



UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

# Tarea 01 - Problemas de Optimización

ASIGNATURA

**Cómputo Evolutivo**

PROFESOR

**Oscar Hernández Constantino**

AYUDANTE

**Malinali Gónzalez Lara**

ALUMNOS

**Carlos Emilio Castañon Maldonado & Dana Berenice Hérnandez Norberto**

# Ejercicio 1. Funciones de prueba para optimización continua

1 **Investiga algunas (al menos 3) funciones que se utilizan para probar métodos de optimización continua. Las funciones deben ser diferentes a las vistas en clase.**

a) **Para cada función, da una breve descripción / discusión de sus características.**
Elegimos las siguientes funciones debido a que no poseen características similares, por lo tanto, podemos diferenciar sus resultados de manera más clara y clasificarlas posteriormente.

✪ **Función de suma ponderada**
La función de suma ponderada es una función multimodal similar a la función sphere pero con diferentes pesos y en valor absoluto.

$$f(x) = \sum_{i=1}^{d} |x_i|^{i+1}$$

Usualmente el área en la que se evalúa se encuentra restringida por el hipercubo

$$-1 \leq x_i \leq 1 \; con \; i \in 1, \ldots, n$$

El mínimo global de la función es de $f(x*) = 0$ con $x = (0, \ldots, 0)$
Es una función convexa y con un solo mínimo global, por lo cual pensamos que su complejidad radica en que no es una función uniforme como en el caso de la función sphere, si no que tiene cierta inclinación, además de abrirse más que esta función. Consideramos que es interesante conocer de qué forma influyen estos cambios en la función.

✪ **Función de Zakharov**
La función de Zakharov es una función convexa que se asemeja a una sabana siendo sostenida por dos lados. S encuentra dada por la siguiente función:

$$f(x) = \sum_{i=1}^{d} {x_i}^{i+1} + \left( \sum_{i=1}^{d} 0.5 i x_i \right)^2 + \left( \sum_{i=1}^{d} 0.5 i x_i \right)^4$$

Usualmente el área en la que se evalúa se encuentra restringida por el hipercubo

$$-5 \leq x_i \leq 10 \; con \; i \in 1, \ldots, n$$

El mínimo global de la función es de $f(x*) = 0$ con $x = (0, \ldots, 0)$
En este caso el área en la que se evalúa es más grande y por lo tanto, es posible que pueda tardar más en llegar al óptimo global, o llegar a él muy rápidamente, dependiendo del intervalo de evaluación. De la misma forma, es importante observar que por las características de de la función, esta pueda llegar a muchas soluciones que puedan parecer óptimas.

✪ **Función de Schwefel**
En el caso de la función de schwefel representa un reto mayor encontrar el óptimo global debido a que posee multiples mínimos y máximos locales que parecieran no tener ningún orden establecido. Se encuentra dada por la siguiente función:

$$f(x) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

Usualmente el área en la que se evalúa se encuentra restringida por el hipercubo

$$-500 \leq x_i \leq 500 \; con \; i \in 1, \ldots, n$$

El mínimo global de la función es de $f(x*) = 0$ con $x = (420.9687, \ldots, 420.9687)$
En comparación con las funciones anteriores, el área de evaluación es sumamente más grande, con partes con tanto convexas como cóncavas. Con el riesgo de que en un proceso de optimización se quede con una solución equivocada.

b) **¿Cuál podría ser más fácil y difícil de optimizar?, ¿Por qué?**
Consideramos que la más complicada de optimizar es la función de Schwefel debido a que tiene múltiples óptimos locales y a área en que se evalúa la función. La segunda función más complicada de optimizar consideramos que es la función de Zakharov debido a que a pesar de que tiene un solo óptimo global, existen soluciones que se acercan mucho debido a su estructura, por lo que puede ser sumamente complicado encontrar el óptimo exacto. Por último, creemos que la función de suma ponderada será más fácil de optimizar debido a su semejanza con sphere.

c) **¿La dimensión podría alterar la dificultad?**
Sí, la dimensión podría alterar la dificultad debido a que son más puntos necesarios a aproximar y por lo tanto, poseemos una variación más grande. Llegando a necesitar una cantidad más grande de iteraciones.

2 **Implementar las funciones del inciso anterior. Cada función deberá estar implementada en un método (o función) diferente. En todos los casos, se debe considerar que el parámetro de entrada será el valor del punto a evaluar x; dependiendo del lenguaje y la implementación, se puede recibir también la dimensión del problema.**

**Ejemplo:**

```
double sphere( double x [ ] ) {
double res = 0;
for(int i=0; i < x.len; i++) {
res += ....
}
return res;
}
```

A continuación podemos observar nuestras funciones implementadas, donde como parámetro requerido pueden insertar tanto un dato numérico como una lista o un arreglo.

## Función de suma ponderada

```
def sum_of_powers(x):
    x = pd.Series(x)
    d = len(x)
    y = sum(abs(x)**(x+1))
    return y
```

## Función de Zakharov

```
def zakharov(x):
    x = pd.Series(x)
    d = len(x)
    t1 = sum(x**2)
    t2 = 0.5*sum(range(d)*(x**2))
    y = t1 + (t2**2) + (t2**4)
    return y
```

### Función de Schwefel

```python
def schwefel(x):
    x = pd.Series(x)
    d = len(x)
    t1 = 418.9829*d
    t2 = sum(x*np.sin(abs(x)**(1/2)))
    y = t1-t2
    return y
```

3 **Genera un programa que te permita evaluar las funciones del ejercicio anterior. El programa debe poder ejecutarse desde consola, y recibir todos los parámetros al momento de la ejecución. La función a evaluar puede pasarse como un número o una cadena, para elegir alguna de las descritas anteriormente.**

**Por ejemplo:**

```
batman@root: /Tarea01$ evaluar sphere 2 2.1 -0.1
```

En el ejemplo:

- "evaluar" es el nombre del ejecutable
- "sphere" es el nombre de la función que queremos evaluar (de manera alternativa, pueden implementarlo como un parámetro numérico, en cuyo caso, la línea de ejecución sería algo como: $\$evaluar\ 1\ 2\ 2.1\ -0.1$ [ el 1 es un número arbitrario, que debe definirse en el reporte para especificar que con ese número indicaremos la ejecución de la función sphere ]
- 2 es la dimensión del problema
- Después de la dimensión, siguen $n$ números, correspondientes a los valores de $x_i$

### Definimos a nuestro evaluador

Definimos el evaluador donde podemos ingresar el nombre de la función como una cadena de texto, la dimensión de nuestro problema como un valor numérico y los argumentos o parámetros donde queremos evaluar alguna de nuestras funciones.

```python
def evaluar(funcion_objetivo, dimension, *args):
    if funcion_objetivo == "sum_of_powers":
        print("Funcion Objetivo: sum_of_powers")
        print("Dimension del problema: ", dimension)
        print("Evaluacion: ", sum_of_powers(args))
    elif funcion_objetivo == "zakharov":
        print("Funcion Objetivo: zakharov")
        print("Dimension del problema: ", dimension)
        print("Evaluacion: ", zakharov(args))
    elif funcion_objetivo == "schwefel":
        print("Funcion Objetivo: schwefel")
        print("Dimension del problema: ", dimension)
        print("Evaluacion: ", schwefel(args))
    else:
        print("Funcion Objetivo no valida")
```

Ejecuciones de lo anterior:

```
batman@root: /Tarea01$ python3 evaluar.py sum_of_powers 2 2.1 -0.1
Funcion Objetivo: sum_of_powers
Dimension del problema:  2
Evaluacion:  10.100132533838126
```

```
batman@root: /Tarea01$ python3 evaluar.py zakharov 2 2.1 -0.1
Funcion Objetivo: zakharov
Dimension del problema:  2
Evaluacion:  4.4200250006249995
```

```
batman@root: /Tarea01$ python3 evaluar.py schwefel 2 2.1 -0.1
Funcion Objetivo: schwefel
Dimension del problema:  2
Evaluacion:  835.9124200696331
```

# Ejercicio 2. Búsqueda aleatoria

1 Implementar una búsqueda aleatoria para problemas de optimización continua.

El programa deberá recibir como parámetros:

a) Función objetivo [ Esto puede pasarse como un número o una cadena, para elegir alguna de las descritas anteriormente]

b) Dimensión, que se utilizará para definir el espacio de búsqueda correspondiente

c) Número total de iteraciones a realizar

d) Intervalo de búsqueda

El programa deberá devolver, e imprimir en pantalla, el resultado de la búsqueda imprimiendo el valor de x ( la mejor solución encontrada), así como su evaluación.

**Ejemplo de ejecución y resultado esperado:**

```
batman@root: /Tarea01$ busqueda_aleatoria sphere 2 1000
Funcion: sphere
Dimension del problema: 2
Total de iteraciones: 1000
Mejor solucion encontrada:
x = [ 0.1 1.2 ]
f(x) = 300.86
```

## Implementación de Búsqueda aleatoria

Nuestra función tiene como parámetros de entrada el nombre de la función objetivo, es decir, el nombre de la función que queremos optimizar, la dimensión del problema, el número de iteraciones que queremos realizar y el intervalo en donde queremos evaluar donde queremos evaluar la función.

```python
# Funcion de Busqueda Aleatoria
def busqueda_aleatoria(funcion_objetivo, dimension, iteraciones, intervalo):
    mejor_solucion = np.random.uniform(intervalo[0], intervalo[1], dimension)
    mejor_evaluacion = funcion_objetivo(mejor_solucion)
    for i in range(iteraciones):
        solucion = np.random.uniform(intervalo[0], intervalo[1], dimension)
        evaluacion = funcion_objetivo(solucion)
        if evaluacion < mejor_evaluacion:
            mejor_solucion = solucion
            mejor_evaluacion = evaluacion
    print('Funcion Objetivo:', funcion_objetivo.__name__)
    print('Dimension del problema:', dimension)
    print('Total de iteraciones:', iteraciones)
    print('Mejor Solucion encontrada:')
    print('x =', mejor_solucion)
    print('f(x) =', mejor_evaluacion)
```

Ejecuciones de lo anterior:

```
batman@root: /Tarea01$ python3 busqueda_aleatoria.py sum_of_powers 2 1000 [-5.12,5.12]
Funcion Objetivo: sum_of_powers
Dimension del problema: 2
Total de iteraciones: 1000
Mejor Solucion encontrada:
x = [-4.59638578 -4.555646  ]
f(x) = 0.008701007303661846
```

```
batman@root: /Tarea01$ python3 busqueda_aleatoria.py schwefel 2 1000 [-5.12,5.12]
Funcion Objetivo: schwefel
Dimension del problema: 2
Total de iteraciones: 1000
Mejor Solucion encontrada:
x = [4.81525571 4.97287512]
f(x) = 830.1258116824008
```

```
batman@root: /Tarea01$ python3 busqueda_aleatoria.py zakharov 2 1000 [-5.12,5.12]
Funcion Objetivo: zakharov
Dimension del problema: 2
Total de iteraciones: 1000
Mejor Solucion encontrada:
x = [0.07419368 0.20817246]
f(x) = 0.04931019454632571
```

Con los resultados obtenidos con las ejecuciones obtenidas podemos observar como la función de suma ponderada y la de zakharov se acercan más al óptimo que la función schwefel, como teníamos previsto.

2 **Ejecutar la búsqueda aleatoria para todas las funciones anteriores, considerando 1,000,000 iteraciones; se deberá probar la búsqueda en diferentes dimensiones. Probar al menos con dimensiones 2, 5, 10.**

| Funcion Objetivo | Dimension | Mejor f(x) | Promedio f(x) | Peor f(x) |
|---|---|---|---|---|
| $sum\_of\_powers$ | 2 | 2.098934e-09 | 2.978298e+09 | 1.952262e+11 |
| $sum\_of\_powers$ | 5 | 4.351890e-08 | 7.384374e+09 | 2.500835e+11 |
| $sum\_of\_powers$ | 10 | 1.212787e-03 | 1.481389e+10 | 2.852122e+11 |
| $zakharov$ | 2 | 0.000082 | 6.977483e+05 | 6.252611e+06 |
| $zakharov$ | 5 | 1.620490 | 2.145085e+09 | 5.364599e+10 |
| $zakharov$ | 10 | 692966.098116 | 5.540794e+11 | 1.296510e+13 |
| $schwefel$ | 2 | 830.075215 | 837.969796 | 845.856388 |
| $schwefel$ | 5 | 2075.331918 | 2094.919634 | 2114.534886 |
| $schwefel$ | 10 | 4155.145285 | 4189.813929 | 4224.956487 |

Ahora, al probar nuestro método de búsqueda aleatoria con dos dimensiones tenemos resultados mejores que con cinco o diez dimensiones, parece que al aumentar el número de dimensiones en la búsqueda, tarda más en encontrar el óptimo, tomando mucho tiempo en ser ejecutado. Esto pasa con todas las funciones propuestas, sin embargo, es más notoria la diferencia en la función de zakharov.

# Ejercicio 3. Preguntas de repaso

**1.** Mencionar algún ejemplo de problema de optimización combinatoria, diferente a los mencionados en clase. Indicar claramente:

- Espacio de búsqueda
- Función objetivo
- Tamaño del espacio de búsqueda
- Ejemplar concreto del problema
- Ejemplo de una solución

Decidimos trabajar con el problema de mochila, debido a que no hemos estudiado en profundidad el problema. El planteamiento se basa en que queremos llevar en una mochila la mayor cantidad de objetos más importantes que podamos, basándonos en la capacidad máxima de la mochila la cual denotaremos como $W$.
Debemos tomar en cuenta ciertos factores para decidir si llevar el objeto o no, como el peso o importancia del objeto. A los pesos los denotaremos como $w_i$ y $v_i$ respectivamente, para $i = 1, \ldots, n$, con $n$ como la cantidad de objetos que consideramos llevar en la mochila. El problema lo podemos ver como un problema binario donde $x_i$ representa si llevamos o no el objeto $i$, con el valor de 1, si llevamos el objeto y 0, si no lo llevamos. Por lo tanto, se convierte en un problema con la siguiente función objetivo:

$$max \sum_{i=1}^{n} x_i v_i$$

Con la restricción:

$$W(x) = \sum_{i=1}^{n} x_i w_i \leq W$$

Por lo tanto, nuestro espacio de búsqueda se convierte en todas las soluciones que $x$ que cumplen la restricción dada, con un tamaño de búsqueda de $2^n$.Para visualizar mejor el problema es conveniente dar un ejemplo con los siguientes pesos.

| Objeto | 1 | 2 | 3 |
|---|---|---|---|
| Peso | 2 | 3 | 2 |
| Importancia | 1 | 2 | 3 |

Por lo tanto, que queremos maximizar

$$max\ x_1 + 2x_2 + 3x_3$$

s.a

$$2x_1 + 3x_2 + 2x_3 \leq 4$$

En este caso tenemos 8 soluciones posibles, veamos cuales son en este problema en particular.

| x | $w_i$ | $v_i$ |
|---|---|---|
| 0 0 0 | 0 | 0 |
| 1 0 0 | 2 | 1 |
| 0 1 0 | 3 | 2 |
| 0 0 1 | 2 | 3 |
| 1 1 0 | 5 | 3 |
| 1 0 1 | 4 | 4 |
| 0 1 1 | 5 | 5 |
| 1 1 1 | 7 | 6 |

Como podemos observar, nuestra solución óptima es $x = (1, 0, 1)$ con un peso total de 4 y una importancia óptima de 4. En este caso como tenemos pocos objetos podemos contabilizarlos de esta forma, sin embargo, no siempre será de esta forma debido a que podemos tener una cantidad muy grande de soluciones.

## Consideraciones Generales

➢ Para la lectura de parámetros desde consola, pueden apoyarse de bibliotecas auxiliares (pero se debe agregar la referencia y aclaración correspondiente en el reporte).

El entrégale para esta tarea deberá ser un archivo zip con la siguiente estructura:

```
+ # cuenta / <--- Nombre de la carpeta
    - src / <--- Carpeta con el código fuente de su implementación
    - README.txt <--- Archivo con instrucciones para compilar y ejecutar
                      se debe incluir el comando para ejecutar un ejemplo
                      de cada inciso
    - makefile <-- [Opcional]
    - reporteT1.pdf <--- Reporte de la tarea
    - ejecuciones.csv <-- [Opcional] Hoja de cálculo con la información de las
                                     ejecuciones realizadas.
```

El reporte (reporteT1.pdf) deberá incluir al menos:

- Nombre completo
- Título y número de Tarea
- Respuestas a los ejercicios planteados
- Pseudocódigo de algoritmos implementados
- Capturas de pantalla de la ejecución
- Comentarios / Conclusiones

No hay extensión mínima ni máxima, pero deben incluir las respuestas / comentarios que se piden en cada uno de los ejercicios.

## Referencias


- https://www.sfu.ca/ ssurjano/schwef.html
- https://www.sfu.ca/ ssurjano/sumpow.html
- https://www.sfu.ca/ ssurjano/zakharov.html
- Garey, Michael R. and David S. Johnson, Computers and intractibility
- Christos h. papadimitriou, Computational complexity
- https://repositorio.unan.edu.ni/8853