Reporte Técnico: Algoritmo Híbrido (Genético + Recocido Simulado)

Documentación de Código

4 de diciembre de 2025

1. Resumen Ejecutivo

Este documento detalla la implementación de un **Algoritmo Memético (Híbrido)**. La estrategia combina dos metaheurísticas para optimizar funciones complejas:

1. **Fase 1 (Genético - GA):** Utiliza una población para explorar todo el espacio de búsqueda y encontrar la "montaña.º zona prometedora (*Exploración Global*).
2. **Fase 2 (Recocido Simulado - SA):** Toma al mejor individuo encontrado por el GA y realiza una búsqueda fina en esa zona específica para hallar el mínimo exacto (*Explotación Local*).

2. Análisis Detallado por Componentes

2.1. La Función Objetivo (`michalewicz`)

Esta es la función matemática que intentamos minimizar.

```python
def michalewicz(x, m=10):
    d = len(x)  # Detecta la dimensionalidad
    suma = 0
    for i in range(d):
        # Formula compleja con senos que crea multiples minimos locales
        xi = x[i]
        suma += np.sin(xi) * np.sin(((i + 1) * xi**2) / np.pi)**(2 * m)
    return -suma # Retorna negativo para minimizar
```

La función crea muchos "valles falsos"que engañan a algoritmos simples. El retorno es negativo porque los algoritmos de optimización estándar suelen buscar minimizar el valor.

2.2. Componentes del Algoritmo Genético (GA)

2.2.1. Selección (`seleccion_torneo`)

Decide qué individuos tienen derecho a reproducirse.

- **Lógica:** Elige al azar k individuos (por defecto 3).
- **Ganador:** De esos 3, selecciona el que tenga el mejor *fitness* (menor valor). Es una competencia local que preserva la presión selectiva.

2.2.2. Cruce (`cruce`)

Combina dos padres para crear descendencia utilizando el método **BLX-α**.

```
hijo1[i] = np.random.uniform(val_min - alpha*rango, val_max + alpha*rango)
```

A diferencia de cortar y pegar genes, este método genera hijos en un intervalo numérico alrededor de los padres, expandido por un factor α. Esto fomenta la diversidad alrededor de las soluciones buenas.

2.2.3. Mutación (`mutacion`)

Introduce variaciones aleatorias para evitar el estancamiento.

```
if np.random.rand() < tasa_mutacion:
    # Ruido gaussiano
    mutado[i] += np.random.normal(0, sigma)
    # Clipping para mantener dentro de los limites
    mutado[i] = np.clip(mutado[i], lim_inf, lim_sup)
```

El uso de `np.clip` es crucial para asegurar que la mutación no saque al individuo del espacio de búsqueda válido (ej. 0 a π).

2.3. El Motor Genético (`algoritmo_genetico`)

Es el bucle principal de la primera fase.

1. **Inicialización:** Crea una población aleatoria dispersa uniformemente.
2. **Elitismo:**

   ```
   nueva_poblacion = [poblacion[i].copy() for i in range(tam_elite)]

   ```

   Los mejores individuos (por defecto 2) pasan intactos a la siguiente generación (ïnmortales"). Esto garantiza que la calidad de la solución nunca disminuya.
3. **Salida:** Tras las generaciones definidas, entrega el mejor individuo encontrado para ser usado por la siguiente fase.

2.4. El Motor de Recocido Simulado (`recocido_simulado`)

Esta es la fase de refinamiento.

2.4.1. Recepción de la Solución

```
solucion_actual = solucion_inicial.copy()
```

Aquí ocurre la hibridación. La solución inicial **no es aleatoria**, sino que es el mejor resultado entregado por el Algoritmo Genético. Empezamos con ventaja.

2.4.2. Exploración de Vecinos y Criterio de Metrópolis

El algoritmo explora vecinos cercanos mediante perturbación gaussiana.

- Si el vecino es mejor ($\Delta E < 0$): Se acepta inmediatamente.
- Si el vecino es peor: Se acepta con una probabilidad probabilística:

$$P(\text{aceptar}) = e^{-\Delta E/T}$$

Esto permite al algoritmo "saltar"fuera de pequeños mínimos locales para buscar una mejor caída.

2.4.3. Enfriamiento

```
temperatura *= alfa
```

La temperatura disminuye geométricamente. Al inicio (alta temperatura), el algoritmo acepta muchos movimientos "malos"(exploración). Al final, se vuelve estricto (explotación).

2.5. El Orquestador (`hibrido_ga_sa`)

Esta función coordina ambas fases y genera los resultados.

1. **Fase 1:** Ejecuta `algoritmo_genetico` y obtiene un ganador provisional.
2. **Transición:** Pasa ese ganador como `solucion_inicial` al `recocido_simulado`.
3. **Fase 2:** Ejecuta el SA para pulir la solución.
4. **Reporte:** Calcula el porcentaje de mejora entre el final del GA y el final del SA, y genera las gráficas de convergencia.

3. Conclusión: ¿Por qué híbrido?

La combinación supera a los algoritmos individuales:

- Si usamos solo **GA**: Puede acercarse mucho al óptimo, pero le cuesta converger al valor exacto decimal debido a la naturaleza estocástica del cruce.
- Si usamos solo **SA**: Corre el riesgo de quedar atrapado en un mínimo local si la solución inicial aleatoria cae en un "valle equivocado".
- **Híbrido**: El GA actúa como un mapa satelital (encuentra la zona correcta) y el SA actúa como un dron táctico (aterriza en el punto exacto).