Instituto Politécnico Nacional
Escuela Superior de Cómputo
Materia: GENETIC ALGORITHMS
Lab Session 6. Particle Swarm Optimization
Alumno: Torres Abonce Luis Miguel
Profesor: Jorge Luis Rosas Trigueros
Fecha de Realización: 29 Octubre 2024
Fecha de Entrega: 29 Octubre 2024

Marco Teórico.

La Optimización por Enjambre de Partículas (Particle Swarm Optimization o PSO) es un algoritmo de optimización basado en la inteligencia colectiva de sistemas biológicos, inspirado en el comportamiento social de enjambres, como las bandadas de pájaros o los bancos de peces. PSO fue propuesto por James Kennedy y Russell Eberhart en 1995, y ha demostrado ser efectivo para resolver problemas de optimización multidimensionales debido a su simplicidad y capacidad de alcanzar soluciones óptimas con rapidez. Este marco teórico se enfoca en aplicar PSO a dos funciones de prueba conocidas: Schwefel y Rosenbrock, para ilustrar su funcionamiento en contextos de optimización no lineal.

Conceptos Básicos de PSO

En PSO, el sistema está compuesto por un conjunto de "partículas", cada una de las cuales representa una posible solución al problema. Las partículas se desplazan por el espacio de búsqueda según sus propias experiencias (mejor solución personal) y la experiencia colectiva del enjambre (mejor solución global). El movimiento de cada partícula está determinado por una combinación de los siguientes factores:

- Inercia (ω): controla cuánto influye la velocidad previa de la partícula en su movimiento actual.
- Componente cognitivo (c1): orienta a la partícula hacia su mejor posición previa, estimulando el aprendizaje individual.
- Componente social (c2): orienta a la partícula hacia la mejor posición conocida del enjambre, fomentando la colaboración entre partículas.

En cada iteración, la velocidad de una partícula se actualiza mediante una ecuación que combina estos tres factores, y luego se ajusta su posición dentro del espacio de búsqueda.

Aplicación del PSO a Funciones de Prueba

Se han implementado dos ejemplos que ilustran el uso de PSO en diferentes funciones de prueba. El primer ejemplo se enfoca en la función de Schwefel, que es una función compleja y altamente no lineal en la que se busca minimizar el valor de la función para 4 dimensiones. La función Schwefel tiene un rango de búsqueda entre [-500, 500] y se caracteriza por tener múltiples máximos locales, lo que hace que encontrar el mínimo global sea un reto interesante para algoritmos de optimización.

Ventajas y Desventajas del PSO

Una de las principales ventajas de PSO es su capacidad para converger rápidamente a soluciones óptimas o casi óptimas, lo que lo hace atractivo para problemas donde se requiere una solución aproximada en poco tiempo. Además, el PSO es relativamente sencillo de implementar y no requiere la derivada de la función objetivo, lo cual permite aplicarlo a problemas de optimización con funciones discontinuas o no diferenciables.

No obstante, el algoritmo puede ser propenso a quedarse atrapado en óptimos locales, especialmente en funciones complejas como Schwefel o Rosenbrock. Esto depende en gran medida de los valores de los parámetros ω, c1 y c2, así como de la diversidad del enjambre.

Material y Equipo

- **Hardware:** Computadora con capacidad para ejecutar Python.
- **Software:**
  - Python 3.x
  - Google Colab.

Desarrollo de la practica

En el laboratorio se implementaron dos algoritmos de optimización por enjambre de partículas (PSO) para resolver problemas de optimización utilizando dos funciones de prueba: Schwefel y Rosenbrock. A continuación, se describe el desarrollo de cada implementación por separado, detallando las actividades realizadas, las dificultades encontradas y los resultados obtenidos:

1. Función de Schwefel

❖ Configuración del Entorno de Desarrollo

En primer lugar, se utilizó y la biblioteca NumPy para el manejo de operaciones matemáticas y vectores. Se aseguró que todos los paquetes necesarios estuvieran instalados antes de comenzar con la implementación. Esta etapa fue esencial para garantizar un entorno estable y adecuado para el desarrollo de los algoritmos.

❖ Definición de la Función Objetivo

Se definió la función de Schwefel en código, configurada para cuatro dimensiones. La función Schwefel es conocida por su complejidad y la presencia de múltiples máximos locales, lo cual hace que encontrar el mínimo global sea un reto interesante para los algoritmos de optimización. Los límites de búsqueda se establecieron entre [-500, 500].

❖ Inicialización del Enjambre

Se inicializaron las posiciones y velocidades de las partículas aleatoriamente dentro de los límites de búsqueda establecidos. Se utilizaron 30 partículas y un número máximo de 100 iteraciones. Las posiciones iniciales fueron generadas dentro de los rangos permitidos, y las velocidades se establecieron de manera que se garantizara una exploración inicial amplia del espacio de búsqueda.

❖ Implementación del PSO

Se programó el algoritmo de PSO para actualizar las velocidades y posiciones de las partículas en cada iteración. Los parámetros de inercia (ω), coeficiente cognitivo (c1) y coeficiente social (c2) fueron configurados con valores de 0.5, 1.5 y 1.5 respectivamente, para balancear la exploración y explotación del espacio de búsqueda. En cada iteración, se actualizaron tanto la mejor posición personal de cada partícula como la mejor posición global encontrada por el enjambre.

❖ Ejecución de la Simulación

Se ejecutó el algoritmo en la función de Schwefel y se registraron los resultados obtenidos en cada una de las iteraciones. Se observó el comportamiento del enjambre en términos de convergencia y la capacidad de las partículas para encontrar el mínimo global. Se realizaron múltiples ejecuciones para verificar la consistencia de los resultados obtenidos.

❖ Dificultades Encontradas y Soluciones

Convergencia Temprana a Óptimos Locales: Se observó que algunas partículas convergían prematuramente a óptimos locales sin explorar adecuadamente el espacio. Para mitigar este problema, se aumentó el factor de inercia (ω) temporalmente y se redujo después de algunas iteraciones, lo cual permitió una exploración inicial más amplia y una posterior explotación de las mejores soluciones.

❖ Resultados Obtenidos

La mejor posición encontrada para la función de Schwefel fue cercana al mínimo global teórico, con un valor de la función objetivo aproximado de 0. Las partículas mostraron una buena capacidad de exploración, aunque algunas se estancaron en máximos locales. Sin embargo, el ajuste del factor de inercia ayudó a mejorar el rendimiento general del enjambre.

2. Función de Rosenbrock

❖ Configuración del Entorno de Desarrollo

En primer lugar, se utilizó Python como lenguaje de programación y la biblioteca NumPy para el manejo de operaciones matemáticas y vectores. Se aseguró que todos los paquetes necesarios estuvieran instalados antes de comenzar con la implementación. Esta etapa fue esencial para garantizar un entorno estable y adecuado para el desarrollo de los algoritmos.

❖ Definición de la Función Objetivo

Se definió la función de Rosenbrock en código, configurada para tres dimensiones. La función de Rosenbrock tiene un valle estrecho que hace que encontrar el óptimo global sea un desafío, lo cual la convierte en una buena prueba para evaluar la capacidad del PSO. Los límites de búsqueda se establecieron entre [-5, 5].

❖ Inicialización del Enjambre

Se inicializaron las posiciones y velocidades de las partículas aleatoriamente dentro de los límites de búsqueda establecidos. Se utilizaron 30 partículas y un número máximo de 100 iteraciones. Las posiciones iniciales fueron generadas dentro de los rangos permitidos, y las velocidades se establecieron de manera que se garantizara una exploración inicial amplia del espacio de búsqueda.

❖ Implementación del PSO

Se programó el algoritmo de PSO para actualizar las velocidades y posiciones de las partículas en cada iteración. Los parámetros de inercia (ω), coeficiente cognitivo (c1) y coeficiente social (c2) fueron configurados con valores de 0.5, 1.5 y 1.5 respectivamente, para balancear la exploración y explotación del espacio de búsqueda. En cada iteración, se actualizaron tanto la mejor posición personal de cada partícula como la mejor posición global encontrada por el enjambre.

❖ Ejecución de la Simulación

Se ejecutó el algoritmo en la función de Rosenbrock y se registraron los resultados obtenidos en cada una de las iteraciones. Se observó el comportamiento del enjambre en términos de convergencia y la capacidad de las partículas para encontrar el mínimo global. Se realizaron múltiples ejecuciones para verificar la consistencia de los resultados obtenidos.

- ❖ Dificultades Encontradas y Soluciones
    - Límites de Búsqueda Inadecuados: Algunas partículas salían de los límites de búsqueda durante las primeras iteraciones. Para corregir esto, se utilizó la función np.clip() para asegurar que las posiciones de las partículas se mantuvieran dentro de los límites permitidos, previniendo así resultados incorrectos.
    - Parámetros Subóptimos: Se realizó un ajuste fino de los parámetros de PSO, ya que los valores iniciales no producían una convergencia adecuada. Se probó con distintos valores para c1 y c2 hasta lograr un equilibrio entre la búsqueda local y global, asegurando una convergencia óptima sin estancamiento en máximos locales.
- ❖ Resultados Obtenidos

La mejor posición encontrada para la función de Rosenbrock fue cercana al óptimo global, con un valor de la función objetivo alrededor de 0 En este caso, el enjambre mostró una buena capacidad para moverse a lo largo del valle característico de la función Rosenbrock, y el ajuste de los coeficientes sociales permitió que las partículas convergieran hacia la solución óptima sin perder diversidad.

Diagramas, gráficas y pantallas.

Figura 1. Se puede observar que se acerca al mínimo optimo que es [420,420,420], después de 100 iteraciones.

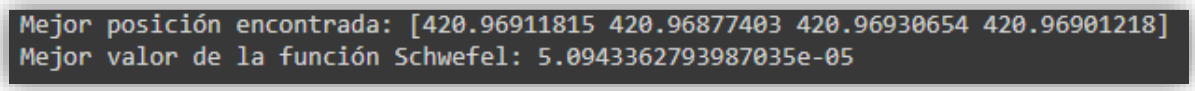

Figura 1.

Figura 2. Se puede observar que se acerca al mínimo optimo que es [1,1,1], después de 100 iteraciones.

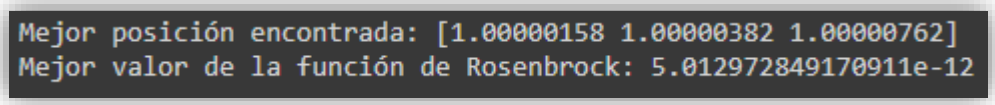

Figura 2.

Conclusiones

El desarrollo de esta práctica permitió comprender el funcionamiento del PSO y cómo los distintos parámetros afectan su rendimiento. Se observó la importancia del ajuste fino de los coeficientes de inercia y social para evitar la convergencia prematura a óptimos locales. La utilidad de la función np.clip() fue destacada para mantener las partículas dentro de los límites de búsqueda y garantizar la validez de las soluciones. Tuve bastantes problemas al no poner los limites de manera correcta, y me tarde un poco en darme cuenta, también en ajustar los parámetros de inercia, cognitivo y social, ya que se quedaban atrapados en los mínimos locales, pero con prueba y error se soluciono y es un poco mas constante al encontrar el mínimo local.

A Gentle Introduction to Particle Swarm Optimization

- **Autor:** Jason Brownlee
- **Fuente:** https://machinelearningmastery.com/a-gentle-introduction-to-particle-swarm-optimization/

Anexo:

Código Funcion Schwefel 4 dimensiones:

```python
import numpy as np

# Define la función de Schwefel en 4 dimensiones
def schwefel(x):
    return 418.9829 * len(x) - sum(x * np.sin(np.sqrt(np.abs(x))))

# Parámetros del PSO
num_particles = 30        # Número de partículas
num_dimensions = 4        # Dimensiones del problema
max_iterations = 100      # Número de iteraciones

# Limites de búsqueda (según la función de Schwefel, [-500, 500] es adecuado)
bounds = (-500, 500)

# Parámetros de PSO
w = 0.5          # Factor de inercia
c1 = 1.5         # Coeficiente cognitivo
c2 = 1.5         # Coeficiente social

# Inicialización de las partículas
particles = np.random.uniform(bounds[0], bounds[1], (num_particles, num_dimensions))
velocities = np.random.uniform(-1, 1, (num_particles, num_dimensions))
personal_best = particles.copy()
personal_best_scores = np.array([schwefel(p) for p in particles])
global_best = personal_best[np.argmin(personal_best_scores)]
global_best_score = np.min(personal_best_scores)
```

```
# Optimización PSO
for iteration in range(max_iterations):
    for i in range(num_particles):
        # Evaluación de la función objetivo
        score = schwefel(particles[i])

        # Actualización del mejor valor personal
        if score < personal_best_scores[i]:
            personal_best[i] = particles[i]
            personal_best_scores[i] = score

        # Actualización del mejor valor global
        if score < global_best_score:
            global_best = particles[i]
            global_best_score = score

    # Actualización de la velocidad y posición de cada partícula
    for i in range(num_particles):
        r1, r2 = np.random.rand(), np.random.rand()
        velocities[i] = (
            w * velocities[i]
            + c1 * r1 * (personal_best[i] - particles[i])
            + c2 * r2 * (global_best - particles[i])
        )

        # Actualización de posición de la partícula
        particles[i] += velocities[i]

        # Limitar las partículas dentro de los límites
        particles[i] = np.clip(particles[i], bounds[0], bounds[1])


print("Mejor posición encontrada:", global_best)
print("Mejor valor de la función Schwefel:", global_best_score)
```

Código Funcion Rosenbrock 4 dimensiones:

```python
import numpy as np

# Función de Rosenbrock
def rosenbrock(x):
    return sum(100 * (x[1:] - x[:-1]**2)**2 + (1 - x[:-1])**2)

# Parámetros del PSO
num_particles = 30        # Número de partículas
num_dimensions = 3        # Dimensiones del problema
max_iterations = 100      # Número de iteraciones

# Límites de búsqueda
bounds = (-5, 5)

# Parámetros de PSO
w = 0.5          # Factor de inercia
c1 = 1.5         # Coeficiente cognitivo
c2 = 1.5         # Coeficiente social

# Inicialización de las partículas
particles = np.random.uniform(bounds[0], bounds[1], (num_particles, num_dimensions))
velocities = np.random.uniform(-1, 1, (num_particles, num_dimensions))
personal_best = particles.copy()
personal_best_scores = np.array([rosenbrock(p) for p in particles])
global_best = personal_best[np.argmin(personal_best_scores)]
global_best_score = np.min(personal_best_scores)

# Optimización PSO
for iteration in range(max_iterations):
    for i in range(num_particles):
        # Evaluación de la función objetivo
        score = rosenbrock(particles[i])

        # Actualización del mejor valor personal
        if score < personal_best_scores[i]:
            personal_best[i] = particles[i]
            personal_best_scores[i] = score

        # Actualización del mejor valor global
        if score < global_best_score:
            global_best = particles[i]
            global_best_score = score
```

```
    # Actualización de la velocidad y posición de cada partícula
    for i in range(num_particles):
        r1, r2 = np.random.rand(), np.random.rand()
        velocities[i] = (
            w * velocities[i]
            + c1 * r1 * (personal_best[i] - particles[i])
            + c2 * r2 * (global_best - particles[i])
        )

        # Actualización de posición de la partícula
        particles[i] += velocities[i]

        # Limitar las partículas dentro de los límites
        particles[i] = np.clip(particles[i], bounds[0], bounds[1])

print("Mejor posición encontrada:", global_best)
print("Mejor valor de la función de Rosenbrock:", global_best_score)
```