Práctica 1. "Introducción a la optimización en espacios continuos"

Dra. Miriam Pescador Rojas
Departamento de Ciencias e Ingeniería de la Computación
Algoritmos bioinspirados, ESCOM- IPN

(email: mpescadorr@ipn.mx)

12 de septiembre 2023

1 Introdución.

Carmona Serrano Ian Carlo
Villaseñor Arvizu Anael Ulaila

Un problema de optimización busca la mejor solución factible para maximizar o minimizar una función, sujeta a ciertas restricciones. Por ejemplo, es posible maximizar el beneficio de una empresa eligiendo una cantidad adecuada de productos para producir, o minimizar el costo de construcción de una estructura eligiendo adecuadamente los materiales y las dimensiones.

Matemáticamente, el problema de optimización se define de la siguiente manera:

$$\begin{array}{ll} \text{Encontrar} & \vec{x} = \{x_1, x_2, \ldots, x_n\} \text{ que} \\ \text{maximice (o minimice)} & f(x) \\ \text{sujeto a} & g_i(x) \leq b_i,\ i = 1, \ldots, m \\ & h_j(x) = c_j,\ j = 1, \ldots, p \end{array} \tag{1}$$

Donde:

- $f(x)$ es la función objetivo que se desea maximizar o minimizar.
- x es el vector de variables de decisión.
- $g_i(x)$ son las funciones de restricción de desigualdad.
- $h_i(x)$ son las funciones de restricción de igualdad.
- m es el número d erestricciones de desigualdad.
- p es el número de restricciones de igualdad.

En el proceso de optimización, debe diferenciarse los conceptos de óptimo global y óptimos locales basándose en la definición de un vecindario. Es decir, un óptimo local es un punto en el cual la función objetivo toma un valor extremo (máximo o mínimo) en comparación con puntos cercanos en su *vecindad*. No necesariamente es el valor extremo más alto o más bajo en todo el dominio de la función, sino solo en una determinada región. Pueden existir varios óptimos locales para una función. Por otro lado, un óptimo global, es el punto en el cual la función objetivo

toma su valor extremo (máximo o mínimo) en todo su dominio. Se trata del *mejor* valor posible de la función. Dependiendo de la función, puede existir un único óptimo global o múltiples puntos que alcancen ese valor extremo global (*multimodalidad*).

Una de las dificultades en un problema de optimización, especialmente cuando se usan técnicas heurísticas o métodos de búsqueda local, es la posibilidad de quedar *atrapado* en un óptimo local sin alcanzar el óptimo global.

En esta práctica estudiaremos, los métodos de búsqueda para funciones de una variable,, los cuales requieren de dos fases:

1. **Fase de acotamiento**. Se realiza una búsqueda inicial de grano grueso para acotar el óptimo.
2. **Fase de refinamiento del intervalo**. Se realiza una secuencia finita de reducciones de intervalo o refinamientos para reducir el intervalo inicial de búsqueda a una cierta precisión deseable.

2 Objetivo.

El objetivo de esta práctica es que el alumno entienda los conceptos básicos de optimización. La definición de funciones matemáticas en espacios continuos, así como el efecto de la búsqueda de soluciones óptimas para funciones mono y multimodales.

3 Consignas

Para entender los conceptos de optimización en funciones de prueba clásicas, el estudiante probará la implementación de los métodos de búsqueda exhaustiva y división por intervalos y los probará en diferentes funciones de una sola variable.

3.1 Búsqueda exhaustiva

La búsqueda exhaustiva en espacios continuos requiere técnicas de discretización para encontrar el óptimo global de una función. Este método es el más simple de los métodos de búsqueda, usa puntos igualmente espaciados e inicia su búsqueda desde el extremo izquierdo del dominio de la función.

Algoritmo

Paso 1:	$x_1 = a; \Delta x = (b-a)/n$
	(n es el número de puntos intermedios)
	$x_2 = x_1 + \Delta x; x_3 = x_2 + \Delta x$
Paso 2:	IF $f(x_1) \geq f(x_2) \leq f(x_3)$ el mínimo se encuentra
	en (x_1, x_3). TERMINAR
	ELSE $x_1 = x_2; x_2 = x_3; x_3 = x_2 + \Delta x$
	GOTO Paso 3
Paso 3:	¿Es $x_3 \leq b$? Si lo es, ir al Paso 2
	ELSE no existe un mínimo en (a, b) o un
	punto extremo (a ó b) es el mínimo.

Figure 1: Búsqueda exhaustiva

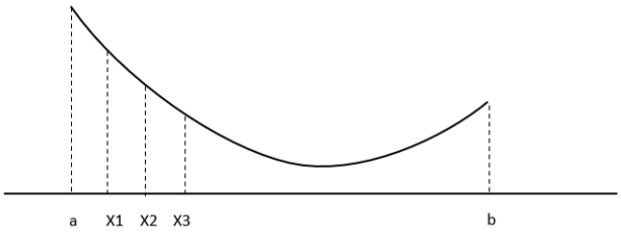

Figure 2: Proceso de búsqueda exhaustiva

3.2 Método de división por intervalos

En este método de eliminación de regiones se consideran valores en tres puntos diferentes (espaciados de forma equidistante) y se borra exactamente una mitad del intervalo a cada iteración.

Algoritmo

Paso 1:	Elegir un límite inferior a y un límite superior b.
	Definir tolerancia ϵ
	$x_m = (a+b)/2$
	$Lo = L = b - a$
	Calcular $f(x_m)$
Paso 2:	$x_1 = a + L/4$; $x_2 = b - L/4$
	Calcular $f(x_1)$ y $f(x_2)$
Paso 3:	IF $f(x_1) < f(x_m)$ $b = x_m$; $x_m = x_1$ GOTO Paso 5
Paso 4:	IF $f(x_2) < f(x_m)$ $a = x_m$; $x_m = x_2$ GOTO Paso 5
	ELSE $a = x_1$; $b = x_2$; GOTO Paso 5
Paso 5:	$L = b - a$
	IF $\|L\| < \epsilon$ TERMINAR
	ELSE GOTO Paso 2

Figure 3: Metodo de división por intervalos

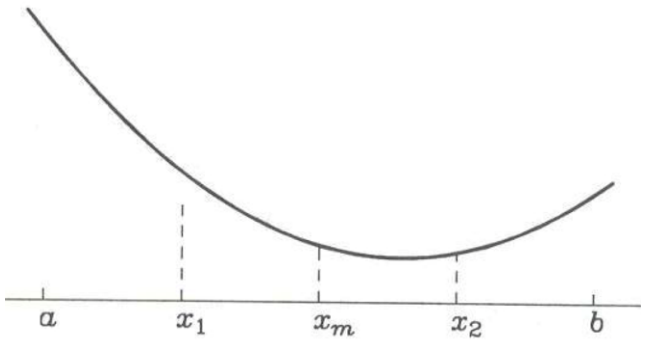

Figure 4: Proceso de búsqueda por división de intervalos

Para probar los dos métodos descritos, puede emplear el código proporcionado en python con colab. En este código aparecen 6 funciones de prueba con diferentes características.

https://colab.research.google.com/drive/1oEiFDVr7_ts_7I5ozVnu4pFF7F4HgaNX?usp=sha

3.3 Experimentos y resultados

Deberá ejecutar los dos algoritmos y reportar para los casos de prueba de la siguiente tabla.

Table 1: Resultados de pruebas con diferentes valores en los métodos de búsqueda del óptimo global

Función	B. exhaustiva (n)	solución	iteraciones	Div por intervalos (eps)	solución	iteraciones
1	10	-0.60 y -0.20	3	0.1	-1 y -0.9375	5
	100	-0.49 y -0.459	26	0.01	-1 y -0.992	8
	1000	-0.481 y -0.477	260	0.001	-1 y -0.99	11
	10,000	-0.481 y -0.480	2596	0.0001	-1 y -0.999	15
2	10	2.0 y 6.0	7	0.1	0.186 y 0.234	8
	100	3.39 y 3.79	68	0.01	0.0195 y 0.0292	11
	1000	3.559 y 3.599	679	0.001	0.0012 y 0.0018	15
	10,000	3.5779 y 3.5819	6790	0.0001	0.00015 y 0.000228	18
3	10	-0.20 y 0.19	5	0.1	-1 y -0.937	5
	100	-0.039 y 6.175e-16	49	0.01	-1 y -0.992	8
	1000	-0.0039 y 8.743e-16	499	0.001	-1 y -0.99	11
	10,000	-0.0002 y 0.00019	5000	0.0001	-1 y -0.999	15
4	10	0 y 2	6	0.1	0.156 y 0.234	7
	100	0.699 y 0.899	58	0.01	0.019 y 0.029	10
	1000	0.7399 y 0.7599	575	0.001	0.0012 y 0.001831	14
	10,000	0.7490 y 0.7510	5750	0.0001	0.000122 y 0.0001831	17
5	10	0.19 y 0.6	7	0.1	0.125 y 0.1875	7
	100	0.440 y 0.480	73	0.01	0.0156 y 0.0234	8
	1000	0.448 y 0.452	725	0.001	0.00195 y 0.0029	11
	10,000	0.4503 y 0.4507	7253	0.0001	0.0001220 y 0.0001831	15
6	10	-0.59 y 1.60	5	0.1	-5 y -4.91	7
	100	0.50 y 0.72	51	0.01	-5 y -4.994	11
	1000	0.599 y 0.621	510	0.001	-5 y -4.9993	14
	10,000	0.61 y 0.6122	5101	0.0001	-5 y -4.99991	17

Para búsqueda exhaustiva se requiere que el parámetro n sea igual a $\{10, 100, 1000, 10000\}$, mientras que para el método de división por intervalos la variable $eps = \{0.1, 0.01, 0.001, 0.0001\}$.

3.4 Discusión de resultados

Para realizar un análisis del comportamiento de los dos algoritmos, de respuesta a las siguientes preguntas:

1. ¿Qué efecto tiene aumentar el valor de n?
   Las iteraciones crecen de una forma considerable generando un gran costo computacional. A demás, los valores de los resultados son más exactos entre mayor sea el valor de n.

2. ¿Qué relación tienen n y el número de iteraciones?
   Las iteraciones mientras mayor sea el valor de n.

3. ¿Qué ocurre si se tienen multiples mínimos locales?
   Explorará todos los mínimos locales y, eventualmente, encontrará el mínimo global si existe en el espacio de búsqueda considerado.

4. ¿Se puede mejorar el método anterior?
   Para mejorar el algoritmo de búsqueda exhaustiva, se pueden aplicar técnicas de poda para evitar explorar soluciones parciales innecesarias, reduciendo así el espacio de búsqueda. Además, se puede implementar una estrategia de búsqueda más eficiente, como la búsqueda en paralelo o la búsqueda basada en heurísticas,

que ayuden a dirigir la exploración hacia soluciones prometedoras antes de recurrir a la búsqueda exhaustiva completa.

5. ¿Cuál es la complejidad del algoritmo?
   La complejidad del algoritmo de búsqueda exhaustiva es exponencial y se expresa como O(N^k)

6. ¿Qué efecto tiene el parámetro eps en la solución encontrada?
   El valor de "eps" influye en la exactitud de la solución. Un "eps" menor genera una solución más precisa, pero conlleva una mayor necesidad de iteraciones y tiempo para alcanzarla.

7. ¿Qué relación tienen eps y el número de iteraciones?
   A medida que eps se vuelve más pequeño existe una mayor precisión deseada, aunque generalmente se necesitan más iteraciones para converger a una solución que cumpla con esa precisión.

8. ¿Qué ocurre si se tienen multiples mínimos locales?
   Cuando hay varios mínimos locales, el método de división de intervalos con un valor de "epsilon" puede terminar en cualquiera de esos mínimos, y cuál se elija dependerá de dónde comiences y del tamaño de "epsilon".

9. ¿Se puede mejorar el método anterior?
   Para mejorar el algoritmo de división de intervalos, se pueden implementar estrategias como ajustar dinámicamente el valor de "epsilon" para una convergencia más rápida y flexible tanto iniciar la búsqueda desde múltiples puntos iniciales como para encontrar soluciones más robustas.

10. ¿Cuál es la complejidad del algoritmo?
    Su complejidad temporal se describe en términos de notación Big O. En el peor caso, su complejidad es O(log n).

11. ¿Qué tan eficiente es este algoritmo respecto al de búsqueda exhaustiva?
    La complejidad del algoritmo de división por intervalos es menor que el algoritmo de búsqueda exhaustiva. Una desventaja del algoritmo de división por intervalos es que puede exclusivamente ser de rangos pequeños, a diferencia del algoritmo de búsqueda exhaustiva.

12. ¿Cuáles son las mejoras que aplicaría a cada método y por qué?
    Una mejora para el algoritmo de división por intervalos es que tuviera un rango mayor. Por otra parte, una mejora para el algoritmo de búsqueda exhaustiva sería que empleé una menor cantidad de recursos computacionales.

3.5 Funciones de prueba

Implemente las siguientes funciones de prueba, para cualquier número de variables (n).

- Ackley
- Rosenbrock
- Rastrigin
- Griewank

Consulte el documento proporcionado (FuncionesPrueba.pdf).

Realice pruebas, evaluando el valor óptimo (x^*).

3.6 Análisis de lectura

El texto trata sobre cómo resolver problemas muy difíciles en diferentes áreas como la economía, la ingeniería y la biología. En lugar de usar métodos complicados, se usan trucos inteligentes llamados heurísticas y metaheurísticas. Estos trucos son como atajos que nos ayudan a encontrar buenas soluciones rápidamente, aunque no siempre sean las mejores. También se mencionan algunas propiedades importantes para evaluar cuán buenos son estos trucos, como ser simples, precisos y eficientes. En resumen, las heurísticas y metaheurísticas son herramientas útiles para resolver problemas complejos en la vida real.

Diferencia entre Heurística y Metaheurística:

En el texto, se establece que las heurísticas son procedimientos que buscan soluciones de alta calidad en un tiempo computacional razonable, aunque no garantizan la optimalidad. Se utilizan cuando los problemas son difíciles de resolver de manera exacta o cuando los métodos exactos son computacionalmente costosos. Estas heurísticas son flexibles y pueden adaptarse a problemas complejos.

Por otro lado, las metaheurísticas son estrategias generales para diseñar y/o mejorar procedimientos heurísticos. Son utilizadas para resolver problemas de optimización combinatoria difíciles, donde los heurísticos clásicos no son efectivos. Las metaheurísticas proporcionan un marco de trabajo referido a algoritmos que pueden aplicarse a diversos problemas de optimización con pocos cambios significativos. Se basan en conceptos de diferentes campos, como la física, la evolución y la biología, y se utilizan para guiar la construcción de soluciones en las heurísticas.

3.7 Imagenes de Prueba

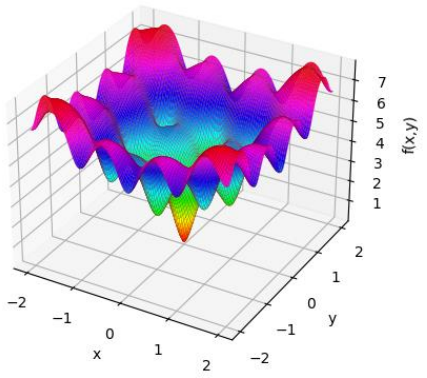

Figure 5: Grafia ackley.

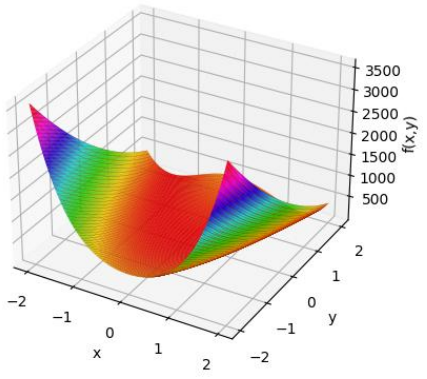

Figure 6: Grafica Rosenbrock.

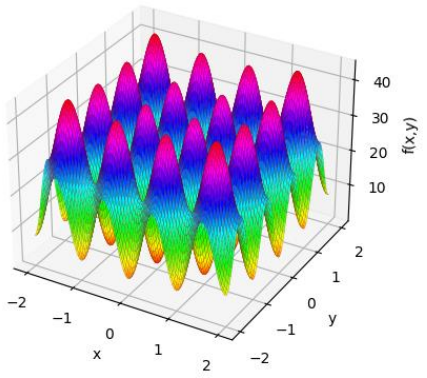

Figure 7: Grafica Rastrigin.

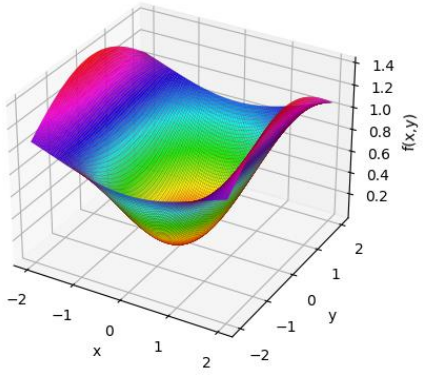

Figure 8: Grafica Griewank.

```python
# funciones de prueba
from numpy import *
from math import *
from pylab import *
import matplotlib.pyplot as plt


def f1(x):  # intervalo [-1.0, 1.0]
    # y = 65 - 0.75 / (1 + x**2) - (0.65 * x * math.atan(1/x))
    y = 65 - 0.75 / (1 + x**2) - (0.65 * x * np.arctan(1.0 / x))
    return y


def f2(x):  # intervalo [-10.0, 10.0]
    y = (1 - x) ** 4 - (2 * x + 10) ** 2
    return y


def f3(x):  # intervalo [-1.0, 1.0]
    y = 3 * x**2 + 12.0 / x**3 - 5.0
    return y


def f4(x):  # intervalo [-5.0, 5.0]
    y = x * (x - 1.5)
    return y
```

```python
def f5(x):  # intervalo [-1.0, 1.0]
    y = 3 * x**4 + (x - 1) ** 2
    return y


def f6(x):  # intervalo [-5.0, 6.0]
    y = 10 * x**3 - 2 * x - 5 * exp(x)
    return y


def buscaMinimo(x1, x2, x3, f):
    if f(x1) >= f(x2) and f(x2) <= f(x3):
        return 1
    else:
        return 0
```

```python
def buscaMinimo(x1, x2, x3, f):
    if f(x1) >= f(x2) and f(x2) <= f(x3):
        return 1
    else:
        return 0


def exhaustiva(a, b, n, f):
    x1 = a
    deltaX = (b - a) / n
    x2 = x1 + deltaX
    x3 = x2 + deltaX
    iteraciones = 0
    while True:
        iteraciones = iteraciones + 1
        if buscaMinimo(x1, x2, x3, f) == 1:
            print("El mínimo se encuentra entre", x1, " y ", x3)
            break
        else:
            x1 = x2
            x2 = x3
            x3 = x2 + deltaX
        if x3 > b:
            break
        # else:
        # print("No existe un mínimo en (a,b) o un punto extremo (a ó b) es el mínimo")
    print("Numero de iteraciones", iteraciones)
```

```python
def divisionIntervalos(a, b, eps, fun):
    xm = (a + b) / 2
    L0 = L = b - a
    fun(eps)
    iteraciones = 0
    while True:
        iteraciones = iteraciones + 1
        x1 = a + L / 4
        x2 = b - L / 4
        if fun(x1) < fun(eps):
            b = xm
            xm = x1
        elif fun(x2) < fun(eps):
            a = xm
            xm = x2
        else:
            a = x1
            b = x2
        L = b - a
        if abs(L) < eps:
            break
    print("El óptimo se encuentra en el intervalo", a, b)
    print("numero de iteraciones", iteraciones)
```

```python
prueba_busqueda_exhaustiva = [10, 100, 1000, 10000]
prueba_division_intervalos = [0.1, 0.01, 0.001, 0.0001]

print("BÚSQUEDA EXHAUSTIVA")
for i in prueba_busqueda_exhaustiva:
    print(f"PRUEBA DE LA FUNCIÓN CON VALOR: {i}\n")
    print(f"Resultado f1 {i}: {exhaustiva(-1, 1, i, f1)}")
    print(f"Resultado f2 {i}: {exhaustiva(-10, 10, i, f2)}")
    print(f"Resultado f3 {i}: {exhaustiva(-1, 1, i, f3)}")
    print(f"Resultado f4 {i}: {exhaustiva(-5, 5, i, f4)}")
    print(f"Resultado f5 {i}: {exhaustiva(-1, 1, i, f5)}")
    print(f"Resultado f6 {i}: {exhaustiva(-5, 6, i, f6)}\n")

print("DIVISIÓN INTERVALOS ")
for i in prueba_division_intervalos:
    print(f"PRUEBA DE DIVISIÓN EN EL INTERVALO: {i}\n")
    print(f"Resultado f1 {i}: {divisionIntervalos(-1, 1, i, f1)}")
    print(f"Resultado f2 {i}: {divisionIntervalos(-10, 10, i, f2)}")
    print(f"Resultado f3 {i}: {divisionIntervalos(-1, 1, i, f3)}")
    print(f"Resultado f4 {i}: {divisionIntervalos(-5, 5, i, f4)}")
    print(f"Resultado f5 {i}: {divisionIntervalos(-1, 1, i, f5)}")
    print(f"Resultado f6 {i}: {divisionIntervalos(-5, 6, i, f6)}\n")
```

```python
from matplotlib import cm
import matplotlib.pyplot as plt
import numpy as np

def ackley(x, y):
    return -20 * np.exp(-0.2 * np.sqrt(0.5 * (x**2 + y**2))) - np.exp(0.5 * (np.cos(2 * np.pi * x) + np.cos(2 * np.pi * y))) +
    np.exp(1) + 20

def rosenbrock(x, y):
    return (1 - x)**2 + 100 * (y - x**2)**2

def rastrigin(x, y):
    return x**2 + y**2 - 10 * (np.cos(2 * np.pi * x) + np.cos(2 * np.pi * y)) + 20

def griewank(x, y):
    return (x**2 + y**2) / 4000 - (np.cos(x) * np.cos(y / np.sqrt(2))) + 1
```

```python
def plotFunction(func, limInf, limSup, pts, nameFunction):
    X = np.linspace(limInf[0], limSup[0], pts)
    Y = np.linspace(limInf[1], limSup[1], pts)
    X, Y = np.meshgrid(X, Y)
    Z = func(X, Y)

    fig = plt.figure()
    ax = fig.add_subplot(111, projection="3d")
    surf = ax.plot_surface(
        X, Y, Z, rstride=1, cstride=1, cmap=cm.hsv, edgecolor="darkred", linewidth=0.1
    )

    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x,y)")

    plt.savefig(nameFunction + ".jpg")
    plt.show()

# Definir los límites y puntos para las funciones
limInf = [-2, -2]
limSup = [2, 2]
pts = 100
```