Programación No Lineal

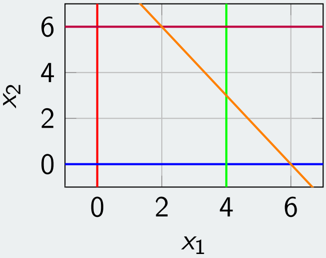
Investigación Operativa

Universidad de San Andrés

Introducción a la Programación No Lineal

Programación Lineal vs No Lineal:

- **Lineal:** Función objetivo y restricciones lineales
- **No Lineal:** Al menos una función no es lineal

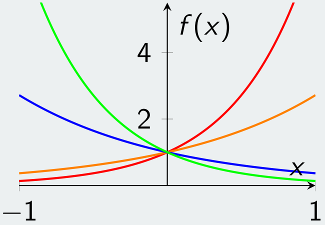


Características:

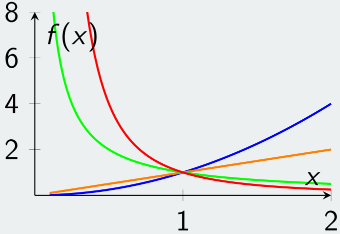
- Múltiples mínimos locales
- Dependencia del punto inicial
- Mayor complejidad computacional

Convexidad

Definición: Una función es convexa si la recta que une dos puntos cualesquiera de su gráfica queda por encima o sobre la función.



Funciones Exponenciales



Funciones Potenciales

Propiedades de Convexidad

Propiedades importantes:

- La suma de dos funciones **convexas** es convexa
- Un problema de optimización es convexo si y solo si:
  1. El conjunto factible es convexo
  2. El objetivo es minimizar una función convexa (o maximizar una cóncava)
- Si un problema es convexo, **cualquier óptimo local es un óptimo global**

Importante
En problemas convexos, alcanza con encontrar *algún* mínimo local que sabremos que es global.

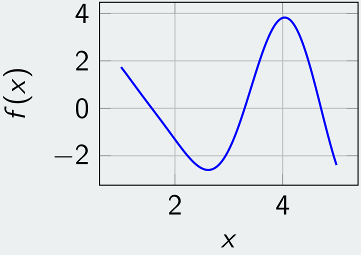
Múltiples Mínimos Locales

Problema: En problemas no lineales puede haber varios mínimos locales.

Solución: Usar seeds (semillas aleatorias) para explorar diferentes regiones.

El resultado depende del punto inicial:

- Punto A $\rightarrow$ Mínimo 1
- Punto B $\rightarrow$ Mínimo 2
- Punto C $\rightarrow$ Mínimo 3



Ejemplo 1: Asignación de Presupuesto

Problema: Una empresa desea asignar su presupuesto diario de publicidad entre Google Ads (x_1) e Instagram Ads (x_2).

Función objetivo:

$$f(x_1, x_2) = -\left(100 \cdot \left(1 - e^{-0{,}05 x_1}\right) + 80 \cdot \left(1 - e^{-0{,}08 x_2}\right)\right)$$

Restricciones:

- $x_1 + x_2 \leq 10,000$ (presupuesto total)
- $x_1 \geq 2,000$ (mínimo Google Ads)
- $x_2 \geq 1,000$ (mínimo Instagram Ads)

¿Es convexo? Sí, porque la función objetivo es cóncava (estamos maximizando) y las restricciones son lineales.

Resolución en Python - Ejemplo 1

```python
def impacto_negativo(x):
    x1, x2 = x
    return -(100 * (1 - np.exp(-0.05 * x1)) +
             80 * (1 - np.exp(-0.08 * x2)))

restricciones = [
    {'type': 'ineq', 'fun': lambda x: 10000 - (x[0] + x[1])},
    {'type': 'ineq', 'fun': lambda x: x[0] - 2000},
    {'type': 'ineq', 'fun': lambda x: x[1] - 1000}
]

bounds = [(2000, None), (1000, None)]
x0 = [5000, 3000]

res = minimize(impacto_negativo, x0, method='SLSQP',
               bounds=bounds, constraints=restricciones)

if res.success:
    x1_opt, x2_opt = res.x
    impacto_max = -res.fun
    print(f'Inversion optima en Google Ads: ${x1_opt:.2f}')
    print(f'Inversion optima en Instagram Ads: ${x2_opt:.2f}')
    print(f'Impacto total maximo: {impacto_max:.2f}')
    print(f'Total invertido: ${x1_opt + x2_opt:.2f}')
else:
    print('Error:', res.message)
```

Ejemplo 2: Producción Óptima

Problema: Una empresa fabrica dos productos A y B con ganancias unitarias de \$40 y \$30.

Función objetivo:

$$\max f(x_1, x_2) = 40x_1 + 30x_2$$

Restricciones no lineales:

- **Capacidad de máquinas:** $x_1^2 + x_2^2 \leq 2500$
- **Compatibilidad:** $\dfrac{x_1}{x_2 + 1} \leq 4$
- **No negatividad:** $x_1 \geq 0$, $x_2 \geq 0$

¿Es convexo? Sí, porque el conjunto factible es convexo (intersección de conjuntos convexos) y la función objetivo es lineal.

Ejemplo 3: Optimización en Biotecnología

Problema: Una startup busca desarrollar empaques ecológicos usando fibra vegetal (x_1) y alga marina (x_2).

Función objetivo:

$$f(x_1, x_2) = \sin(x_1) \cdot \cos(x_2) + 0{,}1(x_1 + x_2) + 25$$

Restricciones:

- $x_1 + x_2 \geq 2$ (mínimo de ingredientes)
- $x_1 + 2x_2 \leq 8$ (capacidad máxima)
- $0 \leq x_1 \leq 6$, $0 \leq x_2 \leq 6$

¿Es convexo? NO, porque contiene términos trigonométricos que pueden tener múltiples mínimos locales. Necesitamos múltiples puntos iniciales aleatorios.

Resolución con Múltiples Puntos Iniciales

```python
import numpy as np
from scipy.optimize import minimize

def costo(x):
    x1, x2 = x
    return np.sin(x1) * np.cos(x2) + 0.1 * (x1 + x2) + 25

restricciones = [
    {'type': 'ineq', 'fun': lambda x: x[0] + x[1] - 2},
    {'type': 'ineq', 'fun': lambda x: 8 - (x[0] + 2 * x[1])}
]

bounds = [(0, 6), (0, 6)]

np.random.seed(42)
resultados = []
for i in range(10):
    x0 = np.random.uniform(0, 6, size=2)
    res = minimize(costo, x0, method='SLSQP',
                   bounds=bounds, constraints=restricciones)
    if res.success:
        resultados.append((res.fun, res.x))

resultados.sort()
mejor_valor, mejor_x = resultados[0]
print(f'Mejor solucion: x1 = {mejor_x[0]:.4f}, x2 = {mejor_x[1]:.4f}')
print(f'Costo minimo: {mejor_valor:.4f}')
```

Programación Cuadrática (QP)

Definición: Problemas donde la función objetivo es cuadrática y las restricciones son lineales.

Forma general:

$$\min_{x} x^T Qx - a^T x$$

$$\text{s.a. } Ax \leq b, \quad x \geq 0$$

Ejemplo: Asignación de presupuesto en medios

Contexto: Un estudio de televisión quiere distribuir 100 unidades de presupuesto entre tres canales.

Impacto lineal: $a^T x = 5x_1 + 4x_2 + 3x_3$

Penalización cuadrática: Por rendimientos decrecientes y canibalización

$$Q = \begin{pmatrix} 0{,}04 & 0{,}01 & 0 \\ 0{,}01 & 0{,}05 & 0{,}005 \\ 0 & 0{,}005 & 0{,}03 \end{pmatrix}$$

Restricciones:

- $x_1 + x_2 + x_3 = 100$
- $10 \leq x_1 \leq 60$, $5 \leq x_2 \leq 60$, $0 \leq x_3 \leq 40$

Función Objetivo QP Expandida

Objetivo: Maximizar impacto neto = impacto lineal - penalización cuadrática

Versión de minimización: $\min Z(x) = x^T Qx - a^T x$

Forma expandida:

$$\begin{aligned} Z(x) = 0{,}04x_1^2 &+ 0{,}02x_1x_2 + 0{,}05x_2^2 \\ &+ 0{,}01x_2x_3 + 0{,}03x_3^2 \\ &- 5x_1 - 4x_2 - 3x_3 \end{aligned}$$

Resolución QP en Python

```python
import numpy as np
from scipy.optimize import minimize

# Datos del QP
a = np.array([5.0, 4.0, 3.0])  # Impacto lineal
Q = np.array([
    [0.04, 0.01, 0.00],
    [0.01, 0.05, 0.005],
    [0.00, 0.005, 0.03]
], dtype=float)

def Z(x):
    return float(x @ Q @ x - a @ x)

restricciones = [
    {'type': 'eq', 'fun': lambda x: 100.0 - (x[0] + x[1] + x[2])}
]
bounds = [(10.0, 60.0), (5.0, 60.0), (0.0, 40.0)]

x0 = np.array([40.0, 30.0, 30.0])
res = minimize(Z, x0, method='SLSQP', bounds=bounds,
               constraints=restricciones)

if res.success:
    x1_opt, x2_opt, x3_opt = res.x
    impacto_neto = a @ res.x - res.x @ Q @ res.x
    print(f'TV: {x1_opt:.2f}, Online: {x2_opt:.2f}, Radio: {x3_opt:.2f}')
    print(f'Impacto neto: {impacto_neto:.2f}')
```

Optimización Multiobjetivo

Problema: Cuando queremos optimizar varias cosas a la vez que suelen estar en conflicto.

Ejemplos:

- Minimizar costo, tiempo y emisiones
- Maximizar ganancia y minimizar contaminación
- Maximizar retorno y minimizar riesgo

Solución: Buscar el frente de Pareto (conjunto de opciones eficientes donde ninguna es mejor en todo al mismo tiempo).

Métodos: Suma ponderada (weighted sum), ϵ-constraint, Programación por metas, Metaheurísticas (algoritmos genéticos, etc.)

Frentes de Pareto

Definición: Conjunto de opciones donde ninguna es mejor en todo al mismo tiempo.

Ejemplos cotidianos:

- **Celular:** Mejor cámara vs precio más bajo
- **Ruta:** Más rápida vs sin peajes
- **Dieta:** Más sana vs más barata

Método Weighted Sum:

$$\min \sum_{i=1}^{n} w_i \cdot f_i(x)$$

Donde w_i son los pesos (que suman 1) y $f_i(x)$ son los objetivos normalizados.

Normalización de Objetivos

Problema: Los objetivos pueden tener escalas muy distintas (costo en millones, tiempo en decenas).

Solución: Normalizar los valores:

$$f_1^{\text{norm}}(x) = \frac{f_1(x) - f_1^{\text{mín}}}{f_1^{\text{máx}} - f_1^{\text{mín}}}$$

$$f_2^{\text{norm}}(x) = \frac{f_2(x) - f_2^{\text{mín}}}{f_2^{\text{máx}} - f_2^{\text{mín}}}$$

Función combinada:

$$f(x) = \alpha\, f_1^{\text{norm}}(x) + (1-\alpha)\, f_2^{\text{norm}}(x), \quad 0 \leq \alpha \leq 1$$

Proceso: Generar soluciones factibles, encontrar mín/máx de cada objetivo, normalizar y aplicar weighted sum.

Ejemplo Multiobjetivo en Python

```python
# Puntos iniciales para normalizacion
initial_points = [[10,5],[50,30],[80,15],[20,70],[60,20]]
f1_vals = np.array([f1(x) for x in initial_points])
f2_vals = np.array([f2(x) for x in initial_points])
f1_min, f1_max = f1_vals.min(), f1_vals.max()
f2_min, f2_max = f2_vals.min(), f2_vals.max()
```

Weighted sum para distintos alphas

```python
# Weighted sum para distintos alphas
alphas = np.linspace(0,1,11)
pareto_points = []

for alpha in alphas:
    def weighted(x):
        f1_val = f1(x)
        f2_val = f2(x)
        f1_norm = (f1_val - f1_min)/(f1_max - f1_min) if f1_max!=f1_min else 0
        f2_norm = (f2_val - f2_min)/(f2_max - f2_min) if f2_max!=f2_min else 0
        return alpha*f1_norm + (1-alpha)*f2_norm

    x0 = [50, 25]  # punto inicial factible
    res = minimize(weighted, x0=x0, bounds=bounds, constraints=cons,
         method='SLSQP')
    if res.success:
        pareto_points.append([res.x[0], res.x[1], -f1(res.x), f2(res.x)])

pareto_points = np.array(pareto_points)
```

Visualizar frente de Pareto

```python
# Visualizar frente de Pareto
plt.scatter(pareto_points[:,2], pareto_points[:,3], c='red', label='Frente de
    Pareto')
plt.xlabel("Retorno")
plt.ylabel("Riesgo")
plt.title("Frente de Pareto: Inversion A y B")
plt.legend()
plt.show()
```

Decisión Final

Importante: En optimización multiobjetivo la máquina solo encuentra las soluciones óptimas para un criterio ponderado.

La elección final depende de la decisión del negocio:

- ¿Más ganancia aunque haya más contaminación?
- ¿Menos contaminación aunque haya menos ganancia?
- ¿Qué tanto puedo ganar contaminando en el borde de lo que dice la ley?

El solver solo te da todos los posibles trade-offs para distintos α y calcula los valores de las variables que minimizan la función ponderada. La decisión final la tiene un humano.

¿Dudas?
¿Consultas?

Universidad de SanAndrés