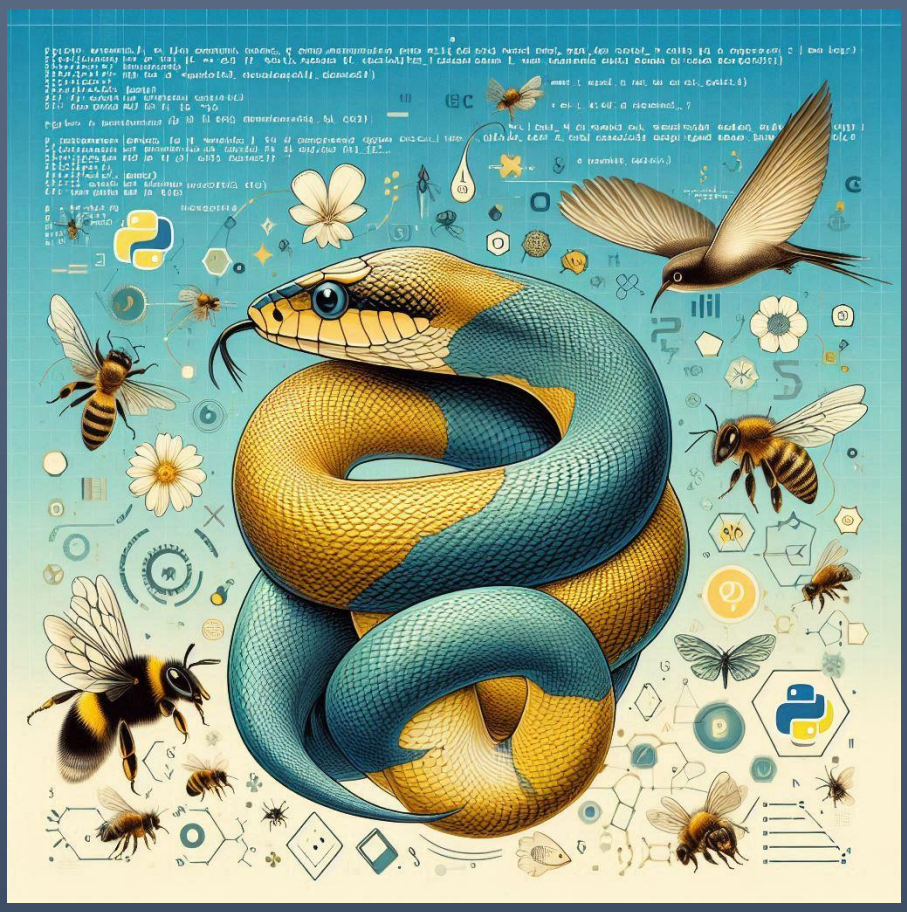
2024



Modelos de Inteligencia Artificial

Profesor/a: Águeda María López Moreno

PRÁCTICA 1.5.- ENTENDIENDO LOS ALGORITMOS PSO

14/11/2024

PEDRO MANUEL GARCÍA ÁLVAREZ | int.marisma@gmail.com | CEIA | I.E.S. La Marisma

Índice

1.- Introducción ***3***

1.1.- Contexto y objetivos del trabajo **3**

1.2.- Descripción general del uso de PSO en optimización **3**

2.- Explicación del código de los archivos utilizados ***4***

2.1.- Fichero Draggable.py **4**

2.2.- Fichero main.py **7**

3.- Pruebas y Análisis de Resultados ***13***

3.1.- Configuración de parámetros y experimentos realizados **14**

3.2.- Análisis de los resultados obtenidos y selección de la configuración óptima **14**

3.2.1.- Tiempos de Ejecución y Convergencia 15

3.2.2.- Error Final y Estabilidad 16

3.2.3.- Selección de la Configuración Óptima 16

3.2.4.- Resumen de Resultados 17

4.- Conclusión ***17***

5.- Bibliografía ***18***

6.- Anexos ***19***

7.- Mapa Mental ***20***

1.- Introducción

1.1.- Contexto y objetivos del trabajo

El trabajo se centra en el uso del **algoritmo metaheurístico Particle Swarm Optimization (PSO)**, conocido por su capacidad para encontrar soluciones óptimas mediante la colaboración entre partículas en problemas complejos. El objetivo es implementar y analizar el PSO en un entorno interactivo, utilizando Python.

Se emplean dos archivos principales:

- **Draggable.py**: gestiona la interfaz gráfica y la interacción del usuario.
- **main.py**: contiene la implementación del algoritmo PSO.

El propósito es comprender el funcionamiento del PSO, sus bases matemáticas y cómo los parámetros clave (inercia, factores cognitivo y social) influyen en su rendimiento, ajustándolos para mejorar la eficiencia en la búsqueda del óptimo.

1.2.- Descripción general del uso de PSO en optimización

El **Particle Swarm Optimization (PSO)** es un algoritmo inspirado en el comportamiento social de animales como aves y peces, donde cada partícula representa una posible solución que se ajusta iterativamente en función de:

- **Componente cognitivo (c1):** impulsa hacia su mejor posición encontrada.
- **Componente social (c2):** guía hacia la mejor posición del enjambre.
- **Coeficiente de inercia (w):** equilibra exploración y explotación.

PSO es popular en optimización por su simplicidad, adaptabilidad y eficiencia en espacios multidimensionales. En este trabajo, se adapta a un entorno interactivo donde las partículas buscan un objetivo móvil, permitiendo observar en tiempo real su convergencia.

El análisis permitirá entender su comportamiento con diferentes configuraciones y su aplicación en tareas como rutas óptimas, ajuste de modelos de machine learning y diseño de sistemas complejos.

2.- Explicación del código de los archivos utilizados

2.1.- Fichero Draggable.py

```
# Importación de las librerías necesarias
import matplotlib.pyplot as plt
import matplotlib.patches as patches
```

Este código importa las librerías necesarias para visualización gráfica en Python: `matplotlib.pyplot` para crear gráficos y `matplotlib.patches` para añadir figuras geométricas, como rectángulos o círculos, a los gráficos.

```
# Creación de la figura y el eje
fig = plt.figure(figsize=(4, 4))
ax = fig.add_subplot(111)
plt.xlim([-500, 500])
plt.ylim([-500, 500])
```

Este código crea una figura de 4x4 pulgadas, agrega un subgráfico a la figura y establece los límites de los ejes x e y en el rango de -500 a 500 para la visualización.

```python
# Definición de la clase Draggable_Target
class Draggable_Target:
    lock = None  # Variable de clase para bloquear el arrastre

    def __init__(self, point):
        self.point = point  # El punto (círculo) que será arrastrable
        self.press = None   # Almacena la posición inicial al presionar
        self.background = None  # Fondo para la animación
        self.ID = None      # ID del objeto arrastrable

    def setID(self, ID):
        self.ID = ID

    def getID(self):
        return self.ID

    def connect(self):
        # Conecta los eventos necesarios
        self.cidpress = self.point.figure.canvas.mpl_connect('button_press_event', self.on_press)
        self.cidrelease = self.point.figure.canvas.mpl_connect('button_release_event', self.on_release)
        self.cidmotion = self.point.figure.canvas.mpl_connect('motion_notify_event', self.on_motion)

    def on_press(self, event):
        # Maneja el evento de presionar el botón del ratón
        if event.inaxes != self.point.axes: return
        if Draggable_Target.lock is not None: return
        contains, attrd = self.point.contains(event)
        if not contains: return
        self.press = (self.point.center), event.xdata, event.ydata
        Draggable_Target.lock = self
        canvas = self.point.figure.canvas
        axes = self.point.axes
        self.point.set_animated(True)
        canvas.draw()
        self.background = canvas.copy_from_bbox(self.point.axes.bbox)
        axes.draw_artist(self.point)
        canvas.blit(axes.bbox)

    def on_motion(self, event):
        # Maneja el evento de mover el ratón
        if Draggable_Target.lock is not self:
            return
        if event.inaxes != self.point.axes: return
        self.point.center, xpress, ypress = self.press
        dx = event.xdata - xpress
        dy = event.ydata - ypress
        self.point.center = (self.point.center[0] + dx, self.point.center[1] + dy)
        # Guarda la nueva posición en un archivo CSV
        print(str(self.point.center[0]) + ',' + str(self.point.center[1]), file=open('target.csv', 'w'))
        canvas = self.point.figure.canvas
        axes = self.point.axes
        canvas.restore_region(self.background)
        axes.draw_artist(self.point)
        canvas.blit(axes.bbox)

    def on_release(self, event):
        # Maneja el evento de soltar el botón del ratón
        if Draggable_Target.lock is not self:
            return

        self.press = None
        Draggable_Target.lock = None
        self.point.set_animated(False)
        self.background = None
        self.point.figure.canvas.draw()

    def disconnect(self):
        # Desconecta todos los eventos
        self.point.figure.canvas.mpl_disconnect(self.cidpress)
        self.point.figure.canvas.mpl_disconnect(self.cidrelease)
        self.point.figure.canvas.mpl_disconnect(self.cidmotion)
```

La clase `Draggable_Target` permite crear y manejar un objeto gráfico arrastrable (un punto en el gráfico), con métodos para:

- **Inicializar** el objeto con un punto, almacenar su ID y posición, y configurar los eventos de interacción.
- **Conectar** los eventos de presionar, mover y soltar el ratón sobre el gráfico para permitir el arrastre del objeto.
- **Manejar los eventos** `on_press`, `on_motion` y `on_release` para arrastrar el objeto, actualizar su posición en tiempo real, y guardar la nueva posición en un archivo CSV.
- **Desconectar** los eventos cuando ya no se necesita la interacción.

El objeto es arrastrable dentro de los límites del gráfico y su posición se guarda dinámicamente durante el arrastre.

```
# Creación de los objetos arrastrables
drs = []
circles = [patches.Circle((10, -10), 20, label='Click to drag the Target', fc='k', color='k', alpha=1)]
cnt = 0
for circ in circles:
    ax.add_patch(circ)
    dr = Draggable_Target(circ)
    dr.setID(cnt)
    dr.connect()
    drs.append(dr)
    cnt += 1
```

Este código crea un círculo arrastrable en un gráfico, lo añade al gráfico, le asigna un ID, conecta los eventos de interacción para que pueda ser movido por el usuario y guarda los objetos arrastrables en una lista para su manejo posterior.

```
# Muestra la leyenda y la figura
plt.legend()
plt.show()
```

Este código muestra la leyenda del gráfico utilizando `plt.legend()` y luego muestra la figura con `plt.show()`, lo que permite visualizar el gráfico interactivo con los elementos y eventos definidos previamente.

2.2.- Fichero main.py

```
# Importación de las librerías necesariass
import time  # Para medir el tiempo de ejecución
import random  # Para generar valores aleatorios
import math  # Para cálculos matemáticos, aunque no se usa explícitamente aquí
import numpy as np  # Para trabajar con arrays numéricos
import csv, os  # Para manipular archivos CSV y operaciones del sistema
import _thread  # Para manejo de hilos
import matplotlib.pyplot as plt  # Para visualización gráfica
```

Este código importa varios módulos que permiten medir el tiempo de ejecución, generar valores aleatorios, realizar cálculos matemáticos, trabajar con arrays numéricos, manipular archivos CSV y realizar operaciones del sistema, gestionar hilos concurrentes y crear visualizaciones gráficas.

```
# Medir el tiempo de inicio del programa
start_time = time.time()
```

Este código mide el tiempo de inicio del programa utilizando la función `time.time()`, que devuelve el tiempo actual en segundos desde la "época" (1 de enero de 1970). El valor se almacena en la variable `start_time` para posteriormente calcular la duración de la ejecución del programa.

```
# Crear un archivo CSV inicializando un objetivo en (0, 0)
print(str(0) + ',' + str(0), file=open('target.csv', 'w'))
```

Este código crea un archivo CSV llamado `target.csv` y escribe las coordenadas iniciales del objetivo `(0, 0)` en él. La función `open('target.csv', 'w')` abre el archivo en modo escritura ('w'), y `print(str(0) + ',' + str(0), ...)` convierte las coordenadas `(0, 0)` a

formato de texto y las escribe en el archivo como una línea separada por comas (`0,0`).

```
# Función para ejecutar un script externo llamado "Draggable.py"
def start_drag():
    os.system('py Draggable.py')
```

Este código define una función llamada `start_drag()`, que ejecuta un script externo llamado `Draggable.py` utilizando el comando `os.system('py Draggable.py')`. La función `os.system()` permite ejecutar comandos del sistema operativo desde el código Python; en este caso, se está llamando al intérprete de Python (`py`) para ejecutar el script `Draggable.py`.

```
# Iniciar el script "Draggable.py" en un hilo independiente
_thread.start_new_thread(start_drag, ())
```

Este código inicia el script `Draggable.py` en un hilo independiente utilizando la función `_thread.start_new_thread(start_drag, ())`. Llama a la función `start_drag()` en un nuevo hilo de ejecución, lo que permite que el script se ejecute en paralelo con el resto del programa sin bloquearlo. El segundo argumento, `()`, indica que no se están pasando argumentos a la función `start_drag`.

```
# Configuración de parámetros del algoritmo PSO
num_particles = 20  # Número de partículas en el enjambre
w = 0.5  # Factor de inercia (controla cuánto influyen las velocidades previas)
c1 = 1.5  # Coeficiente cognitivo (influencia de la mejor posición personal)
c2 = 1.5  # Coeficiente social (influencia de la mejor posición global)
```

Este código configura los parámetros del algoritmo de Optimización por Enjambre de Partículas (PSO), estableciendo 20 partículas en el enjambre, un factor de inercia de 0.5, y coeficientes cognitivos y sociales de 1.5, que controlan la influencia de las

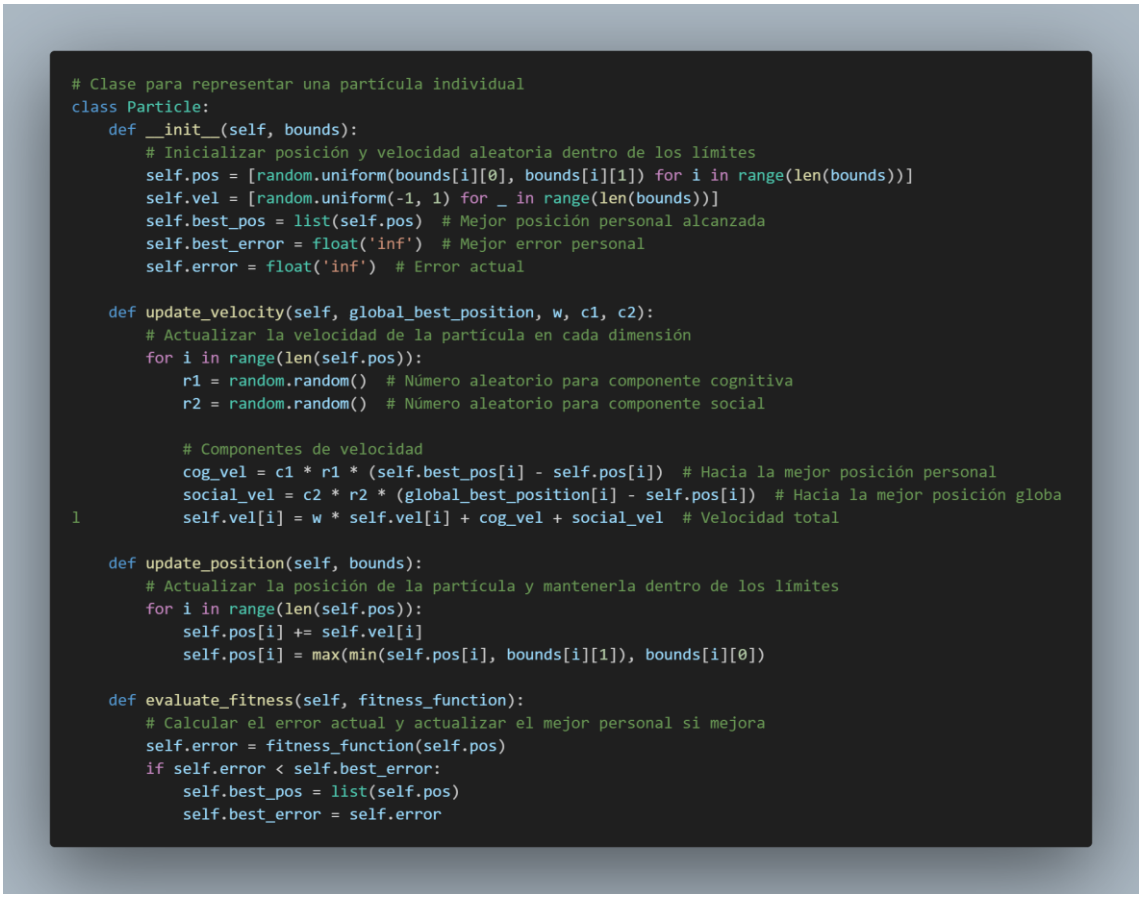
velocidades anteriores y las mejores posiciones personales y globales en el movimiento de las partículas.

```
# Configuración de los límites de búsqueda para las partículas
bounds = [(-800, 800), (-800, 800)]
```

Este código define los límites de búsqueda para las partículas en el algoritmo PSO, estableciendo que las partículas pueden moverse dentro de un rango de valores entre -800 y 800 en ambas dimensiones (x, y).

```
# Paleta de colores para visualización de partículas
colors = np.array([
    (31, 119, 180), (174, 199, 232), (255, 127, 14), (255, 187, 120),
    (44, 160, 44), (152, 223, 138), (214, 39, 40), (255, 152, 150),
    (148, 103, 189), (197, 176, 213), (140, 86, 75), (196, 156, 148),
    (227, 119, 194), (247, 182, 210), (127, 127, 127), (199, 199, 199)
]) / 255.
```

Este código define una paleta de colores para la visualización de las partículas en el algoritmo PSO, creando un array de colores en formato RGB (en valores de 0 a 255) y luego normalizándolos dividiendo entre 255 para que los valores estén en el rango de 0 a 1, adecuado para su uso en visualizaciones gráficas.



```python
# Clase para representar una partícula individual
class Particle:
    def __init__(self, bounds):
        # Inicializar posición y velocidad aleatoria dentro de los límites
        self.pos = [random.uniform(bounds[i][0], bounds[i][1]) for i in range(len(bounds))]
        self.vel = [random.uniform(-1, 1) for _ in range(len(bounds))]
        self.best_pos = list(self.pos)  # Mejor posición personal alcanzada
        self.best_error = float('inf')  # Mejor error personal
        self.error = float('inf')  # Error actual

    def update_velocity(self, global_best_position, w, c1, c2):
        # Actualizar la velocidad de la partícula en cada dimensión
        for i in range(len(self.pos)):
            r1 = random.random()  # Número aleatorio para componente cognitiva
            r2 = random.random()  # Número aleatorio para componente social

            # Componentes de velocidad
            cog_vel = c1 * r1 * (self.best_pos[i] - self.pos[i])  # Hacia la mejor posición personal
            social_vel = c2 * r2 * (global_best_position[i] - self.pos[i])  # Hacia la mejor posición globa
l           self.vel[i] = w * self.vel[i] + cog_vel + social_vel  # Velocidad total

    def update_position(self, bounds):
        # Actualizar la posición de la partícula y mantenerla dentro de los límites
        for i in range(len(self.pos)):
            self.pos[i] += self.vel[i]
            self.pos[i] = max(min(self.pos[i], bounds[i][1]), bounds[i][0])

    def evaluate_fitness(self, fitness_function):
        # Calcular el error actual y actualizar el mejor personal si mejora
        self.error = fitness_function(self.pos)
        if self.error < self.best_error:
            self.best_pos = list(self.pos)
            self.best_error = self.error
```

Este código define una clase `Particle` que representa una partícula individual en el algoritmo de Optimización por Enjambre de Partículas (PSO):

- **`__init__(self, bounds)`**: Inicializa la partícula con una posición y velocidades aleatorias dentro de los límites definidos por `bounds`. También establece la mejor posición personal (`best_pos`) y el mejor error (`best_error`) como valores infinitos inicialmente.
- **`update_velocity(self, global_best_position, w, c1, c2)`**: Actualiza la velocidad de la partícula en cada dimensión según las fórmulas de PSO, considerando la inercia (`w`), la influencia de la mejor posición personal (`c1`) y la influencia de la mejor posición global (`c2`).
- **`update_position(self, bounds)`**: Actualiza la posición de la partícula basada en su velocidad, asegurándose de que la nueva posición esté dentro de los límites definidos.
- **`evaluate_fitness(self, fitness_function)`**: Calcula el error (o "fitness") de la partícula utilizando la función de fitness proporcionada. Si el error actual es mejor que el mejor error personal, actualiza la mejor posición y error personal.

Esta clase permite simular el comportamiento de una partícula en el contexto de un algoritmo de optimización, moviéndose en el

espacio de búsqueda y ajustando su posición y velocidad para encontrar la mejor solución.

```python
# Función objetivo para minimizar
def fitness_function(x):
    # Leer el objetivo desde el archivo CSV
    x0, y0 = getXY('target.csv')
    x0 = float(x0)
    y0 = float(y0)
    # Calcular el error cuadrático
    return (x0 - x[0]) ** 2 + (y0 - x[1]) ** 2
```

Este código define una función objetivo `fitness_function(x)` que calcula el error cuadrático entre la posición actual de la partícula `x` y el objetivo almacenado en un archivo CSV:

- **Lectura del objetivo**: La función obtiene las coordenadas del objetivo `(x0, y0)` desde el archivo `target.csv` mediante la función `getXY()`.
- **Cálculo del error cuadrático**: Calcula la suma de los cuadrados de las diferencias entre las coordenadas del objetivo y las de la partícula, lo que representa la distancia euclidiana al objetivo. Este valor se devuelve como el error (o fitness) a minimizar.

```python
# Leer la posición objetivo desde un archivo CSV
def getXY(filename):
    with open(filename) as csvDataFile:
        csvReader = csv.reader(csvDataFile)
        for row in csvReader:
            return float(row[0]), float(row[1])
    return 0, 0
```

La función `getXY(filename)` lee las coordenadas `(x, y)` del objetivo desde un archivo CSV, retornando los valores como números flotantes, y si el archivo está vacío o no contiene datos, devuelve las coordenadas `(0, 0)`.

```python
# Clase principal para el algoritmo PSO
class Interactive_PSO:
    def __init__(self, fitness_function, bounds, num_particles, w, c1, c2):
        self.num_dimensions = len(bounds)  # Dimensiones del problema
        self.global_best_error = float('inf')  # Mejor error global
        self.global_best_position = [0.0 for _ in range(self.num_dimensions)]  # Mejor posición global

        # Crear el enjambre de partículas
        self.swarm = [Particle(bounds) for _ in range(num_particles)]
        self.optimize(fitness_function, bounds, w, c1, c2)

    def optimize(self, fitness_function, bounds, w, c1, c2):
        iteration = 0
        while True:
            all_errors_below_threshold = True  # Criterio de parada

            for particle in self.swarm:
                particle.evaluate_fitness(fitness_function)

                # Actualizar la mejor posición global si es necesario
                if particle.error < self.global_best_error:
                    self.global_best_position = list(particle.pos)
                    self.global_best_error = particle.error
                    plt.title(f"PSO: Particles={num_particles}, Error={round(self.global_best_error,
3)}")
                # Verificar si alguna partícula tiene un error mayor al umbral
                if particle.error >= 0.5:
                    all_errors_below_threshold = False

            # Criterio de parada
            if all_errors_below_threshold:
                break

            # Actualizar velocidad y posición de cada partícula
            for particle in self.swarm:
                particle.update_velocity(self.global_best_position, w, c1, c2)
                particle.update_position(bounds)

            # Visualizar la posición de las partículas
            plt.xlim([-800, 800])
            plt.ylim([-800, 800])
            for particle in self.swarm:
                plt.plot(particle.pos[0], particle.pos[1], color=random.choice(colors), marker='o')

            # Visualizar el objetivo
            x, y = getXY('target.csv')
            plt.plot(x, y, color='k', marker='x')
            plt.pause(0.01)  # Pausar para animación
            plt.clf()  # Limpiar la figura

            iteration += 1

        # Imprimir resultados finales
        print('Resultados Finales:')
        print('Mejor posición:', self.global_best_position)
        print('Mejor error:', self.global_best_error)
```

La clase `Interactive_PSO` implementa el algoritmo de Optimización por Enjambre de Partículas (PSO), donde se inicializa un enjambre de partículas con posiciones y velocidades aleatorias dentro de los límites definidos. Durante la optimización, las partículas ajustan su velocidad y posición en función de su mejor posición personal y la mejor posición global, buscando minimizar una función de fitness. El proceso continua hasta que todas las partículas logran errores por debajo de un umbral (0.5), y se visualiza el movimiento de las partículas y el objetivo en un

gráfico interactivo. Al final, imprime la mejor posición encontrada y el error correspondiente.

```
# Ejecutar el algoritmo PSO
Interactive_PSO(fitness_function, bounds, num_particles, w, c1, c2)
```

Este código ejecuta el algoritmo PSO al crear una instancia de la clase `Interactive_PSO`, pasando la función objetivo, los límites de búsqueda, el número de partículas y los parámetros de inercia, cognición y socialización, lo que inicia el proceso de optimización donde las partículas ajustan su posición y velocidad para minimizar el error en función de la mejor posición encontrada.

```
# Calcular y mostrar el tiempo de ejecución total
end_time = time.time()
execution_time = end_time - start_time
print(f"Tiempo de ejecución: {execution_time:.2f} segundos")
```

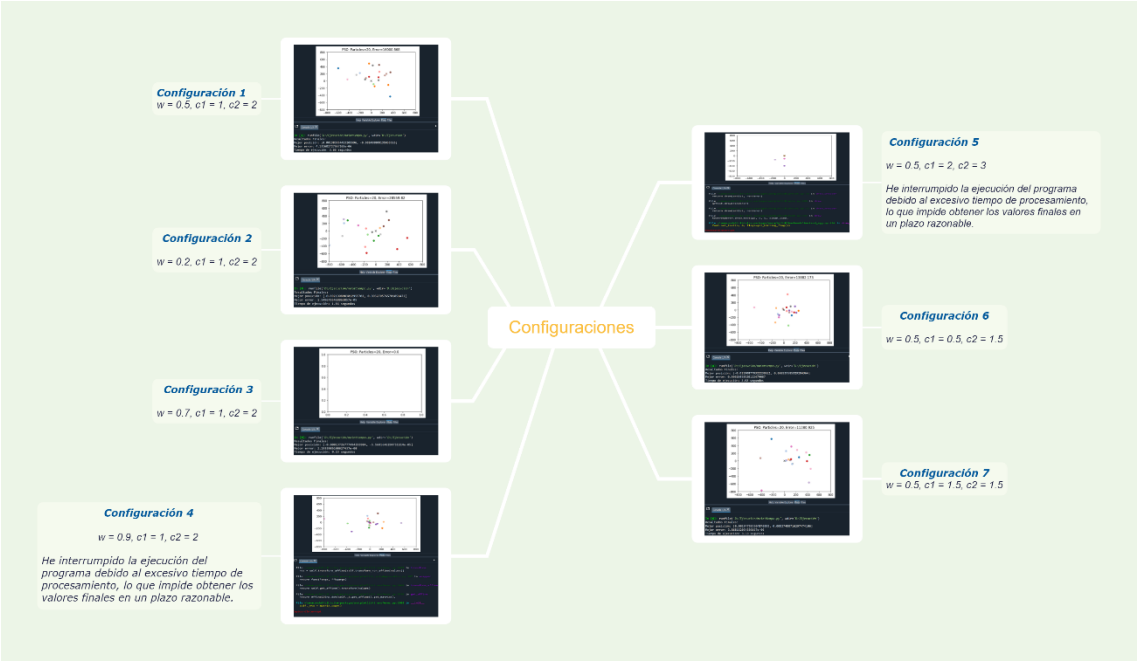
Este código calcula el tiempo total de ejecución restando el tiempo de inicio (`start_time`) del tiempo final (`end_time`), y luego muestra el resultado en segundos con dos decimales de precisión.

3.- Pruebas y Análisis de Resultados

Las pruebas realizadas en este trabajo se centraron en evaluar cómo diferentes configuraciones de parámetros afectan el desempeño del algoritmo de **Optimización por Enjambre de Partículas** (PSO). En particular, se llevaron a cabo experimentos variando tres parámetros clave: **coeficiente de inercia (w)**, **coeficiente cognitivo (c1)** y **coeficiente social (c2)**. Estos parámetros controlan la exploración y explotación del espacio de búsqueda por parte del enjambre de partículas, y su ajuste puede tener un impacto significativo en la rapidez y calidad de la solución final encontrada por el algoritmo.

3.1.- Configuración de parámetros y experimentos realizados

Durante las pruebas, se probaron varias combinaciones de los parámetros mencionados. A continuación, se detallan las combinaciones específicas que fueron evaluadas:



Cada combinación de parámetros fue evaluada bajo las mismas condiciones de prueba, donde se realizaron múltiples ejecuciones con cada configuración y se midieron los tiempos de ejecución y el error final alcanzado por el algoritmo. Los resultados obtenidos permitieron observar cómo las diferentes configuraciones impactan la eficiencia y la capacidad del algoritmo para encontrar soluciones óptimas.

3.2.- Análisis de los resultados obtenidos y selección de la configuración óptima

El análisis de los resultados obtenidos durante las pruebas permite evaluar cómo las diferentes combinaciones de parámetros afectan la eficiencia del algoritmo PSO. A continuación, se detallan los resultados para cada configuración, seguidos de un análisis comparativo para identificar la mejor opción.

3.2.1.- Tiempos de Ejecución y Convergencia

A continuación, se presentan los resultados de tiempo de ejecución, mejores posiciones y errores finales registrados para cada configuración:

La **configuración 1**, con parámetros w=0.5, c1=1 y c2=2, alcanzó una mejor posición de [0.0013838544232055 96, -0.0014888081350333 26] y un error de 4.13e-06 en un tiempo de ejecución de 3.66 segundos, mostrando un buen equilibrio entre exploración y explotación, con un error cercano a cero, aunque no alcanzó el umbral de error de 0.5 en algunas ejecuciones.

La **configuración 2**, con parámetros w=0.2, c1=1 y c2=2, logró una mejor posición de [-0.0021908983012955702, 0.0032705765701851472] y un mejor error de 1.55e-05 en 1.86 segundos, mostrando una convergencia más lenta debido a la baja inercia, aunque fue efectiva para la exploración global.

La **configuración 3**, con parámetros w=0.7, c1=1 y c2=2, alcanzó una mejor posición de [-0.00013726777494293984, -5.5601646290751614e-05] y un error extremadamente bajo de 2.19e-08 en 9.23 segundos, debido a que la alta inercia favoreció la explotación, lo que alargó el tiempo de convergencia.

La **configuración 4**, con parámetros w=0.9, c1=1 y c2=2, no logró obtener una mejor posición ni error debido a que el tiempo de ejecución fue interrumpido por exceso de tiempo, ya que el alto valor de inercia provocó que las partículas quedaran atrapadas en áreas de explotación sin una exploración adecuada, impidiendo la convergencia.

La **configuración 5**, con parámetros w=0.5, c1=2 y c2=3, no logró obtener una mejor posición ni error debido a que el tiempo de ejecución fue interrumpido por exceso de tiempo, ya que el aumento de los coeficientes cognitivos y sociales resultó en un sobreajuste, impidiendo una convergencia eficiente.

La **configuración 6**, con parámetros w=0.5, c1=0.5 y c2=1.5, alcanzó una mejor posición de [-0.011908779322239521, 0.006539585329284364] y un error de 0.00018458520122479087 en 2.92 segundos, logrando un buen balance entre exploración y explotación y alcanzando un error cercano al umbral deseado en un tiempo razonable.

La **configuración 7**, con parámetros w=0.5, c1=1.5 y c2=1.5, alcanzó una mejor posición de [0.001947583664176895, 0.00027400716297474106] y un error de 3.87e-06 en 3.53 segundos,

siendo la más eficiente al lograr un buen equilibrio entre exploración y explotación, con un error bajo en un tiempo moderado, sin riesgo de sobreajuste.

3.2.2.- Error Final y Estabilidad

El análisis del error final indica cómo las diferentes configuraciones afectan la estabilidad de la convergencia:

- **Configuración 1**: (w=0.5, c1=1, c2=2) mostró un error de 4.131602727567102e-06. Esta configuración logró un buen equilibrio entre exploración y explotación, pero la estabilidad fue ligeramente menor en comparación con otras configuraciones.
- **Configuración 2**: (w=0.2, c1=1, c2=2) alcanzó un error de 1.5496706468063857e-05, lo que indica una mayor capacidad de exploración. Sin embargo, esto también resultó en una mayor variabilidad, haciendo que la convergencia sea menos predecible.
- **Configuración 3**: (w=0.7, c1=1, c2=2) obtuvo un error extremadamente bajo 2.1933985108027437e-08, aunque a costa de un tiempo de ejecución elevado. La alta inercia proporcionó estabilidad, pero redujo la capacidad de exploración del algoritmo.
- **Configuración 6** (w=0.5, c1=0.5, c2=1.5) tuvo un error de 0.000184585102122479087, logrando un balance adecuado entre exploración y explotación con un tiempo de ejecución moderado. Esto resultó en una estabilidad aceptable.
- **Configuración 7** (w=0.5, c1=1.5, c2=1.5) mostró el mejor desempeño con un error de 3.868162054330167e-06 y un tiempo razonable de 3.53 segundos. Esta configuración presentó la mejor combinación de estabilidad y eficiencia, evitando tanto la convergencia anticipada como el sobreajuste.

3.2.3.- Selección de la Configuración Óptima

De acuerdo con los resultados obtenidos, la **configuración óptima** es:

La configuración 7, con parámetros w=0.5, c1=1.5 y c2=1.5, alcanzó un error de 3.87e-06 en 3.53 segundos, proporcionando un equilibrio excepcional entre exploración y explotación, logrando una convergencia rápida y soluciones precisas, evitando tanto la exploración insuficiente (por un valor de w demasiado alto) como el sobreajuste (debido a valores elevados de c1 y c2).

Por otro lado:

La configuración 4 (w=0.9) y 5 (c1=2, c2=3) demostraron ser ineficientes debido a tiempos de ejecución prolongados, sin alcanzar una solución adecuada, y fueron interrumpidas antes de lograr la convergencia.

3.2.4.- Resumen de Resultados

En resumen, los resultados de las pruebas sugieren que la configuración más efectiva para el algoritmo PSO en este caso es:

La configuración óptima, con parámetros w=0.5, c1=1.5 y c2=1.5, logró un tiempo de ejecución de 3.53 segundos y un error final de 3.87e-06, destacándose por su alta eficiencia al mantener un balance adecuado que permitió al algoritmo encontrar soluciones óptimas sin comprometer la estabilidad ni aumentar excesivamente el tiempo de ejecución.

La configuración óptima logró maximizar la eficacia del algoritmo, permitiendo que las partículas convergieran eficientemente sin caer en óptimos locales ni sobreajustes, asegurando soluciones de alta calidad en un tiempo de ejecución razonable.

4.- Conclusión

El trabajo permitió evaluar y optimizar el algoritmo de Optimización por Enjambre de Partículas (PSO) en un entorno interactivo con 20 partículas. El objetivo fue entender el comportamiento del algoritmo y determinar la configuración de parámetros más adecuada para obtener los mejores resultados posibles.

Evaluación del desempeño del algoritmo PSO: El algoritmo demostró ser eficaz en encontrar soluciones cercanas a la óptima bajo diversas configuraciones de parámetros. Se observó que el PSO es eficiente cuando se logra un equilibrio adecuado entre exploración y explotación. Las partículas, inicialmente dispersas, pudieron orientarse rápidamente hacia la mejor solución, siempre y cuando los coeficientes cognitivo y social no fueran demasiado elevados para evitar sobreajuste. Además, el entorno interactivo proporcionó una comprensión visual del proceso de optimización, permitiendo ajustar manualmente las posiciones y observar el progreso de las partículas, lo que facilitó el análisis de cómo los parámetros afectan el comportamiento del algoritmo.

Reflexiones sobre la optimización de parámetros: La optimización de parámetros es esencial para mejorar la eficiencia y calidad de las soluciones en el algoritmo PSO. Los resultados muestran que el ajuste de los parámetros como el coeficiente de inercia (w), el coeficiente cognitivo (c1) y el coeficiente social (c2) influye en el tiempo de convergencia y la precisión de los resultados:

1. **Impacto en el tiempo de convergencia:** Un mayor valor de inercia mejora la exploración, pero puede aumentar el tiempo de ejecución.
2. **Equilibrio entre exploración y explotación:** Un balance adecuado entre estos dos factores permite una convergencia rápida sin sacrificar la calidad de la solución.
3. **Evitar la convergencia precipitada:** Los valores altos de c1 y c2 pueden causar convergencia precipitada, limitando la exploración de nuevas soluciones.
4. **Adaptación a problemas específicos:** Los parámetros deben ajustarse a cada problema específico para obtener los mejores resultados.
5. **Conclusión general:** Un ajuste correcto de los parámetros mejora el rendimiento del algoritmo, optimizando el uso de recursos, reduciendo los tiempos de ejecución y evitando la convergencia precipitada, lo que lleva a soluciones más precisas y eficientes.

En resumen, la optimización adecuada de los parámetros en PSO es crucial para maximizar la eficiencia del algoritmo, mejorando tanto el tiempo de ejecución como la precisión de las soluciones.

5.- Bibliografía

La bibliografía utilizada en la elaboración de este trabajo abarca tanto fundamentos teóricos como recursos prácticos para la implementación del algoritmo de optimización de enjambre de partículas (PSO) en un entorno interactivo. Se destacan las siguientes fuentes:

1. Wikipedia. (s.f.). *Particle Swarm Optimization*. Recuperado de: https://en.wikipedia.org/wiki/Particle_swarm_optimization
2. Sancho, F. (2019). *PSO: Optimización por enjambre de partículas*. Recuperado de: https://www.cs.us.es/~fsancho/Blog/posts/PSO.md
3. Amat Rodrigo, J. (2018). *Funciones de optimización PSO en R*. Recuperado de:

https://github.com/JoaquinAmatRodrigo/Estadistica-con-R/blob/master/code/funciones_optimizacion_pso.R
4. Ciencia de Datos. (2020). *Optimización con Particle Swarm*. Recuperado de: https://cienciadedatos.net/documentos/49_optimizacion_con_particle_swarm
5. Ciencia de Datos. (2020). *Crear partícula en PSO*. Recuperado de: https://cienciadedatos.net/documentos/49_optimizacion_con_particle_swarm#crear_part%C3%ADcula
6. YouTube. (2019). *Optimización con PSO - Introducción*. Recuperado de: https://www.youtube.com/watch?si=xhhCzb3jpw24GFjT&v=YXF3ZJQx1gg&feature=youtu.be&themeRefresh=1
7. YouTube. (2019). *Particle Swarm Optimization - Explicación y aplicación*. Recuperado de: https://www.youtube.com/watch?v=kmZqXGevwKA
8. YouTube. (2019). *Algoritmo PSO: Solución de problemas de optimización*. Recuperado de: https://www.youtube.com/watch?v=P7qJayfGkrU
9. YouTube. (2020). *Implementación de PSO en Python*. Recuperado de: https://www.youtube.com/watch?v=k8lGpNmjnwk
10. YouTube. (2020). *Particle Swarm Optimization para la resolución de problemas de optimización*. Recuperado de: https://www.youtube.com/watch?v=VJW1jeBFQSo
11. **Códigos proporcionados por el profesor (Draggable.py y Main.py)**: Referencias clave que facilitaron la implementación interactiva y visualización del PSO en el entorno de trabajo.

Estas fuentes fueron fundamentales para comprender tanto la teoría como la implementación práctica del PSO, permitiendo una visualización interactiva en tiempo real del proceso de optimización.

6.- Anexos

El fichero **Práctica 1.5.- Entendiendo los algoritmos PSD.pdf** contiene un análisis exhaustivo de los algoritmos de optimización por enjambre de partículas (PSO), abarcando desde el contexto y objetivos del trabajo, hasta la implementación y análisis de resultados. Incluye secciones como la descripción general del uso de PSO en optimización, una explicación detallada del código de los archivos utilizados (como **Draggable.py** y **main.py**), pruebas y análisis de resultados, y la

selección de la configuración óptima. Además, presenta un mapa mental que resume los conceptos clave de PSO y una sección de anexos con recursos adicionales.

El archivo **PantalladeejecucióndelaConfiguraciones.rar** es un archivo comprimido que incluye las capturas de pantalla y resultados generados por las distintas configuraciones del algoritmo PSO.

El archivo **Mapa Mental - Entendiendo los algoritmos PSO (Particle Swarm Optimization).pdf** es un mapa mental que resume de manera visual los conceptos clave, principios y aplicaciones del algoritmo PSO, facilitando su comprensión y conexión con otros temas relacionados.

El archivo **main.py** es el script principal en Python que implementa el algoritmo PSO y se utilizó en las pruebas realizadas para este estudio.

El archivo **Draggable.py** es un archivo Python relacionado con la implementación y ejecución de la optimización en problemas específicos, complementando al script principal.

7.- Mapa Mental

El mapa mental desarrollado abarca el análisis del algoritmo PSO (Particle Swarm Optimization) y su aplicación en Python, incluyendo la gestión de la interfaz gráfica con el archivo 'Draggable.py', la implementación del PSO en 'main.py', y el análisis de diversas configuraciones de parámetros (inercia, coeficiente cognitivo y social) para optimizar los resultados en términos de tiempo de ejecución y precisión, detallando tanto el funcionamiento de las clases y métodos empleados como el impacto de cada ajuste en la eficiencia del algoritmo.

Índice Alfabético

A

actualiza 10
adapta 3
adaptabilidad 3
afectan 13, 14, 16, 17
agrega 4
ajusta 3
ajustan 12, 13
ajustando 11
ajuste 3, 13, 18, 21
alargó 15
alcanzando 15
alcanzó 15, 16
aleatorios 7
algoritmo 3, 8, 9, 10, 12, 13, 14, 16, 17, 18, 20, 21
algoritmos 19, 20
almacena 7
ambas 9
análisis 3, 14, 16, 17, 19, 21
animales 3
anteriores 9
añade 6
aplicación 3, 19, 21
aplicaciones 20
archivos 2, 3, 4, 7, 19
áreas 15
argumento 8
argumentos 8
arrastrables 6
arrastre 6
asegurando 17
asegurándose 10
asigna 6
aumento 15

B

balance 15, 16, 17, 18
bases 3
buscan 3
buscando 12
búsqueda 3, 9, 11, 13

C

calcula 11, 13
cálculos 7
calidad 13, 17, 18
capacidad 3, 14, 16
centra 3
centraron 13
cercano 15
círculos 4
clase 6, 10, 12, 13
clave 3, 13, 19, 20
código 2, 4, 6, 7, 8, 9, 10, 11, 13, 19
cognición 13
cognitivos 8, 15
colaboración 3
colores 9
comando 8
comandos 8
comas 8
combinación 14, 16
combinaciones 14
comparación 16
complejos 3
comportamiento 3, 10, 17
comprensión 17, 20
concurrentes 7
condiciones 14
conecta 6
configura 8
considerando 10
controlan 8, 13
convergencia 3, 15, 16, 17, 18
convergieran 17
costa 16
creando 9
csv 7, 11
cuadrados 11
cuadrático 11

D

decimales 13
define 8, 9, 10, 11
definidos 6, 10, 12
demostraron 17
demostró 17
descripción 19
destacan 18
destacándose 17
detallan 14
devuelve 7, 11
diferencias 11
dimensión 10
dimensiones 9
diseño 3
dispersas 17
distancia 11
dividiendo 9
duración 7

E

eficacia 17
eficiencia 3, 14, 16, 17, 18, 21
ejecución 7, 8, 13, 14, 15, 16, 17, 18, 20, 21
ejecuciones 14, 15
ejecuta 8, 13
ejecute 8
elaboración 18
elementos 6
elevados 16, 17
emplean 3

enjambre 3, 8, 12, 13, 18, 19
entorno 3, 17, 18, 19
época 7
equilibra 3
equilibrio 15, 16, 17
error 10, 11, 13, 14, 15, 16, 17
errores 12, 15
escritura 7
espacios 3
específicos 18, 20
estabilidad 16, 17
establece 4, 10
estableciendo 8, 9
estén 9
estudio 20
eventos 6
evitando 16, 18
exceso 15
experimentos 2, 13, 14
explicación 19
exploración 3, 13, 15, 16, 17, 18
explotación 3, 13, 15, 16, 17, 18

F

facilitando 20
facilitó 17
factores 3, 18
favoreció 15
figuras 4
finales 15
flotantes 11
formato 8, 9
fórmulas 10
fuentes 18, 19
fueran 17
funcionamiento 3, 21

G

gestiona 3
globales 9
gráficas 7, 9
gráficos 4
guarda 6

H

hilos 7

I

impactan 14
impacto 13, 21
implementa 12, 20
implementación 3, 18, 19, 20, 21
importa 4, 7
imprime 13
impulsa 3
indica 8, 16
inercia 3, 8, 10, 13, 15, 16, 18, 21
influyen 3
inicia 8, 13
iniciales 7
inicializa 12
inicio 7, 13
instancia 13
interacción 3, 6
interactiva 19
interactivo 3, 6, 13, 17, 18
interfaz 3, 21
intérprete 8

L

leyenda 6
librerías 4
límites 4, 6, 9, 10, 12, 13
lista 6
locales 17
logran 12
logrando 15, 16
logró 15, 16, 17
llamando 8
llevaron 13

M

manejo 6
matemáticas 3
matemáticos 7
matplotlib 4
mejora 18
mejores 9, 15, 17, 18
metaheurístico 3
métodos 6, 21
modelos 3
módulos 7
mostrando 15
mostró 16
moverse 9
moviéndose 10
movimiento 9, 12
muestra 6, 13
muestran 18
multidimensionales 3
múltiples 14

N

necesarias 4
necesita 6
numéricos 7
números 11

O

objetivos 2, 3, 19
objeto 6
objetos 6
opción 14
operaciones 7
optimización 2, 3, 10, 12, 13, 17, 18, 19, 20
optimizando 18

optimization ____ 18
óptimos ____ 17

P

paralelo ____ 8
parámetros ____ 2, 3, 8, 13, 14, 15, 16, 17, 18, 21
particle ____ 19
partículas ____ 3, 8, 9, 12, 13, 15, 17, 18, 19
pasando ____ 8, 13
pdf ____ 19, 20
peces ____ 3
permite ____ 6, 8, 10, 14, 18
permiten ____ 7
permitiendo ____ 3, 17, 19
permitieron ____ 14
permitió ____ 17
permitirá ____ 3
personales ____ 9
posibles ____ 17
posición ____ 3, 6, 10, 11, 12, 13, 15
posiciones ____ 9, 12, 15, 17
posterior ____ 6
precisas ____ 16, 18
precisión ____ 13, 18, 21
presentan ____ 15
presentó ____ 16
principales ____ 3
probaron ____ 14
problemas ____ 3, 18, 19, 20
proceso ____ 12, 13, 17, 19
profesor ____ 19
programa ____ 7, 8
progreso ____ 17
prolongados ____ 17
proporcionando ____ 16
proporcionó ____ 16, 17
provocó ____ 15
pruebas ____ 13, 14, 17, 19, 20
pso ____ 19
pudieron ____ 17
pulgadas ____ 4

Q

quedaran ____ 15

R

rango ____ 4, 9
rapidez ____ 13
ratón ____ 6
realizaron ____ 14
rectángulos ____ 4
recursos ____ 18, 20
rendimiento ____ 3, 18
representa ____ 3, 10, 11
restando ____ 13
resto ____ 8
resultados ____ 2, 14, 15, 16, 17, 18, 19, 20, 21
resultó ____ 15, 16
resume ____ 20
resumen ____ 17, 18
retornando ____ 11
riesgo ____ 16
rutas ____ 3

S

seguidos ____ 14
segundos ____ 7, 13, 15, 16, 17
selección ____ 2, 14, 20
simplicidad ____ 3
sistemas ____ 3
sobreajustes ____ 17
sociales ____ 8, 15
socialización ____ 13
solución ____ 3, 11, 13, 17, 18
soluciones ____ 3, 14, 16, 17, 18
subgráfico ____ 4
swarm ____ 18, 19

T

tareas ____ 3
teoría ____ 19
términos ____ 21

U

uso ____ 2, 3, 9, 18, 19
usuario ____ 3, 6
utilizando ____ 3, 6, 7, 8, 10

V

valores ____ 7, 9, 10, 11, 16, 18
variando ____ 13
velocidades ____ 9, 10, 12
visualiza ____ 12
visualización ____ 4, 9, 19
visualizaciones ____ 7, 9