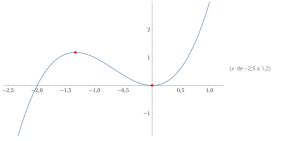
Optimización Continua

- Las variables pueden tomar valores en un intervalo continuo.
- El espacio de búsqueda es un conjunto de puntos continuos.
- Opt. de funcs. matemáticas, problemas de programación lineal continua.

E

Ej. Encuentra el mín/máx de la función $f(x) = x^2 + \log x$



⇒ Sí el espacio de búsqueda es un conjunto de puntos
↳ La dimensión "afecta" a la cantidad puntos de este conjunto

$f(\underline{x}) = \ldots$ $\quad \underline{x} = (x_1, x_2 \ldots)$

$f(x_1, \ldots, x_n) =$

Optimización combinatoria (discreta)

- Las variables están asociadas a conjuntos finitos de elementos discretos o combinatorios.
- El espacio de búsqueda formado por combinaciones de elementos discretos, por lo que los cambios en las variables son "saltos" de un conjunto a otro
- Asignación, planeación, gráficas...

TSP 1, 2, 3 / 3, 2, 1

Ej. Encontrar el conjunto de materias tales que...

Ejercicio 2. Búsqueda aleatoria

2.a) Implementar una búsqueda aleatoria para problemas de optimización continua.

El programa deberá recibir como parámetros:

i) Función objetivo [Esto puede pasarse como un número o una cadena, para elegir alguna de las descritas anteriormente]
ii) Dimensión, que se utilizará para definir el espacio de búsqueda correspondiente
iii) Número total de iteraciones a realizar
iv) 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:

```
$ busqueda_aleatoria  sphere 2 1000
Función: sphere
Dimensión del problema: 2
Total de iteraciones: 1000
Mejor solución encontrada:
x = [ 0.1  1.2 ]
f(x) = 300.86
```

Búsqueda Aleatoria

- ¿Qué es?
  Enfoque de optimización
- ¿Qué hace?
  Explorar el espacio de búsqueda (todas las posibles soluciones) de manera aleatoria
  No hay patrones, reglas, etc.
- Ventajas:
  - Útil cuando no hay mucha info de la func objetivo
  - " " el espacio de búsqueda es muy grande (hacemos una mayor exploración)
  - Versatilidad
  - Facilidad de implementación.
- Desventajas
  - Convergencia hacia un óptimo (no hay métodos para alcanzarlo)
  - Mayor dimensión, menor eficiencia
  - Pérdida de info previa

- ¿Cómo funciona?

RandomSearch (funcObj, dim, noIteraciones, intervalo)

Inicialización de parámetros: función que se va a ejecutar, no. de iteraciones, etc.
peorSol, mejorSol.

Ciclo: Repetir hasta llegar al criterio de detención
En cada iteración se genera una sol. aleatoria (contenida dentro del espacio de búsqueda)
Evaluar a la sol. candidata en la func. objetivo.

Comparar:
EvalSolActual mejoró?
↳ Sí → actualizar mejorSol
actualizar promedioSol

↳ No → comparar peorSol
- EvalSolActual empeoró?
↳ Sí → actualizar peorSol
→ actualizar promedioSol
↳ No → actualizar promedioSol

↳ Devolver mejorSol
evalMejorSol

NO SE PUEDEN USAR!
- func. opt.
Sphere, Ackley, Griewank
- Problemas opt. comb.
TSP, SAT

1.3. 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:
$ 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

```python
import sys
import math

def sphere(x):
    return sum(xi**2 for xi in x)

def ackley(x):
    n = len(x)
    sum1 = sum(xi**2 for xi in x)
    sum2 = sum(math.cos(2 * math.pi * xi) for xi in x)
    return -20 * math.exp(-0.2 * math.sqrt(sum1 / n)) - math.exp(sum2 / n) + 20 + math.e

def evalFunc(funcName, dim, *args):
    if funcName == "sphere":
        return sphere(args)
    elif funcName == "ackley":
        return ackley(args)
    else:
        print("Ingrese una función válida. Las funciones disponibles son: sphere, ackley, ...")

if __name__ == "__main__":
    if len(sys.argv) < 4:
        print("Uso: evaluar <funcion> <dim> <valores_x>")
        sys.exit(1)

    funcName = sys.argv[1]
    dim = int(sys.argv[2])
    values_x = [float(arg) for arg in sys.argv[3:]]

    result = evalFunc(funcName, dim, *values_x)
    print(f"Resultado para la función {funcName}: {result}")
```