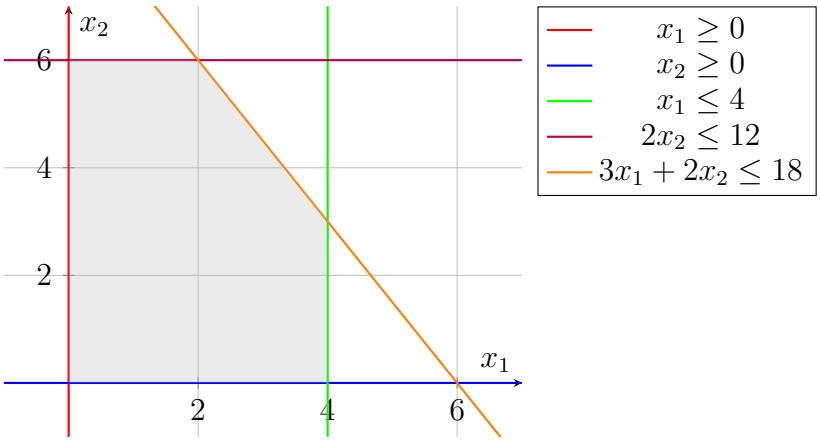
Programación No Lineal

Investigación Operativa, Universidad de San Andrés

Si encuentran algún error en el documento o hay alguna duda, mandenmé un mail a rodriguezf@udesa.edu.ar y lo revisamos.

1. Introducción a la Programación No Lineal

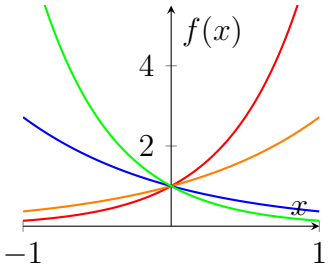
Hasta ahora veníamos viendo problemas solo de programación lineal, en donde tanto la función objetivo como las restricciones eran lineales (o dicho de forma más simple, todo son rectas). Por ejemplo, teníamos gráficos de este estilo:



Ahora vamos a tener al menos una de las dos cosas que no es lineal, que puede ser o bien la función objetivo o bien alguna de las restricciones.

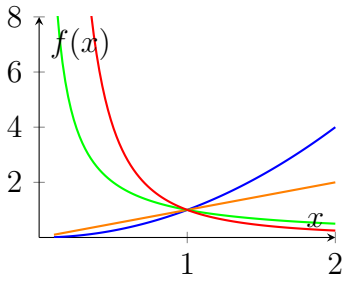
2. Convexidad

La convexidad es un concepto central en optimización. Una función es convexa si la recta que une dos puntos cualesquiera de su gráfica queda por encima o sobre la función. Esto tiene consecuencias muy importantes en problemas de optimización.



Funciones Exponenciales

$f(x) = e^{\alpha x} \quad \forall \alpha \in \mathbb{R}$



Funciones Potenciales

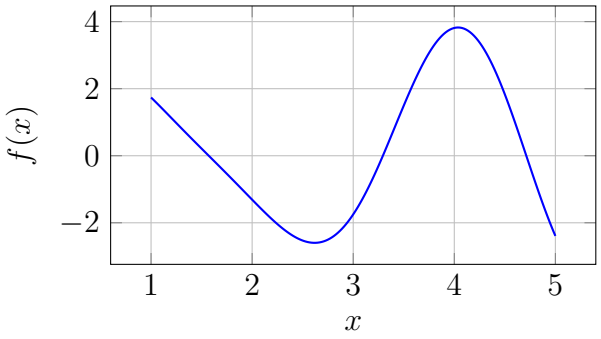
$f(x) = x^{\alpha} \qquad x \geq 0 \qquad \alpha \leq 0 \ \vee \ \alpha \geq 1$

Propiedades:

- 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 de optimización es convexo, **cualquier óptimo local es un óptimo global**. De otra forma, alcanza con encontrar *algún* mínimo local que sabremos que es global.

3. Múltiples mínimos locales

En problemas no lineales, puede haber varios mínimos locales. Depende de donde empiece llego a un mínimo diferente. Para resolver esto usamos seeds (semillas aleatorias). El resultado depende del punto inicial:



4. Ejemplos

4.1. Ejemplo 1: Asignación de presupuesto

Una empresa desea asignar su presupuesto diario de publicidad entre dos plataformas: **Google Ads** (x_1) e **Instagram Ads** (x_2). El objetivo es maximizar el impacto total de la campaña, medido como una **función no lineal del gasto** en cada plataforma, debido a efectos de saturación. La empresa dispone de \$10.000 para su inversión, y para que te dejen meter publicidades, se deben invertir al menos \$2.000 en Google Ads y \$1.000 en Instagram Ads.

La función objetivo ya la sabe la empresa y está basada en datos históricos que relacionan el gasto con el impacto:

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

Nota: La mayoría de los métodos de optimización numérica van a buscar siempre minimizar una función. Cuando querramos maximizar vamos a tener que agregarle un "-" al principio.

Las restricciones sí las tenemos que plantear nosotros, y en este caso son:

- $x_1 + x_2 \leq 10\,000$ (se dispone de \$10.000 para invertir)
- $x_1 \geq 2\,000$ (se debe invertir al menos \$2.000 en Google Ads)
- $x_2 \geq 1\,000$ (se debe invertir al menos \$1.000 en Instagram Ads)

¿Es convexo?

La función objetivo es no lineal por la presencia de los terminos exponenciales. Además, la función es estrictamente creciente y cóncava respecto a cada variable, por lo que el problema es convexo: las restricciones son lineales y la función objetivo es cóncava (como estamos maximizando, eso es lo que queremos para convexidad). Por lo tanto, cualquier optimo local es global.

4.1.1. Resolución en Python

```
1 import numpy as np
2 from scipy.optimize import minimize
3
4 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)
```

4.2. Ejemplo 2: Producción óptima de dos productos

Una empresa fabrica dos productos, **A** y **B**, con ganancias unitarias de \$40 y \$30 respectivamente. La empresa desea determinar cuántas unidades de cada producto debe fabricar por día para **maximizar su ganancia diaria**, enfrentando restricciones no lineales:

$$\text{máx } f(x_1, x_2) = 40x_1 + 30x_2$$

- **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 no lineal? ¿Convexa?

La función objetivo es lineal, pero la restriccion $x_1^2 + x_2^2 \leq 2500$ es no lineal (es una bola en el primer cuadrante) y la restriccion de compatibilidad tambien es no lineal. El conjunto factible es convexo porque ambas restricciones definen conjuntos convexos y su interseccion tambien lo es. Por lo tanto, el problema es convexo y cualquier optimo local es global.

4.3. Resolución en Python

```python
# Funcion objetivo (negativa para maximizar)
def ganancia_negativa(x):
    x1, x2 = x
    return -(40 * x1 + 30 * x2)

# Restriccion 1: x1^2 + x2^2 <= 2500
def restriccion_maquina(x):
    return 2500 - (x[0]**2 + x[1]**2)

# Restriccion 2: x1 / (x2 + 1) <= 4
def restriccion_compatibilidad(x):
    return 4 * (x[1] + 1) - x[0]

# Lista de restricciones
restricciones = [
    {'type': 'ineq', 'fun': restriccion_maquina},
    {'type': 'ineq', 'fun': restriccion_compatibilidad}
]

# Limites (x1, x2 >= 0)
bounds = [(0, None), (0, None)]

# Valor inicial factible
x0 = [1, 1]

# Optimizacion
from scipy.optimize import minimize
res = minimize(ganancia_negativa, x0, method='SLSQP', bounds=bounds,
    constraints=restricciones)

# Mostrar resultado
if res.success:
    x1_opt, x2_opt = res.x
    ganancia_max = -res.fun
    print(f'Produccion optima de A: {x1_opt:.2f} unidades')
```

```
    print(f'Produccion optima de B: {x2_opt:.2f} unidades')
    print(f'Ganancia maxima: ${ganancia_max:.2f}')
else:
    print('Error:', res.message)
```

4.4. Ejemplo 3: Optimización en biotecnología

Una startup de biotecnología busca desarrollar empaques ecológicos usando dos ingredientes clave: **fibra vegetal** (x_1) y **alga marina** (x_2), ambos en kg por lote. El costo total depende de una función periódica y lineal de ambos ingredientes, más un costo fijo:

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

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?

La función objetivo contiene términos trigonométricos $\sin(x_1) \cdot \cos(x_2)$ que no son convexos ni cóncavos en general. Además, las funciones trigonométricas pueden tener múltiples mínimos locales en el dominio considerado. Por lo tanto, este problema **NO es convexo** y puede tener múltiples mínimos locales. Por eso en el código usamos múltiples puntos iniciales aleatorios para explorar diferentes regiones del espacio de búsqueda y encontrar el mejor mínimo local.

4.4.1. Resolución en 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 encontrada: x1 = {mejor_x[0]:.4f}, x2 = {
    mejor_x[1]:.4f}')
print(f'Costo minimo estimado: {mejor_valor:.4f}')
```

5. Programación Cuadrática (QP)

5.1. Ejercicio: Asignación de presupuesto en medios (QP)

Un **estudio de televisión** está por lanzar una serie y quiere distribuir un presupuesto fijo de *100 unidades* (en miles de $) entre tres canales de difusión: **TV abierta** (x_1), **Publicidad Online** (x_2) y **Radio** (x_3).
La dirección de marketing sabe, por campañas anteriores, que:

- Cada canal aporta impacto *lineal* por unidad de presupuesto:

$$a = \begin{pmatrix} 5 \\ 4 \\ 3 \end{pmatrix} \quad \Rightarrow \quad \text{impacto lineal } = a^\top x = 5x_1 + 4x_2 + 3x_3.$$

- También hay **rendimientos decrecientes** (invertir demasiado en el mismo canal rinde menos) y **canibalización** entre canales (por ejemplo, Online y Radio se superponen en audiencia). Eso se modela con una *penalización cuadrática*

usando una matriz simétrica

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

Operativamente, el estudio debe respetar:

$$x_1 + x_2 + x_3 = 100, \qquad 10 \le x_1 \le 60, \quad 5 \le x_2 \le 60, \quad 0 \le x_3 \le 40.$$

Objetivo. Maximizar el *impacto neto*: impacto lineal menos penalización cuadrática. Para llevarlo a una QP **convexa** escribimos la versión de *minimización*:

$$\min_x \; Z(x) \;:=\; x^\top Q x \;-\; a^\top x \qquad \text{s.a. las restricciones anteriores.}$$

5.2. Resolución

5.2.1. Expandimos $x^\top Q x$ y construimos $Z(x)$

Sea $x = (x_1, x_2, x_3)^\top$. Como Q es simétrica,

$$x^\top Q x = 0{,}04\, x_1^2 + 2(0{,}01)\, x_1 x_2 + 0{,}05\, x_2^2 + 2(0{,}005)\, x_2 x_3 + 0{,}03\, x_3^2,$$

es decir,

$$\boxed{x^\top Q x = 0{,}04x_1^2 + 0{,}02x_1x_2 + 0{,}05x_2^2 + 0{,}01x_2x_3 + 0{,}03x_3^2}.$$

El término lineal es $a^\top x = 5x_1 + 4x_2 + 3x_3$. Por lo tanto:

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

Modelo QP final (función objetivo + restricciones)

$$\begin{aligned} \min_{x_1,x_2,x_3} \quad & Z(x) = 0{,}04x_1^2 + 0{,}02x_1x_2 + 0{,}05x_2^2 + 0{,}01x_2x_3 + 0{,}03x_3^2 - 5x_1 - 4x_2 - 3x_3 \\ \text{s.a.} \quad & x_1 + x_2 + x_3 = 100, \\ & 10 \le x_1 \le 60, \qquad 5 \le x_2 \le 60, \qquad 0 \le x_3 \le 40. \end{aligned}$$

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

# ------------------------------
# Datos del QP (del ejemplo)
# ------------------------------
# Impacto lineal: a^T x
a = np.array([5.0, 4.0, 3.0])  # [TV, Online, Radio]

# Penalizacion cuadratica: x^T Q x
Q = np.array([
    [0.04, 0.01, 0.00],
    [0.01, 0.05, 0.005],
    [0.00, 0.005, 0.03]
], dtype=float)

# ------------------------------
# Funcion objetivo a minimizar
# Z(x) = x^T Q x - a^T x
# (equivalente a maximizar a^T x - x^T Q x)
# ------------------------------
def Z(x):
    return float(x @ Q @ x - a @ x)

# ------------------------------
# Restricciones y cotas
# Presupuesto total: x1 + x2 + x3 = 100
# Cotas: 10 <= x1 <= 60 ; 5 <= x2 <= 60 ; 0 <= x3 <= 40
# ------------------------------
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)]

# Punto inicial factible
x0 = np.array([40.0, 30.0, 30.0])

# ------------------------------
# Optimizacion (SLSQP)
# ------------------------------
res = minimize(
    Z, x0,
    method='SLSQP',
    bounds=bounds,
    constraints=restricciones,
    options={'disp': True, 'ftol': 1e-12, 'maxiter': 1000}
```

```
)

# ------------------------------
# Reporte
# ------------------------------
if res.success:
    x1_opt, x2_opt, x3_opt = res.x
    Z_opt = res.fun
    impacto_lineal = a @ res.x
    penal_quad = res.x @ Q @ res.x
    impacto_neto = impacto_lineal - penal_quad  # = -Z_opt

    print('=== Solucion optima (QP Marketing) ===')
    print(f'x1 (TV)     = {x1_opt:.6f}')
    print(f'x2 (Online) = {x2_opt:.6f}')
    print(f'x3 (Radio)  = {x3_opt:.6f}')
    print(f'Suma        = {x1_opt + x2_opt + x3_opt:.6f}  (debe ser
    100)')
    print('---')
    print(f'Z(x*) = x^T Q x - a^T x = {Z_opt:.6f}')
    print(f'Impacto lineal a^T x    = {impacto_lineal:.6f}')
    print(f'Penalizacion x^T Q x    = {penal_quad:.6f}')
    print(f'Impacto neto (max)      = {impacto_neto:.6f}')
else:
    print('Error:', res.message)
```

6. Optimización Multiobjetivo

La optimización multiobjetivo aparece cuando querés mejorar varias cosas a la vez (ej: minimizar costo, tiempo y emisiones), pero esas metas suelen estar en conflicto. Como no existe una única "mejor" solución, se busca el frente de Pareto, que muestra todas las opciones eficientes. Para resolver estos problemas hay varias estrategias: la más simple es la suma ponderada (weighted sum), que combina todos los objetivos en una sola función con pesos; otra es el método ϵ-constraint (epsilon-constraint), donde optimizás un objetivo principal y tratás los demás como restricciones con límites; también existen métodos de programación por metas (goal programming), donde se fijan valores deseados para cada objetivo y se minimiza la desviación respecto a ellos; y en problemas más complejos se usan metaheurísticas como algoritmos genéticos, colonia de hormigas o swarm intelligence, que buscan buenas aproximaciones al frente de Pareto sin necesidad de resolver todo de manera exacta.

6.1. Frentes de Pareto

Un frente de Pareto es el conjunto de opciones donde ninguna es mejor en todo al mismo tiempo: si mejorás en un aspecto, empeorás en otro. Por ejemplo, al elegir un celular, uno puede tener mejor cámara pero ser más caro, y otro ser más barato pero con peor cámara; al elegir una ruta, una puede ser más rápida pero tener peajes, y otra más barata pero más lenta; o al elegir una dieta, una puede ser más sana pero más cara y otra más barata pero menos nutritiva. Todas esas opciones son de Pareto porque no hay una que gane en todo, y el frente es la "frontera" de donde tenés que elegir según qué priorizás.

6.1.1. Método Weighted Sum

El método de weighted sum (suma ponderada) es una forma muy simple de resolver problemas con varios objetivos al mismo tiempo: en lugar de optimizarlos por separado, se combinan todos en una sola función asignándole a cada objetivo un peso que refleja su importancia. Es decir, en lugar de buscar el máximo de cada objetivo por separado, buscamos el máximo de una función que combina todos los objetivos con sus respectivos pesos.

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

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

6.2. Normalización de Objetivos

Cuando tenemos varios objetivos y usamos weighted sum para combinarlos los valores de las funciones pueden tener escalas muy distintas. Por ejemplo, podríamos tener un costo que va en los millones, pero un tiempo que va en las decenas. Si simplemenente hicieramos $f = 0{,}5 \cdot \mathit{costo} + 0{,}5 \cdot \mathit{tiempo}$ estaríamos dando más importancia al costo que al tiempo. Para evitar esto, normalizamos los valores de las funciones dividiendo por el valor máximo que puede tomar cada objetivo.

$$f_1^{\text{norm}}(x) = \frac{f_1(x) - f_1^{\text{mín}}}{f_1^{\text{máx}} - f_1^{\text{mín}}}, \quad f_2^{\text{norm}}(x) = \frac{f_2(x) - f_2^{\text{mín}}}{f_2^{\text{máx}} - f_2^{\text{mín}}}$$

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

Donde $f_i^{\text{máx}}$ y $f_i^{\text{mín}}$ son el valor máximo y mínimo que puede tomar el objetivo i.

Problema: No sabemos cual es el valor máximo y mínimo que puede tomar cada objetivo y para eso entonces vamos a tener que probar. No podemos normalizar antes porque distorsionariamos todo. Vamos a tener que primero generar soluciones, encontrar mín/máx, normalizar y luego hacer la weighted sum.

6.2.1. Método para Normalizar

Vamos a tener que encontrar primero un conjunto de soluciones factibles, osea conjuntos de valores que cumplan con las restricciones. No hace falta probar infinitos puntos, con un barrido discreto (como decir 5 valores de cada varialbe) nos alcanza para ver el Frente de Pareto aproximado y normalizar. En casos más específicos y no dentro de un contexto pedagógico deberíamos calcular el mín/máx de manera más precisa y para esto existe hacer un *mesh grid* o un muestreo aleatorio.

Cada conjunto de puntos lo vamos a evaluar en cada función y nos vamos a traer el mínimo y máximo que tomó. El código quedaría mas o menos así:

```
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()
```

Lógicamente, en el caso de que tengamos más puntos o funciones tendríamos que adaptar el código de arriba para conseguir todo. Luego de eso ya podemos utilizar la función de normalización que vimos antes, buscando por cada alpha, y nos quedaría un código mas o menos así:

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

for i, alpha in enumerate(alphas):
    def weighted(x):
        f1_val = -f1(x)
        f2_val = f2(x)

        # Normalizacion segura para no dividir por 0
        if f1_max != f1_min:
            f1_norm = (f1_val - f1_min) / (f1_max - f1_min)
        else:
            f1_norm = 0

        if f2_max != f2_min:
            f2_norm = (f2_val - f2_min) / (f2_max - f2_min)
```

```
        else:
            f2_norm = 0

        return alpha * f1_norm + (1-alpha) * f2_norm

    x0 = initial_points[i % len(initial_points)] # para usar un x0 distinto para cada alpha y poder explorar mejor el espacio
    res = minimize(weighted, x0=x0, bounds=bounds, constraints=cons)
    if res.success:
        x_opt = res.x
        pareto_points.append([x_opt[0], x_opt[1], -f1(x_opt), f2(x_opt)])

pareto_points = np.array(pareto_points)
```

6.2.2. Visualizando el Frente de Pareto

Por último vamos a tener que visualizar el Frente de Pareto para poder entender como se comporta la función objetivo con distintos alphas, osea, el trade-off. El óptimo α nos los va a dar el solver de *minimize* que estamos sacando de *scipy*.

```
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()
```

Acá arriba hacemos [:,2] y [:,3] porque las primeras dos columnas (0 y 1) son los valores de las variables de decisión x_1 y x_2 y las últimas dos (2 y 3) son los valores de las funciones objetivo f_1 y f_2, que son los que graficamos para ver el Frente de Pareto.

6.2.3. Tomando la decisión final

Es importante notar que en optimización multiobjetivo la máquina solo va a encontrar las soluciones óptimas para un criterio ponderado; la elección final dependie de la decisión del negocio y no del algoritmo. El solver solo te va a dar todos los posibles trade-offs para distintos α y calcular los valores de las variables que minimizan la función ponderada. La decisión final la tiene un humano que se preguntara si quiere más ganancia aunque haya más contaminación, o menos contaminación aunque haya

menos ganancia (aunque a veces esto se termina resolviendo por cuestiones legislativas, y termina siendo qué tanto puedo ganar contaminando en el borde de lo que dice la ley).

6.3. Ejemplo 1: Producción con objetivos conflictivos

Una empresa fabrica dos productos A y B, y quiere optimizar simultáneamente dos objetivos conflictivos:

- **Maximizar ganancia:** $40x_1 + 30x_2$
- **Minimizar contaminación:** $x_1^2 + 2x_2^2$

Restricciones:

- $x_1 + x_2 \leq 100$ (capacidad total)
- $x_1 \geq 10$ (demanda mínima producto A)
- $x_2 \geq 5$ (demanda mínima producto B)

¿Es convexo?

La función de ganancia es lineal (convexa) pero la función de contaminación es cuadrática y convexa. El problema multiobjetivo resultante no es convexo en general, pero cada objetivo individual sí lo es.

6.4. Resolución en Python

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize

def ganancia_negativa(x):
    return -(40 * x[0] + 30 * x[1])

def contaminacion(x):
    return x[0]**2 + 2 * x[1]**2

def objetivo_combinado(x, w1, w2):
    ganancia_norm = ganancia_negativa(x) / 1000
    contaminacion_norm = contaminacion(x) / 10000
    return w1 * ganancia_norm + w2 * contaminacion_norm
```

```
15
16 restricciones = [
17     {'type': 'ineq', 'fun': lambda x: 100 - (x[0] + x[1])},
18     {'type': 'ineq', 'fun': lambda x: x[0] - 10},
19     {'type': 'ineq', 'fun': lambda x: x[1] - 5}
20 ]
21
22 bounds = [(10, None), (5, None)]
23 x0 = [50, 30]
24
25 pesos = [(0.1, 0.9), (0.3, 0.7), (0.5, 0.5), (0.7, 0.3), (0.9, 0.1)]
26 resultados = []
27
28 for w1, w2 in pesos:
29     res = minimize(lambda x: objetivo_combinado(x, w1, w2), x0,
30                    method='SLSQP', bounds=bounds, constraints=
   restricciones)
31     if res.success:
32         x1_opt, x2_opt = res.x
33         ganancia = -ganancia_negativa([x1_opt, x2_opt])
34         contam = contaminacion([x1_opt, x2_opt])
35         resultados.append((ganancia, contam, x1_opt, x2_opt, w1, w2)
   )
36
37 print("Frente de Pareto:")
38 print("Ganancia | Contaminacion | x1 | x2 | w1 | w2")
39 for gan, cont, x1, x2, w1, w2 in resultados:
40     print(f"{gan:.1f} | {cont:.1f} | {x1:.1f} | {x2:.1f} | {w1:.1f}
   | {w2:.1f}")
```

6.5. Ejemplo 2: Logística con tres objetivos

Una empresa de logística debe distribuir productos usando tres rutas diferentes, optimizando simultáneamente:

- **Minimizar costo:** $5x_1 + 3x_2 + 2x_3$
- **Minimizar tiempo:** $2x_1 + 4x_2 + 3x_3$
- **Maximizar confiabilidad:** $0{,}9x_1 + 0{,}8x_2 + 0{,}95x_3$

Restricciones:

- $x_1 + x_2 + x_3 = 50$ (total de unidades a distribuir)

- $x_i \geq 0$ para $i = 1, 2, 3$

¿Es convexo?

Todas las funciones objetivo son lineales, por lo que el problema es convexo. Sin embargo, al combinar múltiples objetivos con pesos diferentes, obtenemos diferentes soluciones óptimas.

6.6. Resolución en Python

```
def costo(x):
    return 5 * x[0] + 3 * x[1] + 2 * x[2]

def tiempo(x):
    return 2 * x[0] + 4 * x[1] + 3 * x[2]

def confiabilidad_negativa(x):
    return -(0.9 * x[0] + 0.8 * x[1] + 0.95 * x[2])

def objetivo_combinado(x, w1, w2, w3):
    costo_norm = costo(x) / 100
    tiempo_norm = tiempo(x) / 100
    conf_norm = confiabilidad_negativa(x) / 50
    return w1 * costo_norm + w2 * tiempo_norm + w3 * conf_norm

restricciones = [
    {'type': 'eq', 'fun': lambda x: x[0] + x[1] + x[2] - 50}
]

bounds = [(0, None), (0, None), (0, None)]
x0 = [20, 15, 15]

pesos = [(0.6, 0.2, 0.2), (0.2, 0.6, 0.2), (0.2, 0.2, 0.6), (0.33,
    0.33, 0.34)]
resultados = []

for w1, w2, w3 in pesos:
    res = minimize(lambda x: objetivo_combinado(x, w1, w2, w3), x0,
                   method='SLSQP', bounds=bounds, constraints=
    restricciones)
    if res.success:
        x1_opt, x2_opt, x3_opt = res.x
        cost = costo([x1_opt, x2_opt, x3_opt])
        time = tiempo([x1_opt, x2_opt, x3_opt])
        conf = -confiabilidad_negativa([x1_opt, x2_opt, x3_opt])
```

```
            resultados.append((cost, time, conf, x1_opt, x2_opt, x3_opt)
    )

print("Soluciones del frente de Pareto:")
print("Costo | Tiempo | Confiabilidad | x1 | x2 | x3")
for cost, time, conf, x1, x2, x3 in resultados:
    print(f"{cost:.1f} | {time:.1f} | {conf:.2f} | {x1:.1f} | {x2:.1
    f} | {x3:.1f}")
```

6.7. Ejemplo 3: Inversión con riesgo y retorno

Un inversor quiere distribuir \$100,000 entre tres activos, optimizando simultáneamente:

- **Maximizar retorno esperado:** $0{,}12x_1 + 0{,}08x_2 + 0{,}15x_3$
- **Minimizar riesgo:** $0{,}2x_1^2 + 0{,}1x_2^2 + 0{,}3x_3^2$

Restricciones:

- $x_1 + x_2 + x_3 = 100000$ (presupuesto total)
- $x_i \geq 0$ para $i = 1, 2, 3$

¿Es convexo?

La función de retorno es lineal (convexa) y la función de riesgo es cuadrática y convexa. El problema multiobjetivo resultante no es convexo en general, pero cada objetivo individual sí lo es.

6.8. Resolución en Python

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

def retorno_negativo(x):
    # -(0.12 x1 + 0.08 x2 + 0.15 x3)
    return -(0.12 * x[0] + 0.08 * x[1] + 0.15 * x[2])

def riesgo(x):
    # 0.2 x1^2 + 0.1 x2^2 + 0.3 x3^2
    return 0.2 * x[0]**2 + 0.1 * x[1]**2 + 0.3 * x[2]**2

```

```
# Gradientes analiticos
def grad_retorno_negativo(_x):
    # derivada de -(0.12, 0.08, 0.15) x
    return np.array([-0.12, -0.08, -0.15])

def grad_riesgo(x):
    # derivada de (0.2 x1^2, 0.1 x2^2, 0.3 x3^2)
    return np.array([0.4*x[0], 0.2*x[1], 0.6*x[2]])

def objetivo_combinado(x, w1, w2):

    ret_neg_norm = retorno_negativo(x) / 1e4
    riesgo_norm  = riesgo(x)           / 1e9
    return w1 * ret_neg_norm + w2 * riesgo_norm

def grad_objetivo_combinado(x, w1, w2):
    g_ret  = grad_retorno_negativo(x) / 1e4
    g_risk = grad_riesgo(x)           / 1e9
    return w1 * g_ret + w2 * g_risk

# ------------------------------
# Restriccion y cotas
# ------------------------------
presupuesto = 100_000.0
restricciones = [{
    'type': 'eq',
    'fun': lambda x: x[0] + x[1] + x[2] - presupuesto,
    'jac': lambda x: np.array([1.0, 1.0, 1.0])
}]
bounds = [(0.0, None), (0.0, None), (0.0, None)]

# Punto inicial factible
x0 = np.array([40_000.0, 30_000.0, 30_000.0])

# ------------------------------
# Barrido de pesos
# ------------------------------
pesos = [(0.1, 0.9), (0.3, 0.7), (0.5, 0.5), (0.7, 0.3), (0.9, 0.1)]
resultados = []

for w1, w2 in pesos:
    res = minimize(
        fun=lambda x: objetivo_combinado(x, w1, w2),
        x0=x0,
        method='SLSQP',
        jac=lambda x: grad_objetivo_combinado(x, w1, w2),
```

```
        bounds=bounds,
        constraints=restricciones,
        options={'disp': False, 'ftol': 1e-12, 'maxiter': 2000}
    )
    if not res.success:
        print(f"Fallo para (w1,w2)=({w1},{w2}):", res.message)
        continue

    x1, x2, x3 = res.x
    ret  = -(0.12*x1 + 0.08*x2 + 0.15*x3)
    risk =  0.2*x1**2 + 0.1*x2**2 + 0.3*x3**2
    resultados.append((w1, w2, -ret, risk, x1, x2, x3))

print("Frente de Pareto - Inversion:")
print("w1  w2  | Retorno   | Riesgo      | x1       | x2       | x3")
for w1, w2, ret, risk, x1, x2, x3 in resultados:
    print(f"{w1:.1f} {w2:.1f} | ${ret:9.0f} | {risk:10.0f} | ${x1:7.0f} | ${x2:7.0f} | ${x3:7.0f}")
```