in range(n_particles):
       for J in range(n_dimensions):
           R1 = np.random.rand()
           R2 = np.random.rand()
           V[I][J] = (weight * V[I][J]
                      + C1 * R1 * (X_lbest[I][J] - X[I][J])
                      + C2 * R2 * (X_gbest[J] - X[I][J]))
           X[I][J] = X[I][J] + V[I][J]
       if rosenbrock(X[I]) < rosenbrock(X_lbest[I]):
           X_lbest[I] = 1 * X[I]
           if rosenbrock(X_lbest[I]) < rosenbrock(X_gbest):
               X_gbest = 1 * X_lbest[I]
```

Para la funcion de Rosenbrock en particular se utilizar funciones para graficar la funcion en 2D con un mapa de calor. Además, se muestran las partículas y un flecha que indica su velocidad y dirección.

Ejecución

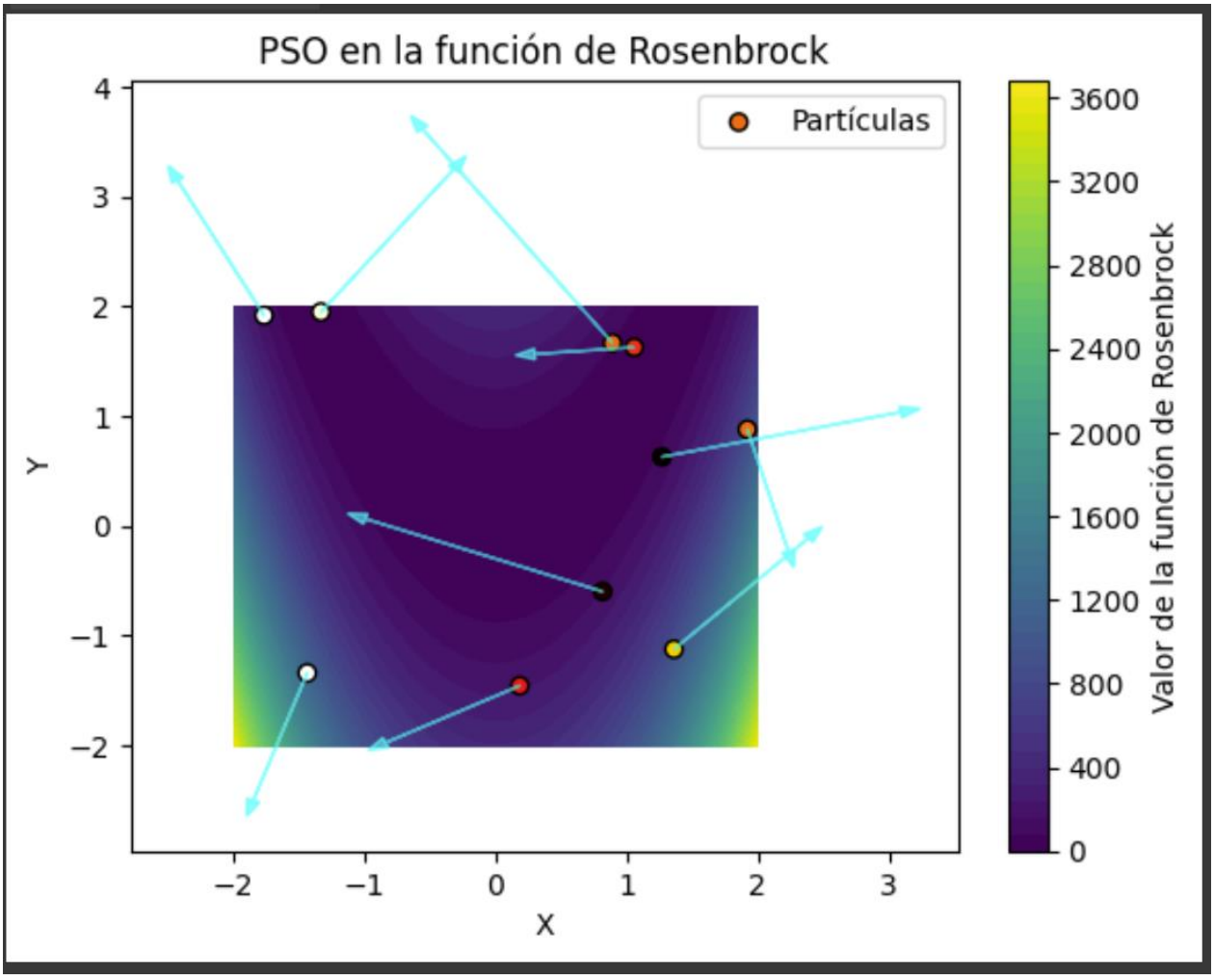

Ilustración 1. Población inicial de la optimización de la funcion de Rosenbrock.

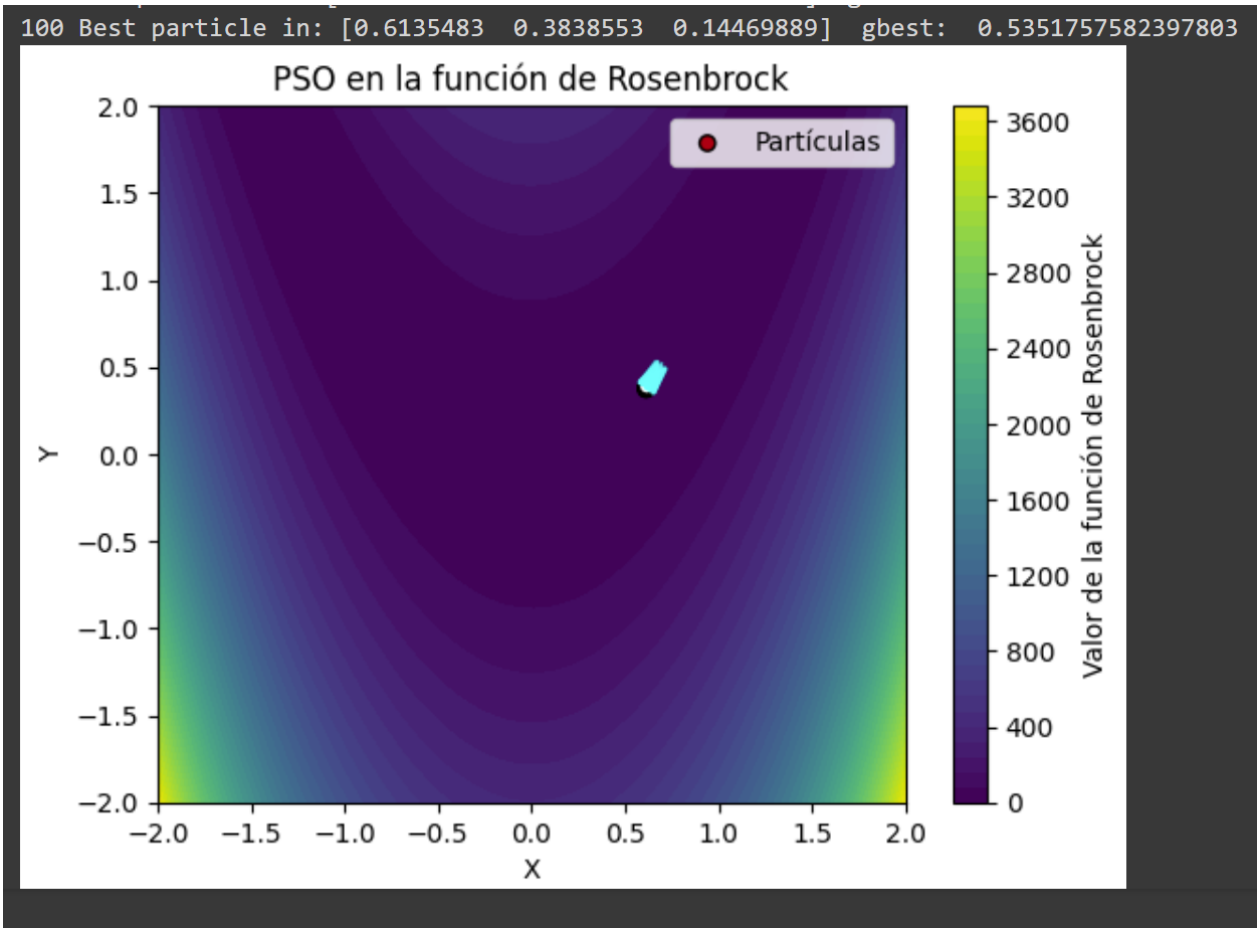

Ilustración 2. Iteración 100 de la optimización de la funcion de Rosenbrock.

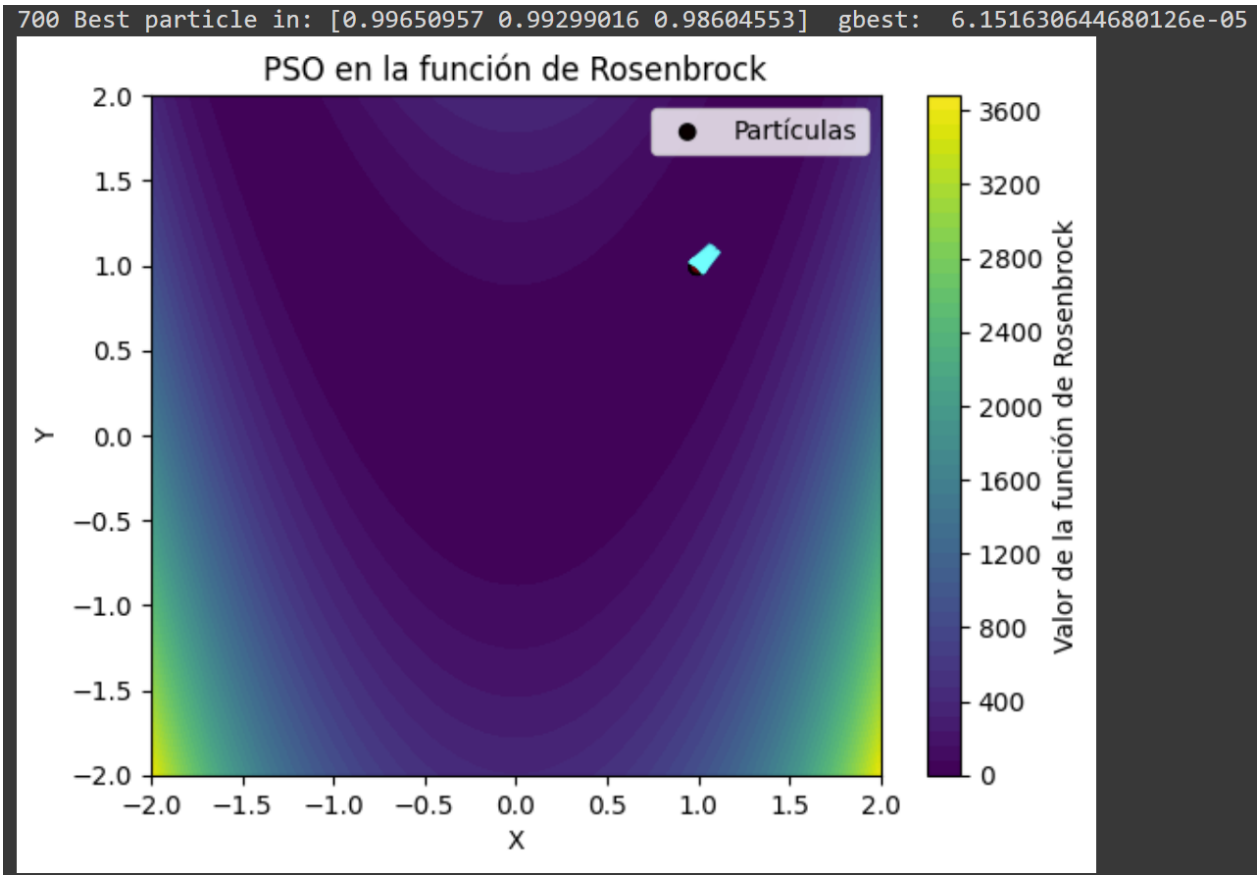

Ilustración 3. Iteración 700 de la optimización de la funcion de Rosenbrock.

Optimización de la funcion de Schwefel

Para realizar la optimización de este algoritmo se utilizó como base el código realizado anteriormente, y se realizaron las siguientes modificaciones.

Primero, se establecieron los límites del espacio de búsqueda de -500 a 500. Se establecieron 200 partículas, de 4 dimensiones cada una.

```
lower_limit = -500
upper_limit = 500

n_particles = 200
n_dimensions = 4
```

Ahora, la funcion objetivo es la funcion de Schwefel, en 4 dimensiones.

```
# Función de Schwefel en 4D
def schwefel(X):
    return 418.9829 * len(X) - sum(X * np.sin(np.sqrt(np.abs(X))))
```

Inicializamos las posiciones de las partículas, utilizando números aleatorios generados de manera uniforme y dentro de los rangos del espacio de búsqueda.

```
# Inicializar posiciones y velocidades de partículas
X = np.random.uniform(lower_limit, upper_limit, (n_particles,
n_dimensions))  # 4 dimensiones
V = np.random.uniform(-1, 1, (n_particles, n_dimensions))
```

Inicializamos los mejores valores de aptitud y sus posiciones.

```
X_lbest = np.copy(X)
X_gbest = np.copy(X_lbest[0])

for I in range(n_particles):
    if schwefel(X_lbest[I]) < schwefel(X_gbest):
        X_gbest = np.copy(X_lbest[I])

# Inicializar las mejores evaluaciones de fitness
fitness_lbest = np.array([schwefel(p) for p in X_lbest])
fitness_gbest = np.min(fitness_lbest)
```

Establecemos los siguientes parámetros del algoritmo.

- Inercia (weight): 0.5
- Coeficiente cognitivo (C1): 1.5
- Coeficiente social (C2): 1.5

```
    weight = 0.5
    C1 = 1.5
    C2 = 1.5
```

Se realizo una modificación en el ciclo for principal del algoritmo, donde ahora realizamos los cálculos de las velocidades y actualizaciones de posiciones de forma vectorial. Por lo que ahora se generan dos vectores de 4 dimensiones con números aleatorios para establecer los pesos de los coeficientes.

Además aplicamos límites al resultado de las posiciones de las partículas, esto para que no se salgan del espacio de búsqueda establecida.

```
for i in range(0, n_particles):
```

```
        r1 = np.random.rand(4)
        r2 = np.random.rand(4)
        cognitive_velocity = C1 * r1 * (X_lbest[i] - X[i])
        social_velocity = C2 * r2 * (X_gbest - X[i])
        V[i] = weight * V[i] + cognitive_velocity + social_velocity

        # Actualizar la posición
        X[i] += V[i]

        # Aplicar límites de búsqueda
        X[i] = np.clip(X[i], lower_limit, upper_limit)

        # Evaluar la nueva posición
        fitness = schwefel(X[i])

        # Actualizar la mejor posición local
        if fitness < fitness_lbest[i]:
            fitness_lbest[i] = fitness
            X_lbest[i] = X[i]

        # Actualizar la mejor posición global
        if fitness < fitness_gbest:
            fitness_gbest = fitness
            X_gbest = X[i]
```

En este caso no fue posible mostrar una representación grafica de la funcion y las partículas. Por lo que, solo se muestra la posición de población inicial creada. Y posteriormente para cada iteración solo mostramos a la mejor partícula y su valor de aptitud.

```
print(count, "Best particle in:", X_gbest, " gbest: ", schwefel(X_gbest))
```

```
print("Initial positions:", X)
```

Ejecución

```
Initial positions: [[ 1.78420648e+02 -1.38052408e+02 -4.21941941e+02  1.34359277e+02]
 [ 2.10958709e+02 -1.79505620e+02 -4.36864700e+02 -1.52263758e+02]
 [-2.36478795e+02 -1.55744584e+02  3.33686542e+02  2.20030688e+02]
 [ 1.63428130e+01  7.82293793e+01  1.29966194e+02 -8.23814841e+00]
 [ 3.29606416e+02  2.95432792e+02 -4.39028511e+01 -1.56081866e+02]
 [ 1.79484742e+02 -3.53393259e+02  4.40796551e+02 -3.75525042e+02]
 [-4.73091087e+02  4.91185565e+02  2.71703147e+02 -1.75403605e+02]
 [ 2.82615431e+02  1.99304019e+02 -1.04603166e+02 -6.46405749e+01]
 [-3.58514124e+02  4.13637336e+02 -3.45877714e+02 -1.26389518e+02]
 [-1.46522008e+02 -3.22608131e+02  1.60287750e+02 -1.10249672e+02]
 [-4.78388337e+02  4.62304658e+02 -2.21885679e+02 -4.17039485e+02]
 [-3.50873531e+02 -5.34251907e+01  8.39631295e+01  1.73706327e+02]
 [ 3.39208514e+02  2.20923702e+02 -2.68886213e+02  3.77786377e+02]
 [-2.83286229e+01  2.81934386e+01 -1.57591872e+02 -2.57310984e+02]
 [-1.96093283e+02  5.86019504e+01  1.60232670e+02 -3.93154023e+02]
 [-3.72246386e+02 -1.35469546e+02 -4.70475221e+02 -1.12882878e+02]
 [-4.06726148e+02  1.62055958e+02 -3.33792799e+02 -1.30083103e+02]
 [ 1.43761383e+02 -2.97617355e+02  2.70041077e+02  3.18089858e+02]
 [ 8.35971140e+01  1.43064092e+02 -3.84245869e+02 -4.50189145e+02]
 [ 3.05699310e+02 -2.91292535e+02  1.75434670e+02  1.04934102e+02]
 [ 2.63187359e+02  1.25986300e+02 -6.65429335e+01  4.77035849e+02]
 [ 1.38573832e+01  3.16117323e+02  3.67087555e+02 -3.11776555e+02]
 [ 3.71604722e+01 -2.32109317e+02  5.85343237e+00 -3.72733348e+02]
 [ 1.60910848e+02 -2.83512417e+02 -2.26369130e+02  2.91256895e+01]
 [-1.29087052e+02 -3.64813860e+02 -3.09510170e+02  2.39990511e+02]
 [-3.89479837e+02 -1.05791459e+01 -4.69654886e+02 -9.86392248e+01]
 [ 3.12458724e+02  4.86965755e+01  1.10313733e+02 -1.86618124e+02]
 [-2.74168050e+02 -2.15769438e+02  2.14626285e+02 -2.52808600e+02]
 [ 3.07025396e+02 -9.27280446e+01  2.70258595e+02  3.50549048e+02]
```

Ilustración 4. Parte de la población inicial de la optimización de la funcion de Schwefel.

```
24 Best particle in: [420.28863639 419.95361823 422.94243888 414.71197804]  gbest:  5.6037918834477277
25 Best particle in: [420.19438766 421.37410735 420.41924018 421.83904119]  gbest:  0.23012171328377917
26 Best particle in: [420.70082768 421.308504   420.99804483 421.67807544]  gbest:  0.08728829683445838
27 Best particle in: [421.46608934 420.62692953 421.22010315 420.56340874]  gbest:  0.07471051752600033
28 Best particle in: [421.68824896 421.9674784  420.61216842 420.20061008]  gbest:  0.2817662217257748
29 Best particle in: [420.78845374 421.60884698 421.00625936 421.01966871]  gbest:  0.056369747905137046
30 Best particle in: [420.53070621 421.07157017 421.10993769 420.82239089]  gbest:  0.03081103287786391
31 Best particle in: [420.79486013 421.15696258 420.6781381  420.94716434]  gbest:  0.019050533381005152
32 Best particle in: [420.99575618 421.17971426 420.70362042 421.10761405]  gbest:  0.017061724376844722
33 Best particle in: [420.86514244 421.13784424 420.73710324 420.78537161]  gbest:  0.016026594194272548
34 Best particle in: [421.07802144 421.14530309 420.78440672 420.85649466]  gbest:  0.011368868115823716
35 Best particle in: [421.02821078 421.04371522 420.94544491 420.99025057]  gbest:  0.0013331841344097484
36 Best particle in: [421.08173502 420.64409331 420.55546654 421.68102986]  gbest:  0.10054326446243067
37 Best particle in: [421.10479093 420.62665141 420.84424007 421.82941562]  gbest:  0.11260564998133304
38 Best particle in: [421.08222028 420.73781792 421.08065629 421.63516998]  gbest:  0.06603842687627548
39 Best particle in: [420.99148518 421.22675757 421.13256115 421.10976192]  gbest:  0.014412692897622037
40 Best particle in: [420.96542115 421.3685117  421.05407133 420.70924021]  gbest:  0.02963637584798562
41 Best particle in: [421.03369565 421.31309697 420.94764458 420.56733893]  gbest:  0.03593219088452315
42 Best particle in: [420.91865383 421.02775984 420.95056244 420.94102891]  gbest:  0.0009456503294131835
43 Best particle in: [420.97903211 420.9179656  420.99491136 420.92832938]  gbest:  0.0006821499937359476
44 Best particle in: [420.90190275 420.93172693 421.0708298  420.79746464]  gbest:  0.005804280501706671
45 Best particle in: [420.95470581 420.97501825 420.98651554 420.94618538]  gbest:  0.0001848177078045 41
46 Best particle in: [421.01883915 420.99920124 420.9138232  421.14144749]  gbest:  0.004628968795032051
47 Best particle in: [421.02675311 420.98260255 420.96089252 421.23547914]  gbest:  0.009485899741093817
48 Best particle in: [420.95505132 420.96776686 421.00807796 421.27849851]  gbest:  0.012378142381749058
49 Best particle in: [420.91905569 420.96771843 421.01853734 421.21488065]  gbest:  0.008320600406023004
50 Best particle in: [420.953694   420.96623313 420.95843293 420.99282994]  gbest:  0.0001669081771069613 8
51 Best particle in: [420.96930389 420.98303014 420.96181483 420.96098394]  gbest:  9.036018172992044e-05
52 Best particle in: [420.95959422 420.96956145 420.96763253 420.95954648]  gbest:  7.239980141093838e-05
53 Best particle in: [420.96072771 420.96825946 420.96871713 420.97322948]  gbest:  6.158979681458732e-05
```

Ilustración 5. Parte de la ejecución donde se encuentra la posición óptima para la funcion Schwefel.

3. CONCLUSIONES

En esta práctica, se utilizaron algoritmos de optimización por enjambre de partículas (PSO) para optimizar las funciones de Rosenbrock y Schwefel

La implementación permitió visualizar cómo las partículas convergen hacia las regiones óptimas en el espacio de búsqueda, adaptando sus posiciones en función de la experiencia individual y grupal.

Un aspecto importante observado fue que el PSO es altamente sensible a la configuración de sus parámetros, como la inercia y los coeficientes cognitivo y social, que afectan significativamente la velocidad de convergencia y la exploración del espacio. Ajustar estos valores en experimentos adicionales ayudó a equilibrar la exploración y explotación, mejorando la precisión en ambas funciones objetivo.

Una sugerencia para futuras prácticas sería experimentar con diferentes tamaños de población y agregar límites a la velocidad para evitar que las partículas sobrepasen el óptimo. También resultaría beneficioso incorporar representaciones gráficas adicionales, como trayectorias de partículas para observar cómo se realizan los ajustes.

4. BIBLIOGRAFÍA

[1] "A Gentle Introduction to Particle Swarm Optimization", *Machinelearningmastery.com*. [En línea]. Disponible en: https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/. [Consultado: 29-oct-2024].

[2] "Rosenbrock Function", *Sfu.ca*. [En línea]. Disponible en: https://www.sfu.ca/~ssurjano/rosen.html. [Consultado: 29-oct-2024].

[3] "Schwefel Function", *Sfu.ca*. [En línea]. Disponible en: https://www.sfu.ca/~ssurjano/schwef.html. [Consultado: 29-oct-2024].

[4] "Particle swarm optimization (PSO) - an overview", *GeeksforGeeks*, 22-abr-2021. [En línea]. Disponible en: https://www.geeksforgeeks.org/particle-swarm-optimization-pso-an-overview/. [Consultado: 29-oct-2024].