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

El presente estudio aplica esta técnica a una función objetivo cuadrática, detallando su funcionamiento paso a paso, su eficiencia y las conclusiones sobre su idoneidad para distintos tipos de problemas.

2. Descripción del código

El código implementa el algoritmo de Ternary Search para encontrar el mínimo de una función de una dimensión dentro de un intervalo dado, el funcionamiento consiste en dividir el intervalo de búsqueda, ya sea si crece o decrece, reduciendo el espacio de búsqueda en cada iteración hasta alcanzar la precisión deseada.

3. Componentes del Código

Variables y parámetros iniciales

```
Rango = [-4, 8]  # Intervalo inicial de búsqueda
a0, b0 = Rango[0], Rango[1]  # Límites iniciales del intervalo
fi = ((-1 + sqrt(5)) / 2)  # Proporción áurea ≈ 0.618
ro = 1 - fi  # Proporción complementaria ≈ 0.382
presicion = 0.0001  # Criterio de parada del algoritmo
ciclo = 0  # Contador de iteraciones
```

Función Objetivo

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

4. Algoritmo Principal

El algoritmo sigue estos pasos en cada iteración:

División del intervalo en tercios:

```
a1 = a0 + ((b0 - a0) / 3)   # Punto a 1/3 del intervalo
b1 = a0 + (2 * ((b0 - a0) / 3))  # Punto a 2/3 del intervalo
```

Evaluación y decisión:

```
if funcion(a1) > funcion(b1):
  a0 = a1
  estado = "Decrece"
else:
  b0 = b1
  estado = "Crece"
```

Aquí ve el código si la función crece o decrece y en base a ello define sus nuevos límites.

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

26 1.7498 1.7501 0.0003 0.0001 0.0002 1.7500 1.7501 0.9375 0.9375
Crece
27 1.7499 1.7501 0.0002 0.0001 0.0001 1.7499 1.7500 0.9375 0.9375
Decrece
28 1.7499 1.7501 0.0001 0.0000 0.0001 1.7500 1.7501 0.9375 0.9375
Crece
29 1.7500 1.7501 0.0001 0.0000 0.0001 1.7500 1.7500 0.9375 0.9375
Decrece
Mínimo en x ≈ 1.750006847654301, f(x) ≈ 0.9375000000468904

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. Y finalmente vemos el valor final al que llega y su evaluación en la función.

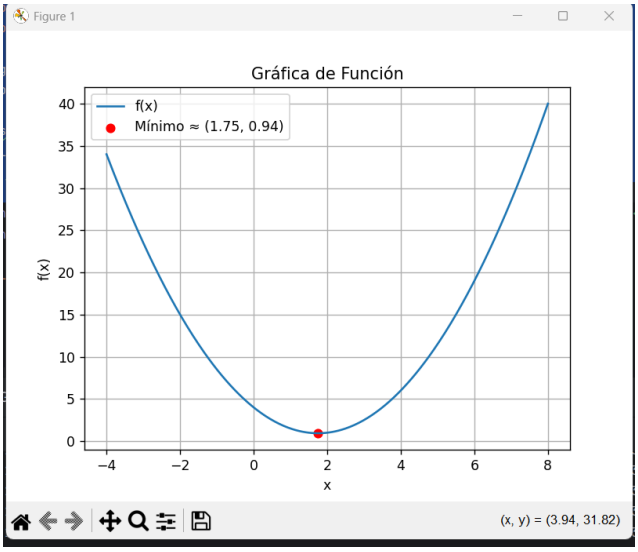

Imagen 1
Gráfica resultante

6. Ventajas del Método

Su principal ventaja es que es sencillo de entender y aplicar, puede no ser el más eficiente ya que en cada ciclo vuelve a calcular todas sus variables, pero eventualmente llega al mínimo.

7. Limitaciones

Comparado con la versión de proporción áurea, requiere más iteraciones, además de la mencionada anteriormente, evalúa dos puntos por iteración en lugar de reutilizar uno

8. Conclusiones

El código implementa correctamente el algoritmo de búsqueda ternaria y demuestra su efectividad para encontrar el mínimo de la función cuadrática especificada. Los resultados obtenidos coinciden con la solución analítica (x = 1.75), validando la correcta implementación del algoritmo. La precisión de 0.0001 se alcanza en 26 iteraciones, demostrando que podemos llegar a la solución de una función de una dimensión, aunque posteriormente habrá algoritmos con mayor eficiencia.

9. Código

```
import matplotlib.pyplot as plt

import numpy as np


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

Rango = [-4, 8]

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


# 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

print(f"{'Ciclo':<6}{'a0':<10}{'b0':<10}{'delta':<10}{'1/3 delta':<12}{'2/3
delta':<12}{'a1':<10}{'b1':<10}{'f(a1)':<12}{'f(b1)':<12}{'Estado':<12}")

print("-" * 110)


# Bucle principal del algoritmo de búsqueda por trisección

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

   # Calcular puntos que dividen el intervalo en tres partes iguales
```

```
    a1 = a0 + ((b0 - a0) / 3)    # Punto a 1/3 del intervalo
    b1 = a0 + (2 * ((b0 - a0) / 3))  # Punto a 2/3 del intervalo

    # Comparar los valores de la función en los puntos a1 y b1
    if funcion(a1) > funcion(b1):
        # Si f(a1) > f(b1), el mínimo está en [a1, b0]
        a0 = a1  # Descartar el subintervalo izquierdo [a0, a1]
        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]
        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}{(b0 - a0)/3:<12.4f}{2*(b0 - a0)/3:<12.4f}{a1:<10.4f}{b1:<10.4f}{funcion(a1):<12.4f}{funcion(b1):<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")  # 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
```