Josue Santana Robledo Corona 325073061

Maestría en Ciencias de la Robótica e Inteligencia Artificial

Algoritmos Bio-Inspirados

Centro Universitario de Ciencias Exactas e Ingeniería.

Mtro. Carlos Alberto López Franco

1. Introducción

Este reporte presenta la implementación optimizada del Método de Golden Search, una versión mejorada que reduce significativamente el tiempo de ejecución del algoritmo al calcular solo un nuevo punto por iteración, en lugar de dos.

Aprovechando la propiedad fundamental de la proporción áurea (ϕ), que permite reutilizar uno de los puntos de la iteración anterior, este algoritmo el funcionamiento del método original mientras minimiza el costo computacional.

2. Descripción del código

El código implementa el algoritmo de Golden Search optimizado con proporción áurea para encontrar el mínimo de una función dentro de un intervalo dado. El funcionamiento consiste en dividir el intervalo de búsqueda usando la proporción áurea (aprox. 0.618), reduciendo el espacio de búsqueda de manera más eficiente en cada iteración hasta alcanzar la precisión deseada. En esta versión del código reutilizaremos variables.

3. Componentes del Código

Variables y parámetros iniciales

```
Rango=[-4,8]
a0,b0= Rango[0],Rango[1]
fi=((-1+sqrt(5))/2)
ro=1-fi
presicion = 0.0001
ciclo=0
```

Función Objetivo

```
def funcion(x):
  return (x-2)**2 + (0.5*x)
```

Reutilizamos función del algoritmo anterior.

4. Algoritmo Principal

El algoritmo sigue estos pasos en cada iteración:

División del intervalo en proporción aurea:

```
a1 = a0 + ro * (b0 - a0)  # Punto en 38.2% del intervalo
b1 = a0 + fi * (b0 - a0)  # Punto en 61.8% del intervalo
```

Evaluación y decisión:

```
    if funcion(a1) > funcion(b1):
      a0=a1
      a1=b1
      fa1=fb1
```

```
    b1=a0 + fi*(b0 - a0)
    fb1=funcion(b1)
    estado="Decrece"
  else:
    b0=b1
    b1=a1
    fb1 = fa1
    a1=a0 + ro*(b0 - a0)
    fa1=funcion(a1)
    estado = "Crece"
```

Aquí ve el código si la función crece o decrece y en base a ello define sus nuevos límites. Pero hay una pequeña modificación respecto al anterior, vemos que al decrecer, sus limites se vuelven a1 a b0, sin embargo b1 se convierte en el nuevo a1, por lo que ese valor y su función se pueden salvar y entonces optimizarse, procedemos a calcular b1 y su valor f(x), lo mismo cuando crece, sus limites se convierten en a0 a b1, por lo que a1 pasa a convertirse en el nuevo b1, y calculamos solamente a1 y su función, esta parte a la hora de implementarlo puede ser algo confuso, me llevó un poco de tiempo y errores, había veces en donde al estar mal implementado hacia un loop infinito, daba un valor erroneo o hacia miles de iteraciones, pero al final el código funcionó.

Criterio de parada: El ciclo continúa hasta que el tamaño del intervalo sea menor que 0.0001.

5. **Resultados Obtenidos**

El código genera una tabla detallada que muestra en cada iteración

```
23  1.7499  1.7501  0.0002  0.0001  0.0001  1.7499  1.7500  0.9375  0.9375
Decrece
24  1.7499  1.7501  0.0001  0.0000  0.0001  1.7500  1.7500  0.9375  0.9375
Decrece
25  1.7499  1.7500  0.0001  0.0000  0.0000  1.7500  1.7500  0.9375  0.9375
Crece
Mínimo en x ≈ 1.7499803071307685, f(x) ≈ 0.9375000003878091
```

En esta tabla podemos ver el número de ciclos y las variables que se fueron utilizando durante el código, lo que nos ayuda a ver la evolución del código hacia encontrar el mínimo. Vemos que hizo la misma cantidad de ciclos que el no optimizado, pero en cuestión de tiempo fue mejor.

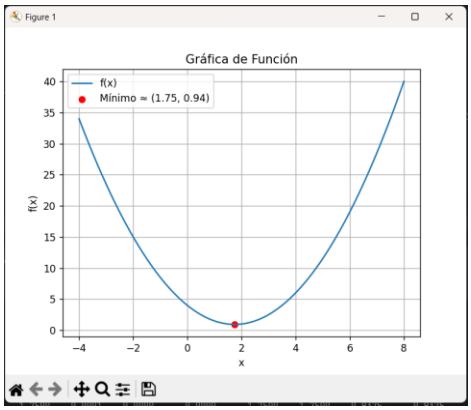

Imagen 1
Gráfica resultante

6. Ventajas del Método

Su principal ventaja es que es más rápido, solo calcula una variable por ciclo, a excepción de la primera iteración, lo que reduce considerablemente el tiempo de computo.

7. Limitaciones

Su limitación sería que igual que los demás solo puede con un mínimo por función.

8. Conclusiones

El código implementa correctamente el algoritmo de búsqueda Golden y demuestra su efectividad para encontrar el mínimo de la función cuadrática especificada. Este fue más confuso de implmentar, como expliqué en el desarrollo me llevó un tiempo de intentar, buscarle la lógica y hacer tablas a mano, pero eentualmente se logró, y es hasta ahora el algoritmo más eficiente de los que llevamos.

9. Código

```
# ## Algoritmo Golden Search Optimizado


import matplotlib.pyplot as plt

import numpy as np

from math import sqrt


# Definir el rango inicial de búsqueda [a0, b0]

Rango = [-4, 8]

a0, b0 = Rango[0], Rango[1]  # Extraer los límites inferior y superior
```

```
# Calcular la proporción áurea (ϕ) y su complemento (ρ)
# ϕ = (√5 - 1)/2 ≈ 0.618 (proporción áurea)
# ρ = 1 - ϕ ≈ 0.382
fi = ((-1 + sqrt(5)) / 2)  # ϕ (phi) - razón áurea
ro = 1 - fi  # ρ (rho) - complemento de la razón áurea

# Parámetros del algoritmo
presicion = 0.0001  # Precisión deseada para la convergencia
ciclo = 0  # Contador de iteraciones


# Función objetivo que queremos minimizar: f(x) = (x-2)^2 + 0.5*x
def funcion(x):
   return (x - 2) ** 2 + (0.5 * x)


# Encabezado de la tabla de resultados (corregido para el método de la sección dorada)
print(
   f"{'Ciclo':<6}{'a0':<10}{'b0':<10}{'delta':<10}{'Ro delta':<12}{'Fi
delta':<12}{'a1':<10}{'b1':<10}{'f(a1)':<12}{'f(b1)':<12}{'Estado':<12}")
print("-" * 110)

# Inicialización de los puntos y evaluación inicial de la función
# (FUERA del bucle para optimizar - solo se calculan una vez al inicio)
a1 = a0 + ro * (b0 - a0)  # Punto a 38.2% del intervalo (ρ)
b1 = a0 + fi * (b0 - a0)  # Punto a 61.8% del intervalo (ϕ)
fa1, fb1 = funcion(a1), funcion(b1)  # Evaluar la función en los puntos iniciales

# Bucle principal del algoritmo de la sección dorada OPTIMIZADO
```

```
while b0 - a0 > presicion:  # Continuar mientras el intervalo sea mayor que la precisión

    # Comparar los valores de la función en los puntos a1 y b1
    if fa1 > fb1:  # Usamos los valores precalculados fa1 y fb1
        # Si f(a1) > f(b1), el mínimo está en [a1, b0]
        a0 = a1  # Descartar el subintervalo izquierdo [a0, a1]
        a1 = b1  # Reutilizar b1 como nuevo a1 (OPTIMIZACIÓN CLAVE)
        fa1 = fb1  # Reutilizar el valor de función ya calculado
        # Calcular solo el nuevo punto b1
        b1 = a0 + fi * (b0 - a0)
        fb1 = funcion(b1)  # Evaluar la función solo en el nuevo punto
        estado = "Decrece"  # La función está decreciendo hacia la derecha

    else:
        # Si f(a1) <= f(b1), el mínimo está en [a0, b1]
        b0 = b1  # Descartar el subintervalo derecho [b1, b0]
        b1 = a1  # Reutilizar a1 como nuevo b1 (OPTIMIZACIÓN CLAVE)
        fb1 = fa1  # Reutilizar el valor de función ya calculado
        # Calcular solo el nuevo punto a1
        a1 = a0 + ro * (b0 - a0)
        fa1 = funcion(a1)  # Evaluar la función solo en el nuevo punto
        estado = "Crece"  # La función está creciendo hacia la derecha

    ciclo += 1  # Incrementar el contador de iteraciones

    # Imprimir resultados de la iteración actual
    print(
        f"{ciclo:<6}{a0:<10.4f}{b0:<10.4f}{(b0 - a0):<10.4f}{ro * (b0 - a0):<12.4f}{fi * (b0 - a0):<12.4f}{a1:<10.4f}{b1:<10.4f}{fa1:<12.4f}{fb1:<12.4f}{estado}")
```

```
# Una vez alcanzada la precisión deseada, calcular el mínimo aproximado
xmin = (a0 + b0) / 2  # Tomar el punto medio del intervalo final como aproximación
ymin = funcion(xmin)  # Evaluar la función en el punto mínimo

# Mostrar el resultado final
print(f"Mínimo en x ≈ {xmin}, f(x) ≈ {funcion(xmin)}")

# Crear visualización de la función y el mínimo encontrado
x = np.linspace(Rango[0], Rango[1], 400)  # Generar 400 puntos en el rango original
y = funcion(x)  # Calcular los valores de la función en esos puntos

# Configurar y mostrar la gráfica
plt.plot(x, y, label="f(x)")  # Graficar la función
plt.scatter(xmin, ymin, color="red", label=f'Mínimo ≈ ({xmin:.2f}, {ymin:.2f})')  # Marcar el mínimo
plt.title("Gráfica de Función y Mínimo Encontrado (Golden Search Optimizado)")  # Título del gráfico
plt.xlabel("x")  # Etiqueta del eje X
plt.ylabel("f(x)")  # Etiqueta del eje Y
plt.legend()  # Mostrar leyenda
plt.grid(True)  # Activar grid
plt.show()  # Mostrar el gráfico
```