# Programación no lineal

Rodrigo Maranzana

# Repaso: clasificaciones generales

- Problema con restricciones.
- Problema sin restricciones.

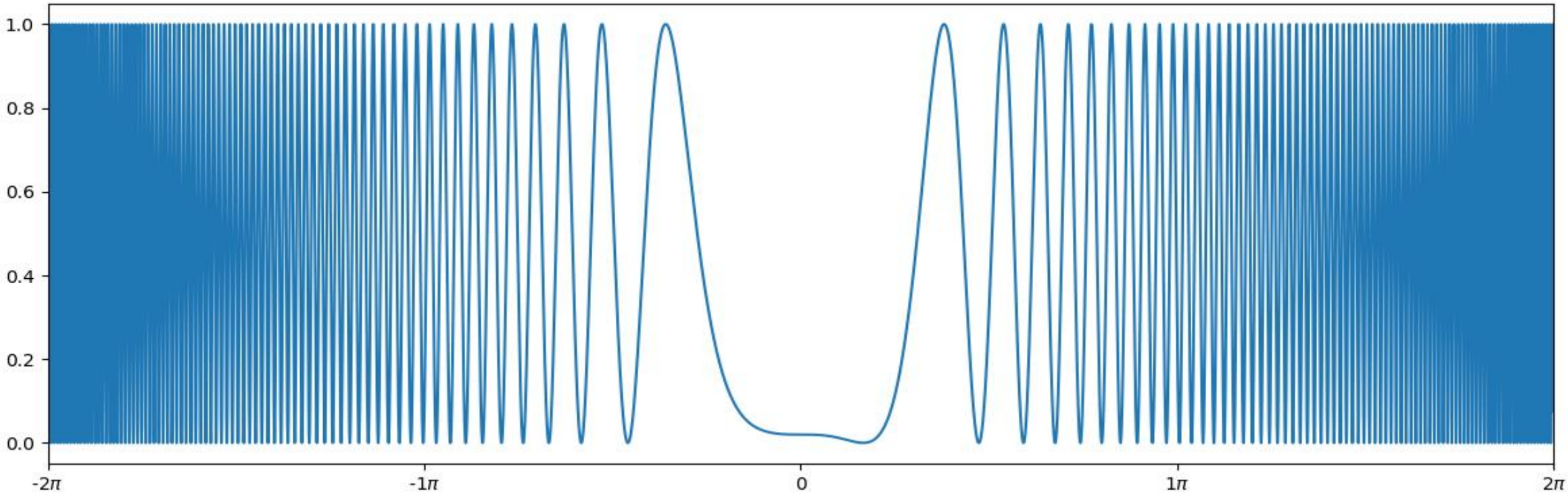
- Problema continuo.
- Problema discreto.

- Programación lineal.
- Programación no lineal: cuadrática, mixta, entera, …

# Repaso: programación matemática

Min $sin(X^3 + 3)^2$

Unconstrained Non-Linear Optimization

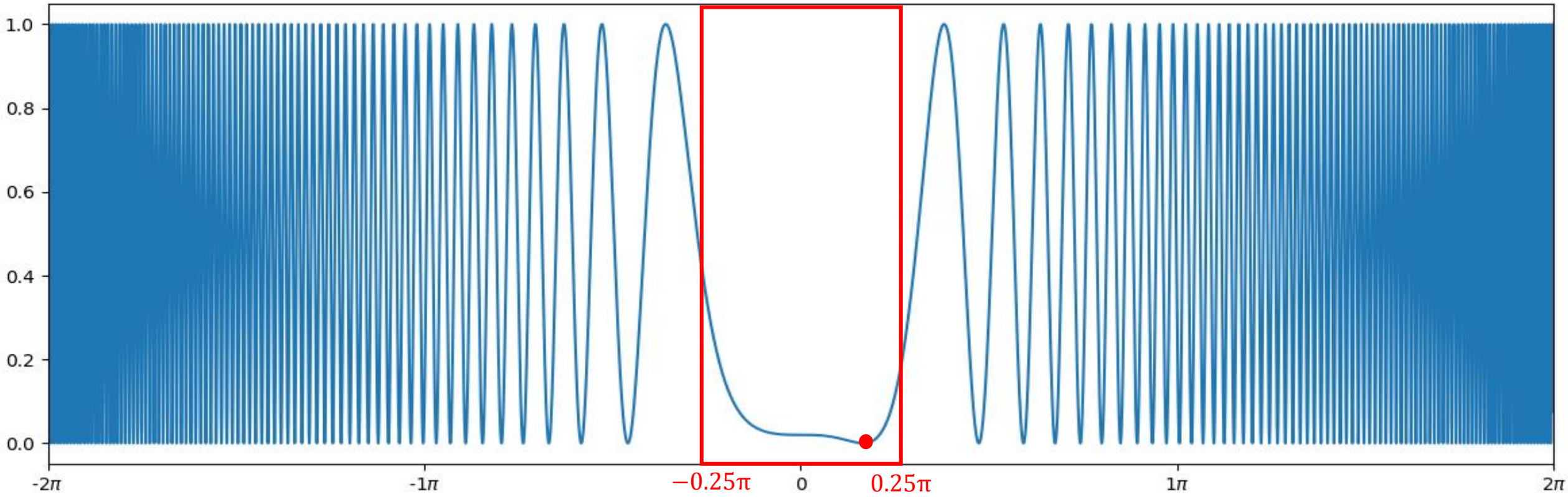


# Repaso: programación matemática

Min $sin(X^3 + 3)^2$

s.t $-0.25\pi \leq X \leq 0.25\pi$

Box-Constrained Non-Linear Optimization

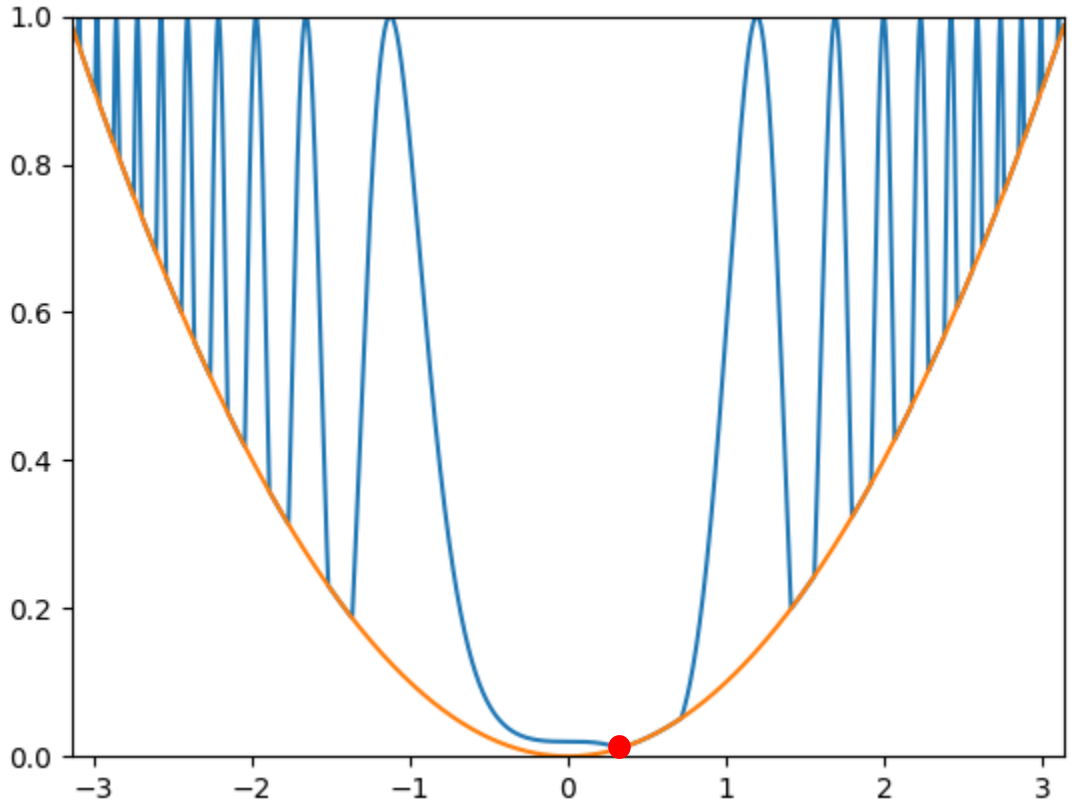


# Repaso: programación matemática

Min $sin(X^3 + 3)^2$

s.t $objetivo \geq 0.1 * X^2$

Constrained Non- Linear Optimization



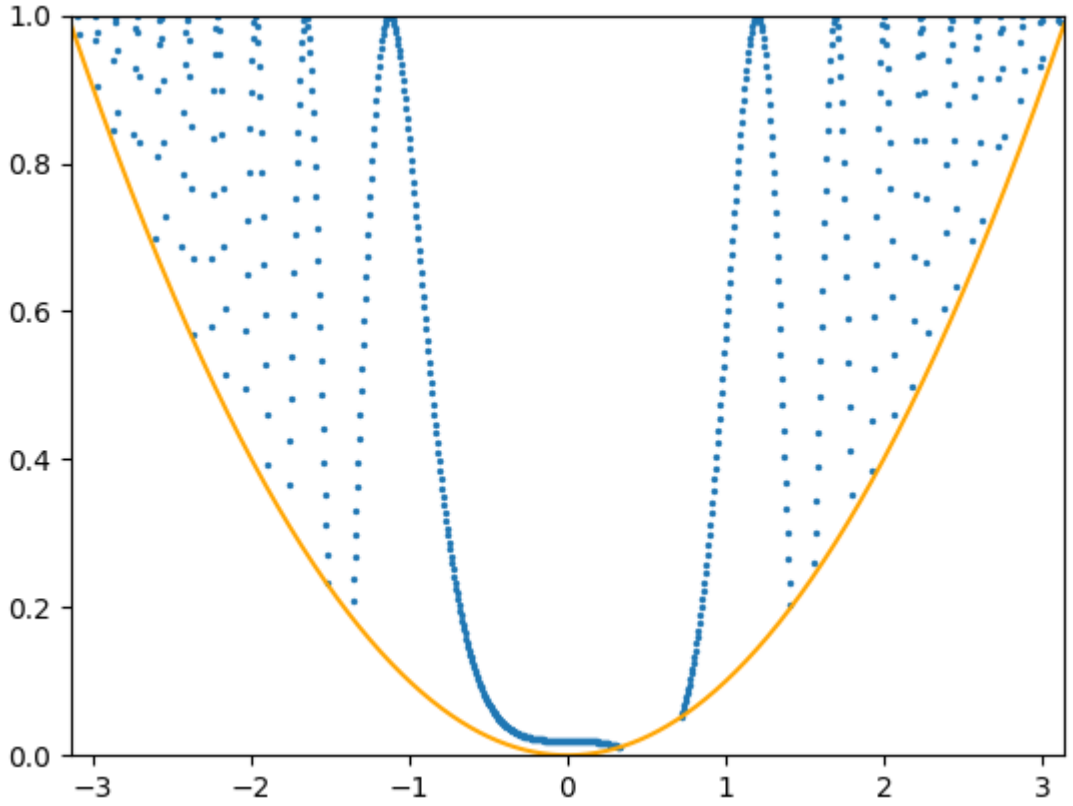
# Repaso: programación matemática

Min $sin(X^3 + 3)^2$

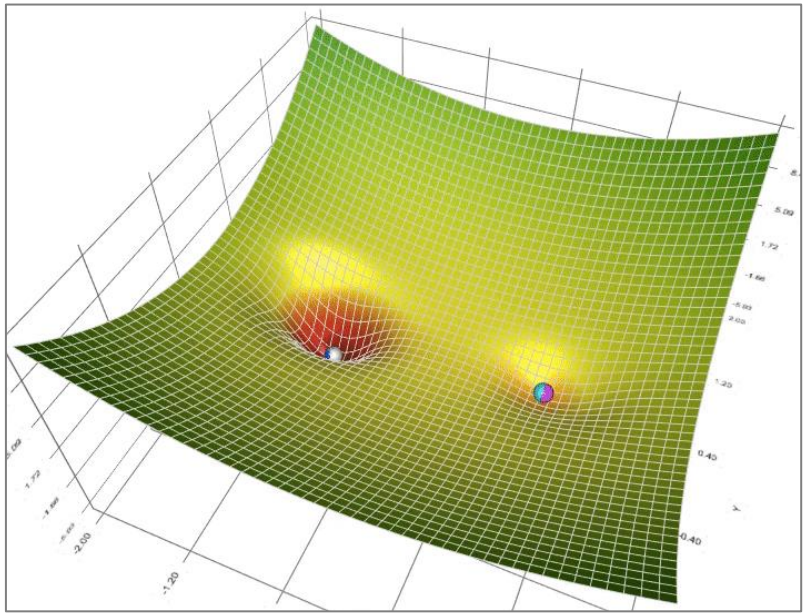
s.t $objetivo \geq X^2$

X equiespaciado

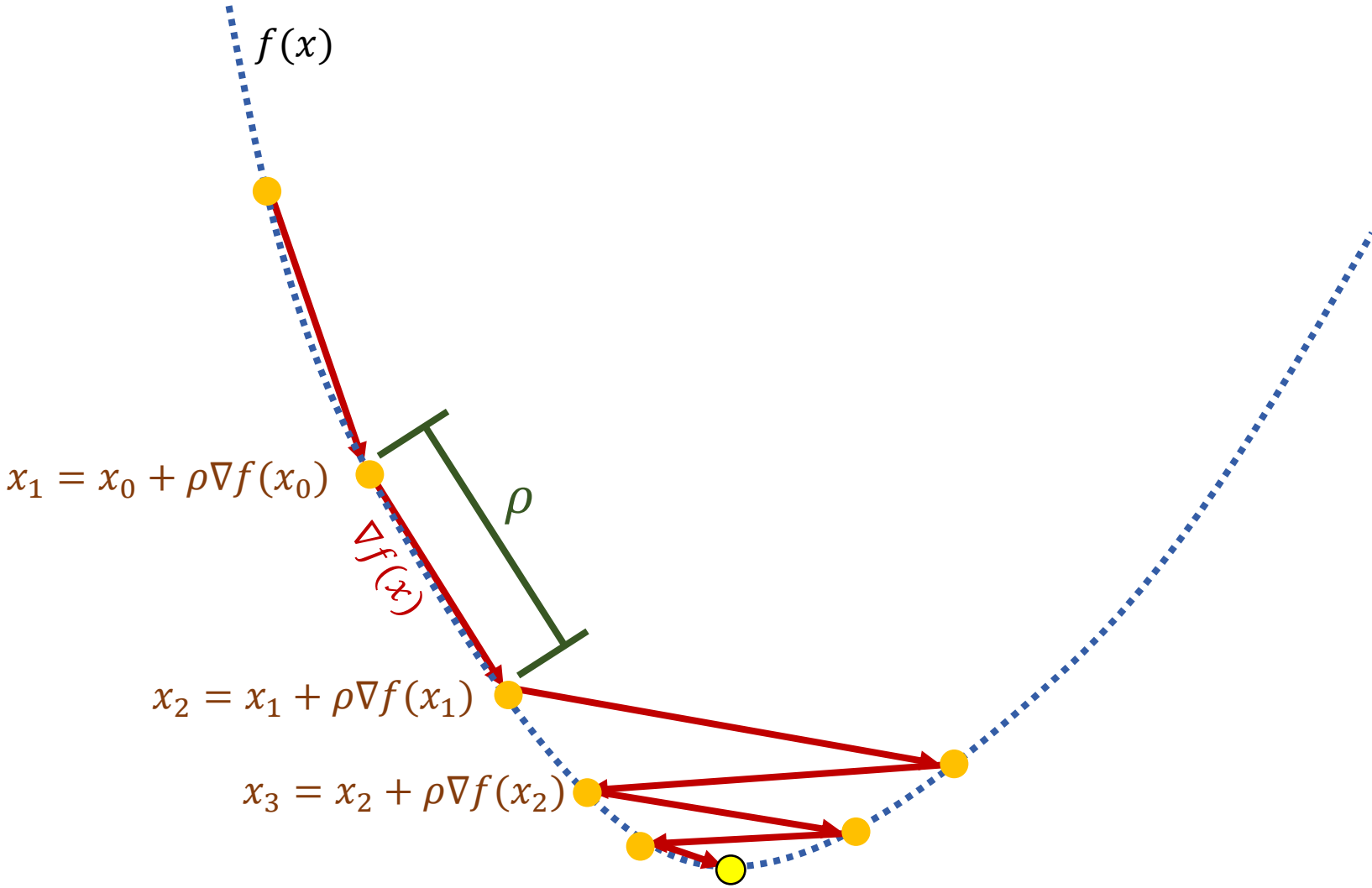
Constrained Non-Linear Optimization



# Solver: método del gradiente descendiente



Fuente: *https://towardsdatascience.com/a-visual-explanation-of-gradient-descent-methods-momentum-adagrad-rmsprop-adam-f898b102325c*



# Solver: método del gradiente descendiente

El método del gradiente, busca en la dirección del gradiente ($\nabla f(x_i)$) de la función, con un paso determinado $\rho$.

Iteraciones:

$$x_1 = x_0 + \rho \nabla f(x_0)$$
$$x_2 = x_1 + \rho \nabla f(x_1)$$
$$\dots$$
$$x_i = x_{i-1} + \rho \nabla f(x_{i-1})$$

Se busca: $f(x_i) < f(x_{i-1})$

El algoritmo finaliza cuando $\nabla f(x_i) \approx 0$, es decir, no existe posibilidad de mejora.

No se asegura el óptimo global, puede quedar estancado en óptimos locales.

# Ejemplo: programación no lineal box-constrained

La fabricación de un producto tiene un costo variable de usd 4.30.

El departamento comercial sugiere precios de venta (P1, P2, P3) al público. Para cada uno se estima la demanda en tres regiones de venta.

Entre esos precios, la demanda se interpola lineal. Formándose una curva piecewise de precio-demanda.

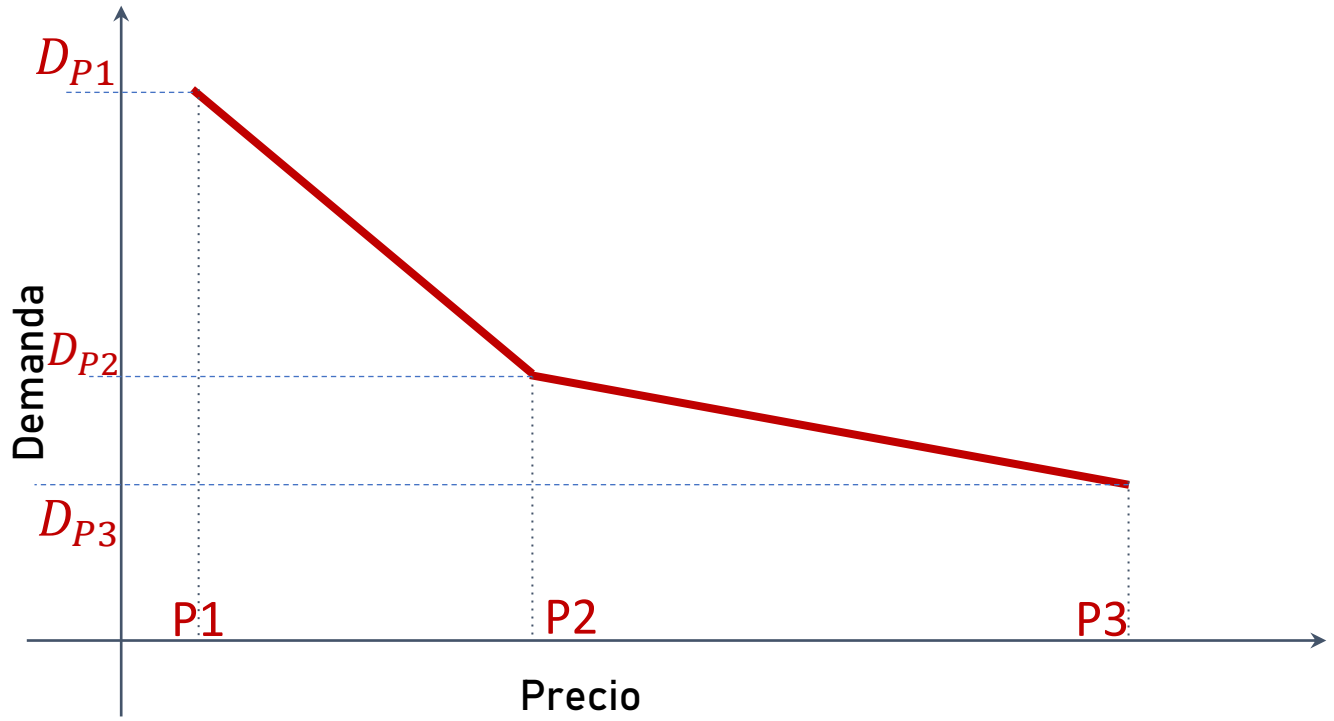
| Precio (usd) | Córdoba (miles u) | Buenos Aires (miles u) | Santa Fe (miles u) |
|---|---|---|---|
| Bajo (5.20) | 40 | 33 | 22 |
| Medio (5.45) | 31 | 28 | 16 |
| Alto (5.74) | 25 | 13 | 10 |

¿Qué precio debe colocarse para maximizar la ganancia?

*Basado en el ejercicio "24 Pricing a Candy Bar", Winston (2004) Operations Research: Applications and Algorithms*

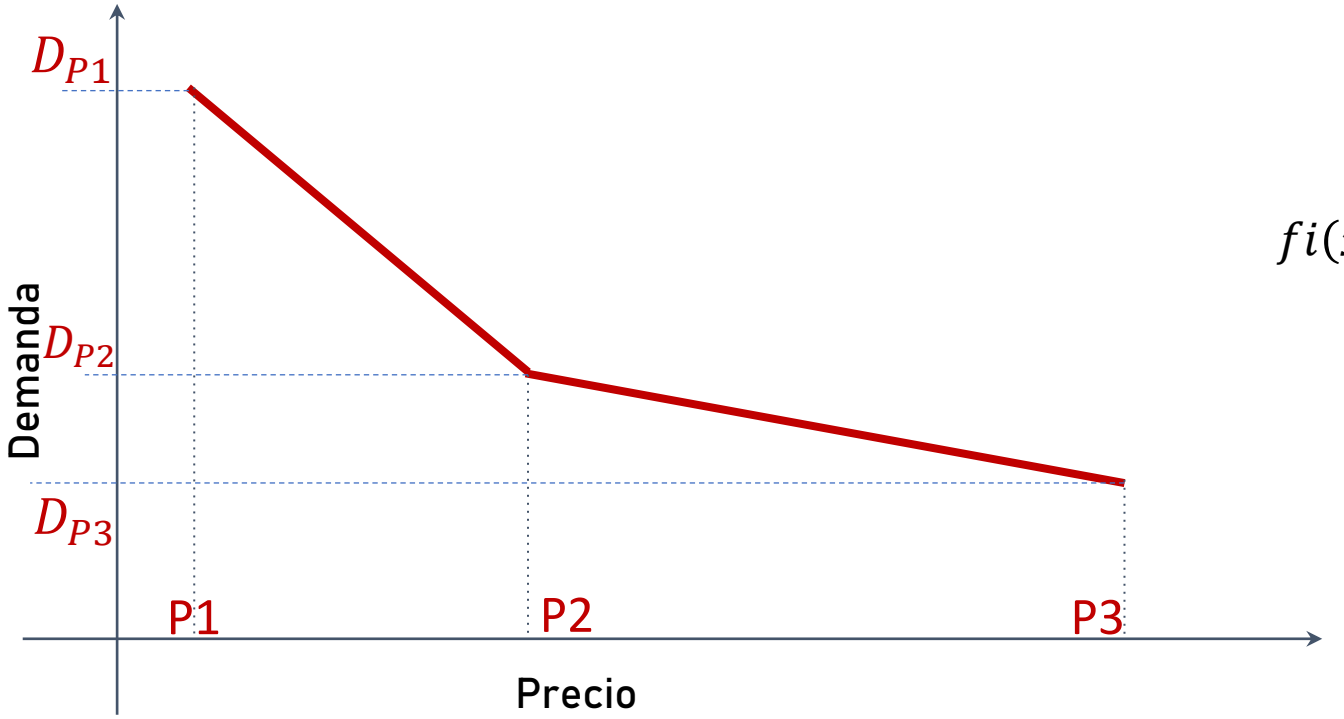
# Curva de precio-demanda

Para cada región:




| Precio (usd) | Córdoba (miles u) | Buenos Aires (miles u) | Santa Fe (miles u) |
|---|---|---|---|
| Bajo (5.20) | 40 | 33 | 22 |
| Medio (5.45) | 31 | 28 | 16 |
| Alto (5.74) | 25 | 13 | 10 |

# Construcción de función objetivo




Para cada región "i":

$$fi(x) = \begin{cases} \frac{\Delta Di_{P2-P1}}{P2-P1}(x - P1) + Di_{P1} & si\ P1 \leq x \leq P2 \\ \frac{\Delta Di_{P3-P2}}{P3-P2}(x - P2) + Di_{P2} & si\ P2 \leq x \leq P3 \end{cases}$$

pendiente desfase ordenada

# Modelo de optimización

$$Max\ Z = (x - cv)\ (f_{Buenos\ Aires} + f_{Cordoba} + f_{Santa\ Fe})$$

precio costo variable cantidad

$$Siendo\ fi(x) = \begin{cases} \frac{\Delta Di_{P2-P1}}{P2-P1}(x - P1) + Di_{P1} & si\ P1 \leq x \leq P2 \\ \frac{\Delta Di_{P3-P2}}{P3-P2}(x - P2) + Di_{P2} & si\ P2 \leq x \leq P3 \end{cases}$$

**s.t.**

$$P1 \leq x \leq P3$$

Box Constrained

# Modificación para Python scipy.optimize

$$Min\ Z = -(x - cv)\ (f_{Buenos\ Aires} + f_{Cordoba} + f_{Santa\ Fe})$$

scipy.optimize solo permite minimización

$$Siendo\ fi(x) = \begin{cases} \frac{\Delta Di_{P2-P1}}{P2 - P1}(x - P1) + Di_{P1} & si\ P1 \leq x \leq P2 \\ \frac{\Delta Di_{P3-P2}}{P3 - P2}(x - P2) + Di_{P2} & si\ P2 \leq x \leq P3 \end{cases}$$

**s.t.**

$$P1 \leq x \leq P3$$

# Resolución con Python

```python
from scipy.optimize import minimize

# Parámetros
## Demanda:
region1 = {"low": 40, "med": 31, "high": 25}
region2 = {"low": 33, "med": 28, "high": 13}
region3 = {"low": 22, "med": 16, "high": 10}

## Precio:
p = [5.20, 5.45, 5.74]

## Costo:
cv = 4.23

regiones = [region1, region2, region3]
```

# Resolución con Python

```python
# Función piecewise por region:
def fi(x, region):
    if x ⩽ p[1]:
        return (region["med"] - region["low"]) / (p[1] - p[0]) * (x - p[0]) + region["low"]
    else:
        return (region["high"] - region["med"]) / (p[2] - p[1]) * (x - p[1]) + region["med"]
```

$$Siendo\ fi(x) = \begin{cases} \frac{\Delta Di_{P2-P1}}{P2-P1}(x - P1) + Di_{P1} & si\ P1 \leq x \leq P2 \\ \frac{\Delta Di_{P3-P2}}{P3-P2}(x - P2) + Di_{P2} & si\ P2 \leq x \leq P3 \end{cases}$$

# Resolución con Python

```python
# Función de optimización:
def f(x):
    return - (x - cv) * sum(fi(x, region_i) for region_i in regiones)

# Punto de inicio de búsqueda:
x0 = p[0]

# Optimización:
res = minimize(fun=f, x0=x0, bounds=[(p[0], p[2])])
```

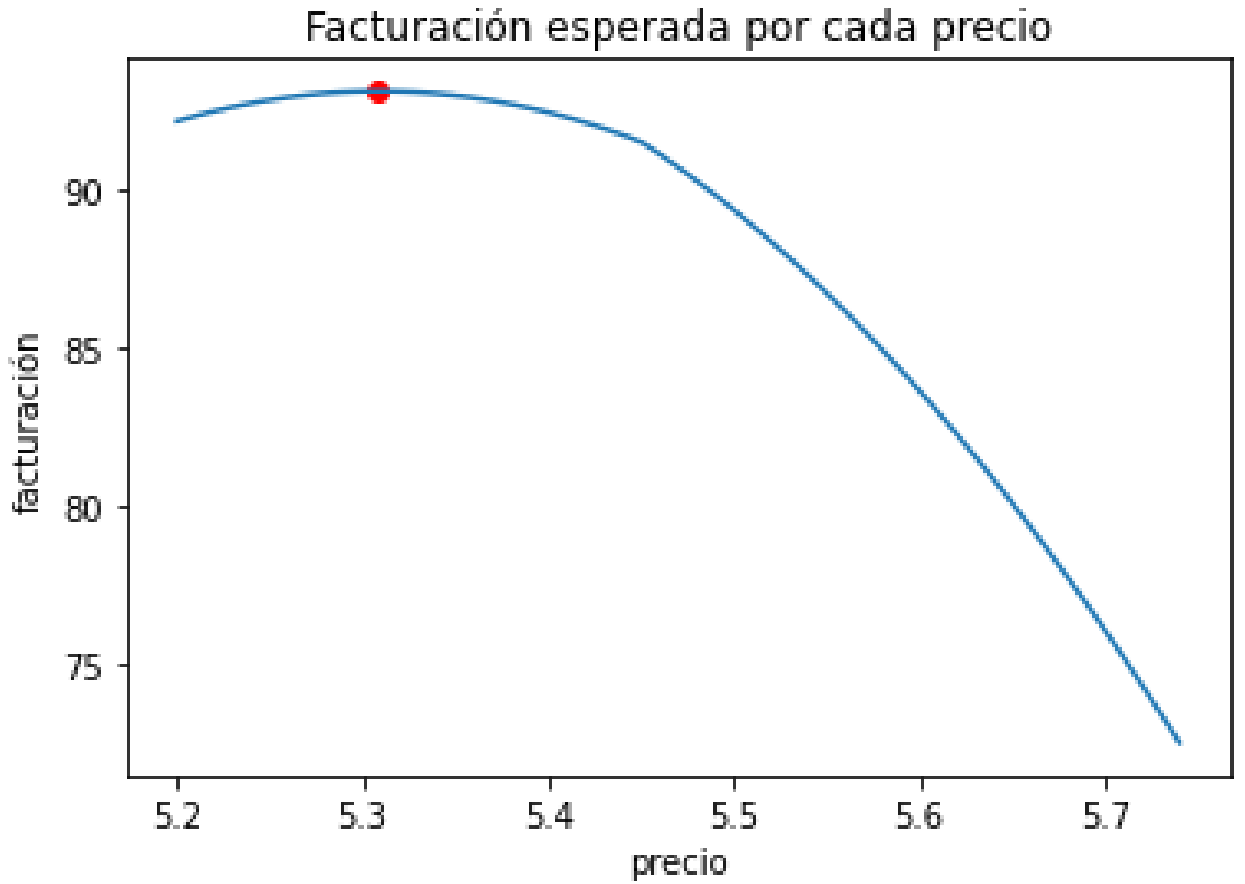
$$Min\ Z = -(x - cv)\ (f_{Buenos\ Aires} + f_{Cordoba} + f_{Santa\ Fe})$$

**s.t.**

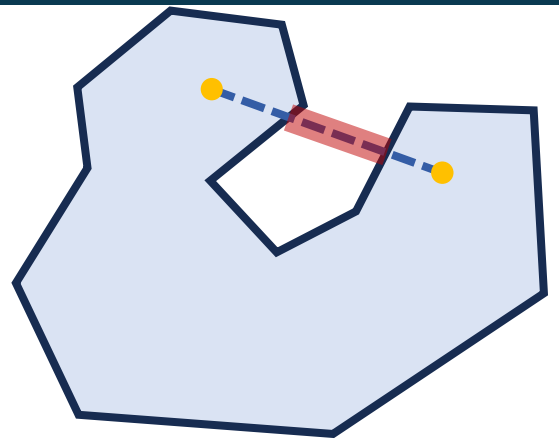
$$P1 \leq x \leq P3$$

# Resolución con Python

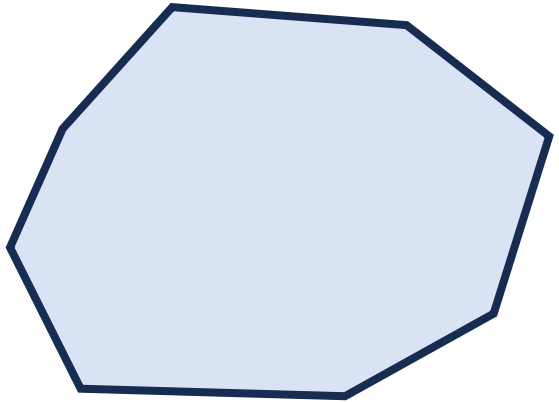
```
# Salida:
print(res)

>>       fun: -93.09612499999997
>> hess_inv: <1×1 LbfgsInvHessProduct with dtype=float64>
>>       jac: array([1.42108548e-06])
>>   message: 'CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_≤_PGTOL'
>>      nfev: 8
>>       nit: 2
>>      njev: 4
>>    status: 0
>>   success: True
>>         x: array([5.30875])
```



# Optimización convexa: conceptos



Conjunto no convexo



Conjunto convexo

## Conjunto convexo:

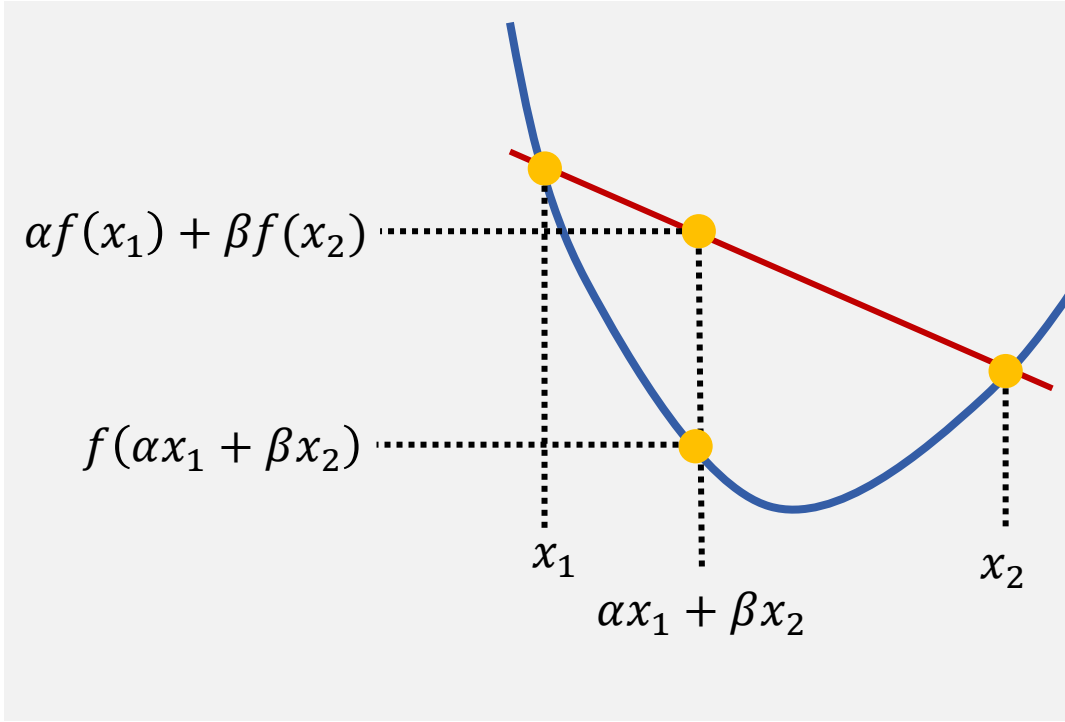
Un conjunto es convexo si la línea que une cualquier par de puntos dentro del conjunto, también pertenece al conjunto.

## Función convexa:

Toda función donde se cumple:

$$f(\alpha x_1 + \beta x_2) \leq \alpha f(x_1) + \beta f(x_2)$$

Para todos $x_1, x_2 \in \mathbb{R}^n$ y $\alpha + \beta = 1$




Función convexa

# Optimización convexa

Un problema de optimización convexa tiene la forma:

$$\begin{aligned} &Min\ f_{obj}(x) \\ &s.t.\ f_i(x) \leq b_i \quad \forall i \end{aligned}$$

Donde todas las funciones que componen el sistema $(f_{obj}(x),\ f_i(x))$ son convexas.

Además, la región de factibilidad es un conjunto convexo.

# Optimización convexa

- Todos los problemas de programación lineales son convexos.
- Los problemas no lineales pueden ser convexos o no convexos.

- Los problemas convexos no tienen solución analítica, pero existen métodos muy eficientes para resolverlos.

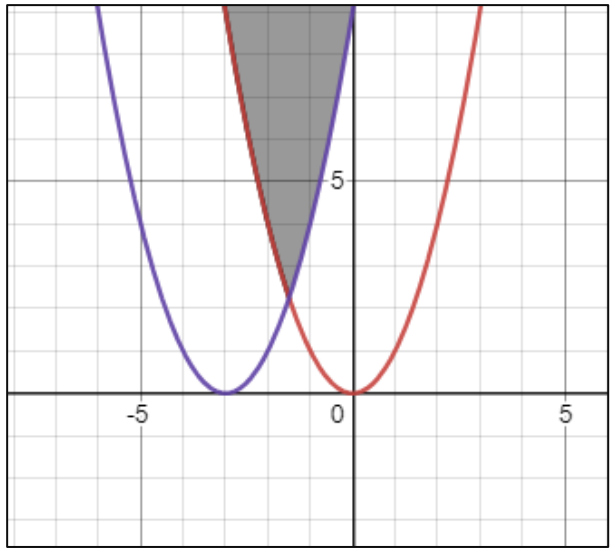
La importancia de llegar a un problema convexo radica en la eficiencia y precisión para resolverlo.

# Optimización convexa

Existen reglas para determinar si un problema es convexo.

Por ejemplo, una de ellas: "el máximo entre dos funciones convexas es una función convexa"

$$\max\{{x_1}^2, (x_1 + 3)^2\}$$




En los soft de optimización convexa, existe un check automático de convexidad.

Siguen las reglas de programación convexa: *DCP (disciplined convex programming) ruleset.*

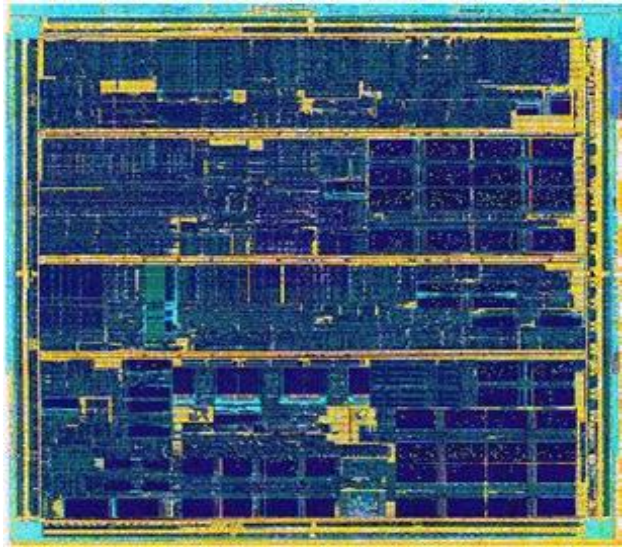
Se pueden encontrar en la página:

*http://cvxr.com/cvx/doc/dcp.html*

# Floorplanning

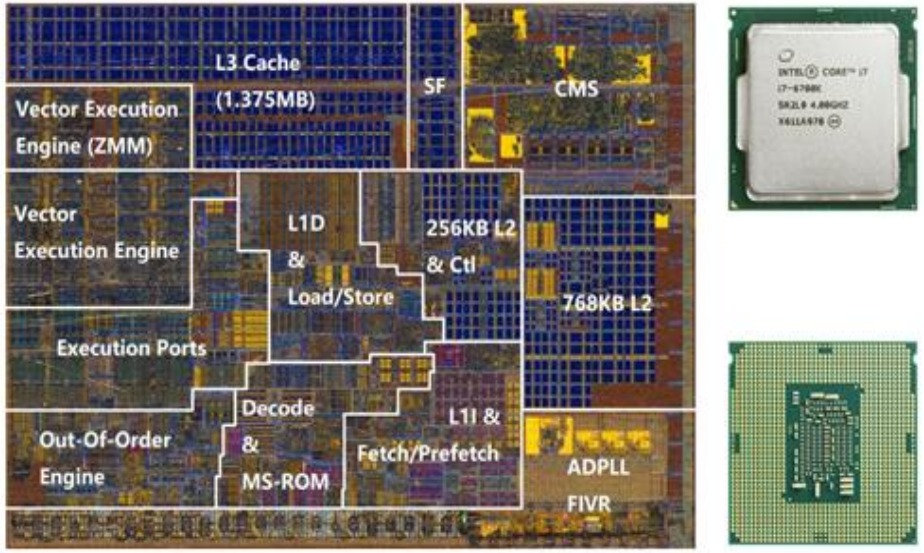
## Floorplanning es un proceso de diseño de optimización de particiones relacionado con microelectrónica.

Layout de un chip



Adya y Markov (2004), “Fixed-outline floorplanning: Enabling Hierarchical design”

Intel i7 Skylake floorplan






https://limsk.ece.gatech.edu/course/ece6133/slides/floorplanning.pdf

# Floorplanning y distribución de plantas industriales

En el campo de distribución de plantas industriales, floorplanning implica optimizar el diseño de un layout de planta sujeto a restricciones.

Algunos ejemplos de restricciones:

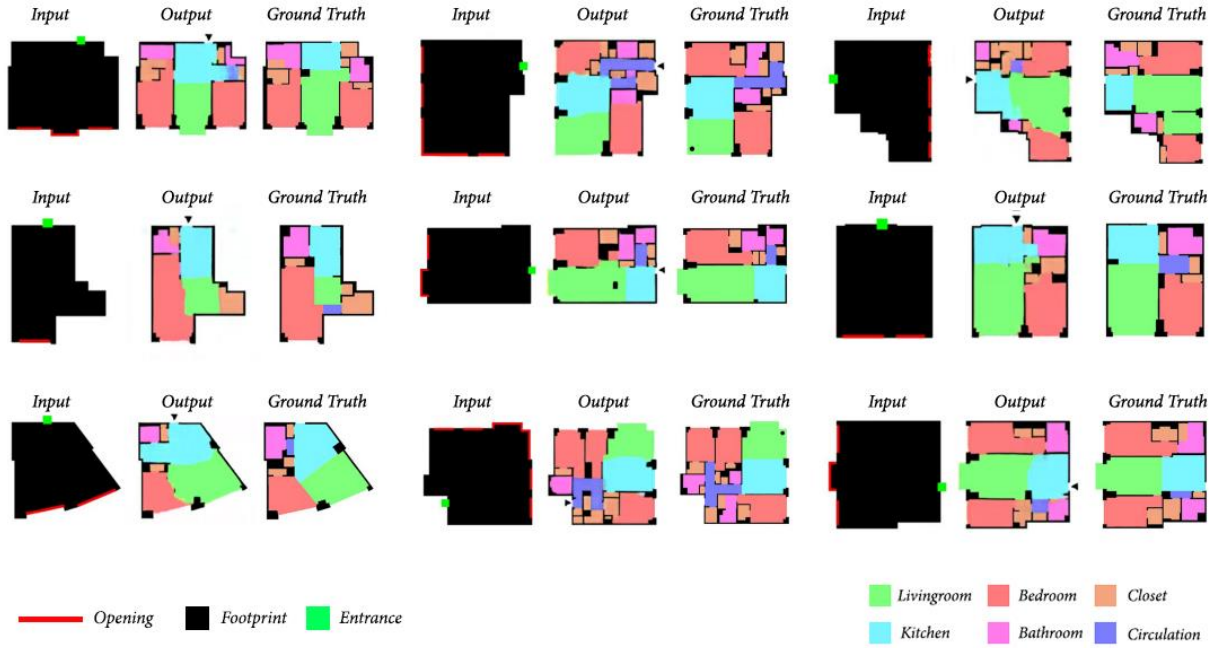
- Flujo de recursos entre áreas o procesos.
- Procesos que deben cumplir requerimientos espaciales. Ej: ancho de zonas de circulación para elevadores.
- Procesos con restricciones de vecindad. Ej: calderas y almacenes con alta carga de fuego.

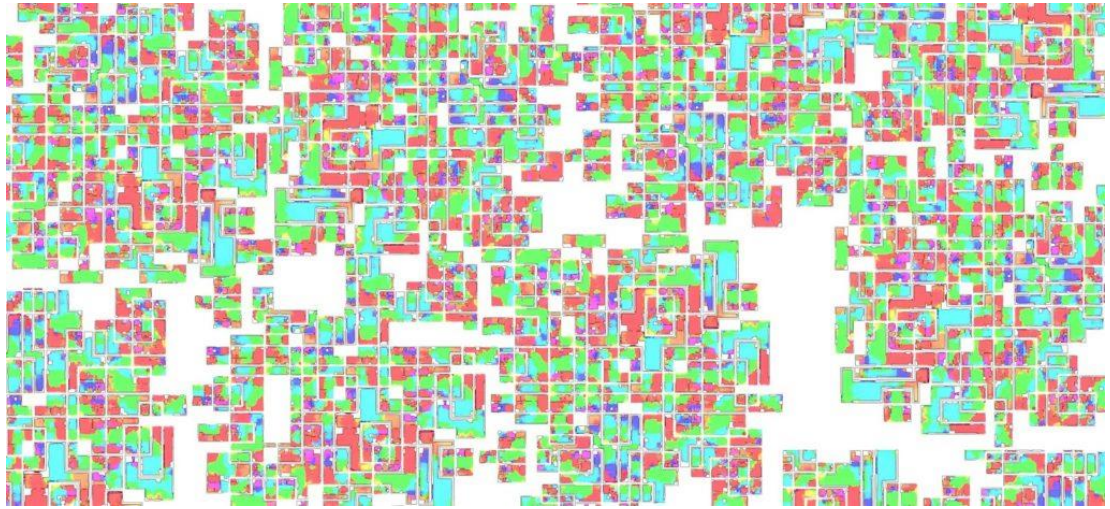
# Nuevas tendencias en Floorplanning

## Nuevas tendencias en optimización de floorplan con Generative AI:

- Modelo se entrena con datasets de distribuciones de planta conocidas.
- Infiere sobre casos no vistos anteriormente.

Nvidia (2019) “ArchiGAN: a Generative Stack for Apartment Building Design”



Fuente: https://developer.nvidia.com/blog/archigan-generative-stack-apartment-building-design/

# Floorplanning como modelo convexo

Si bien el modelo general de floorplanning es no lineal, no convexo, cumpliendo condiciones como:

- Las áreas no pueden solaparse y se consideran rectangulares.
- Se optimiza por área o perímetro.

Es un problema combinatorio muy complejo.

Se puede llegar a una solución convexa, específicamente de programación cuadrática, imponiendo posiciones relativas entre áreas.

Stanford (2009) 8.8 Floor planning (https://web.stanford.edu/~boyd/cvxbook/bv_cvxbook.pdf)

# Ejemplo: modelo de floorplanning convexo

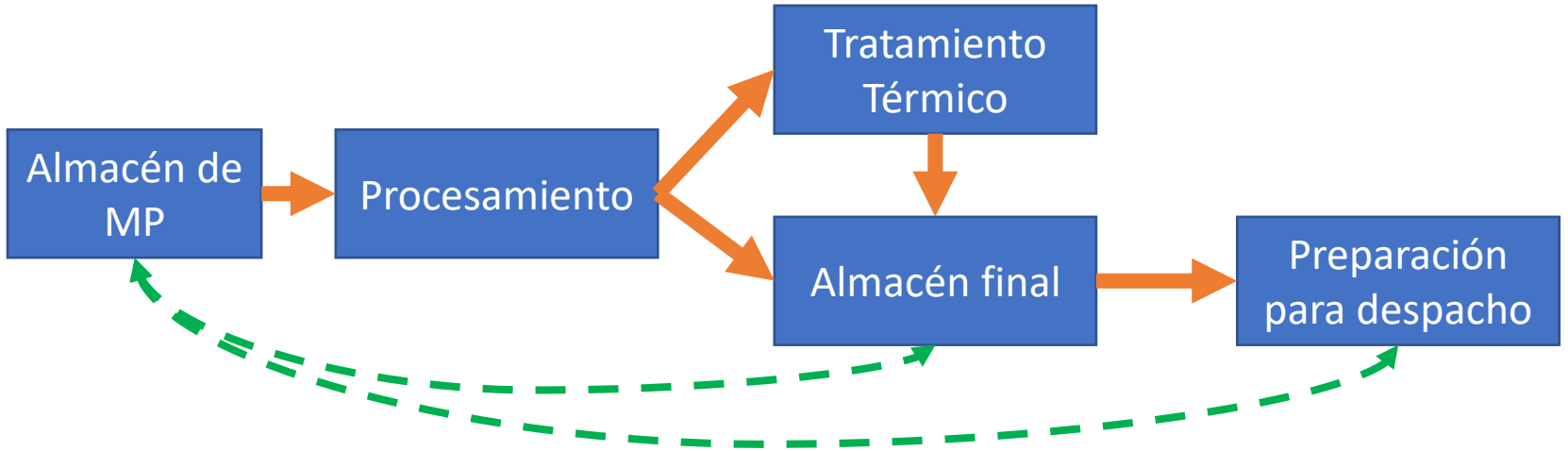
Una empresa metalúrgica planea la construcción de una nueva planta. La misma debería contar con 5 áreas de manufactura que necesitan un mínimo de superficie para poder contener al personal y maquinaria de forma segura y productiva:

- Área de almacenamiento de materia prima, 350 m2
- Área de procesamiento: corte, trefilado y punzonado, 300 m2
- Área de tratamiento térmico, 200 m2
- Área de enfriamiento y almacenaje de producto final, 300 m2
- Preparación para despacho, 300m2

Se busca determinar la posición, largo y ancho de cada área minimizando el perímetro total de la planta.

# Ejemplo: modelo de floorplanning convexo

Las áreas no pueden construirse de cualquier forma, ya que algunas deben estar comunicadas. El flujo productivo puede describirse en el grafo en naranja y comunicación directa adicional en verde.



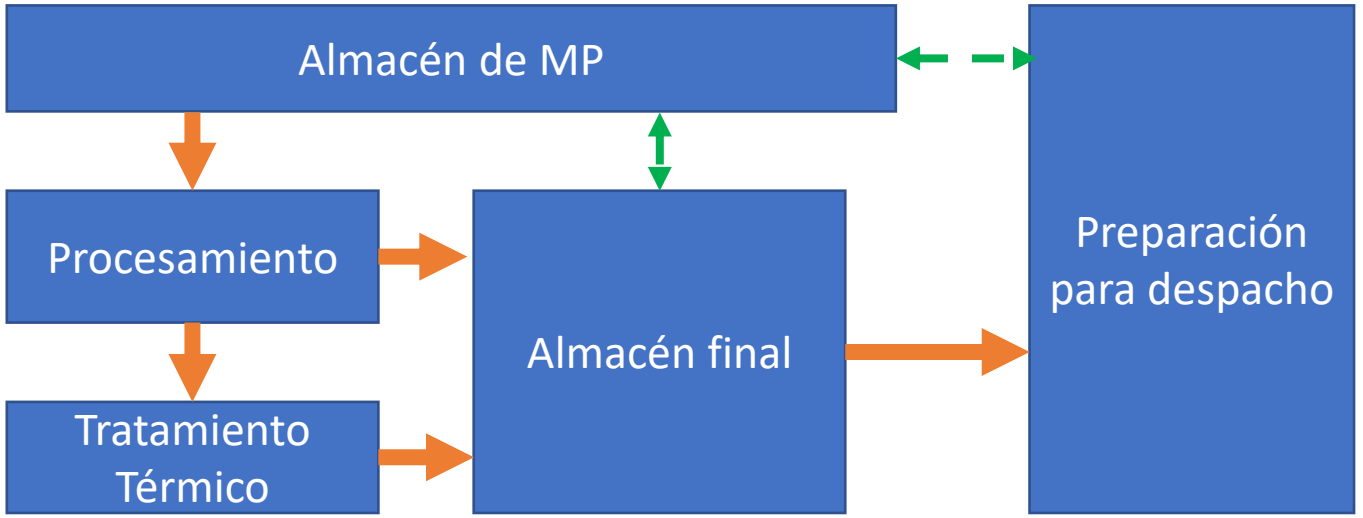

- Como podemos ver, no todos los productos se someten a tratamiento térmico.
- Todas las áreas logísticas están comunicadas.

# Ejemplo: modelo de floorplanning convexo

- Para saber la ubicación de cada área, es necesario transformar el grafo anterior, a uno geográfico con arcos verticales y horizontales. Los arcos no pueden cortarse.

- Si bien este es un problema de optimización adicional, escapa a la materia y vamos a resolverlo intuitivamente (pueden buscar sus propias soluciones).

- Conocer o suponer la posición relativa de las áreas, permite que este problema combinatorio pueda resolverse con un Modelo de Optimización Convexa.



# Ejemplo: modelo de floorplanning convexo

Condiciones:

- No existen restricciones fijas de largo y ancho.
- Hay que cumplir un mínimo y máximo de relación de aspecto (largo:ancho); máximo 5:1 y mínimo 1:5
- Alrededor de cada área debe existir un pasillo de circulación de 2,5 metros.

Si bien conocemos la dependencia geográfica de cada área, desconocemos cada ancho, largo y posición exacta.

Este es nuestro problema de Floorplanning.

# Ejemplo: modelo de floorplanning convexo

## Sets:

i: áreas {0, 1, 2, 3, 4}  Almacenamiento inicial, procesamiento, tratamiento térmico, almacenamiento final, despacho.

## Variables de decisión:

- $x_i$: Coordenada x en el plano del vértice inferior izquierdo del área i.
- $y_i$: Coordenada y en el plano del vértice inferior izquierdo del área i.
- $w_i$: Ancho del área i.
- $h_i$: Largo del área i.
- W: Ancho total de la planta
- H: Largo del área i.

## Parámetros:

- $\rho$: espacio entre áreas, pasillos de circulación.
- $\gamma_{max}$: relación de aspecto máx.
- $\gamma_{max}$: relación de aspecto mín.

# Ejemplo: modelo de floorplanning convexo

**Función objetivo:**

$$Min\ W + H$$

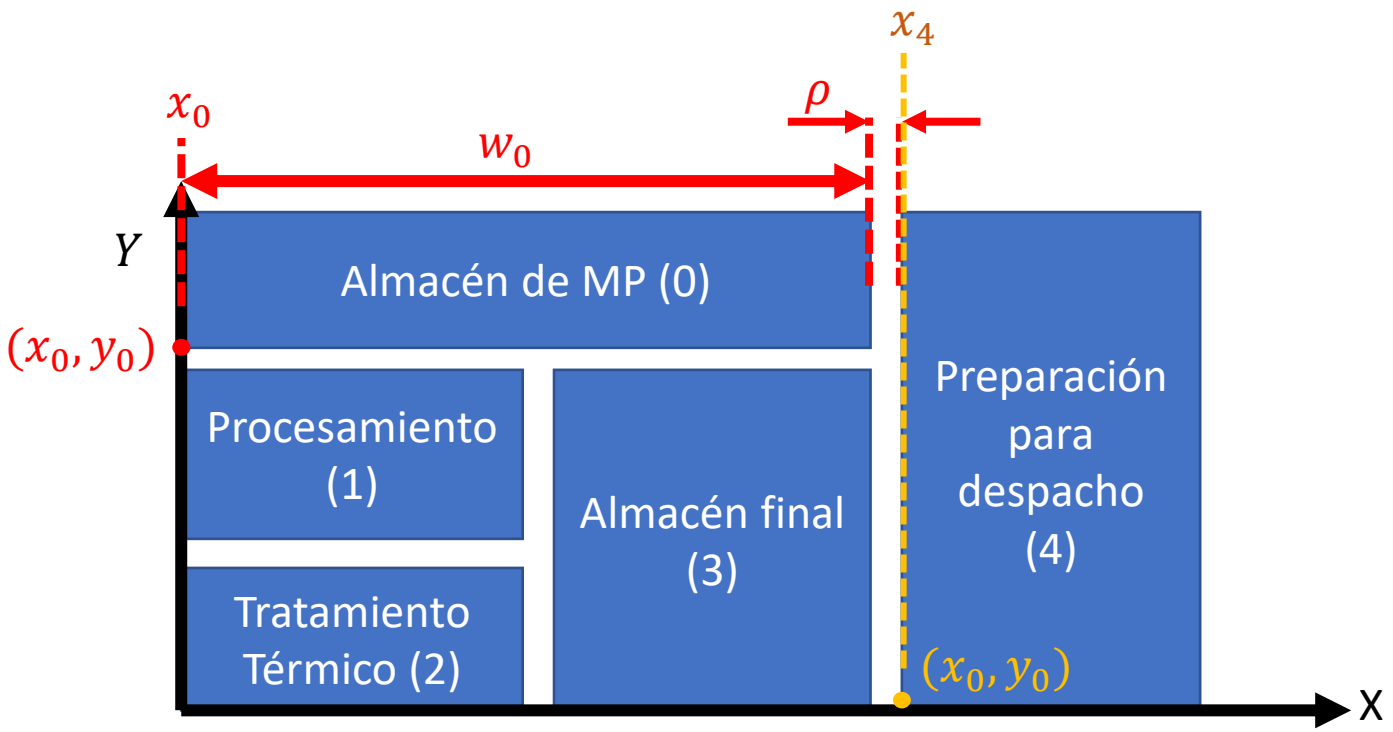
Minimizar perímetro de la planta $2(W + H)$.

Dado que la constante 2 no altera el problema de optimización, la eliminamos.

# Ejemplo: modelo de floorplanning convexo

## Restricciones de posición relativa de áreas:

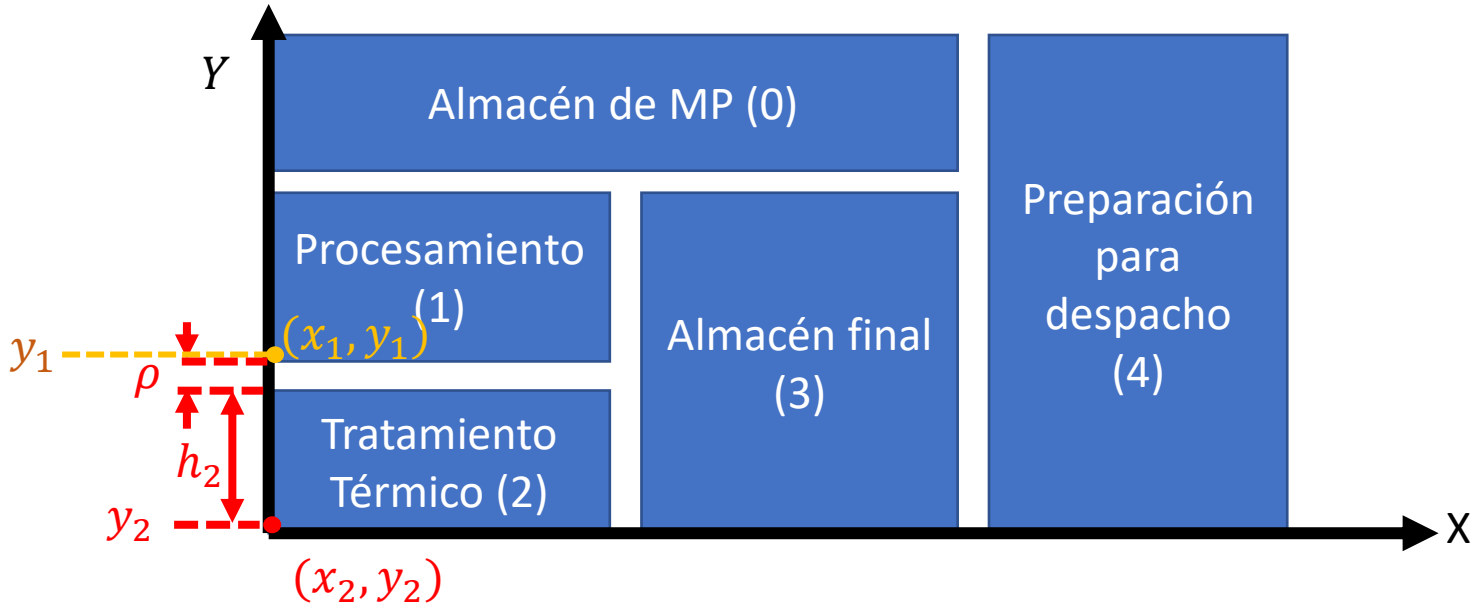
## Ejemplo eje x: $x_0 + w_0 + \rho \leq x_4$



# Ejemplo: modelo de floorplanning convexo

## Restricciones de posición relativa de áreas:

## Ejemplo eje y: $y_2 + h_2 + \rho \leq y_1$



# Ejemplo: modelo de floorplanning convexo

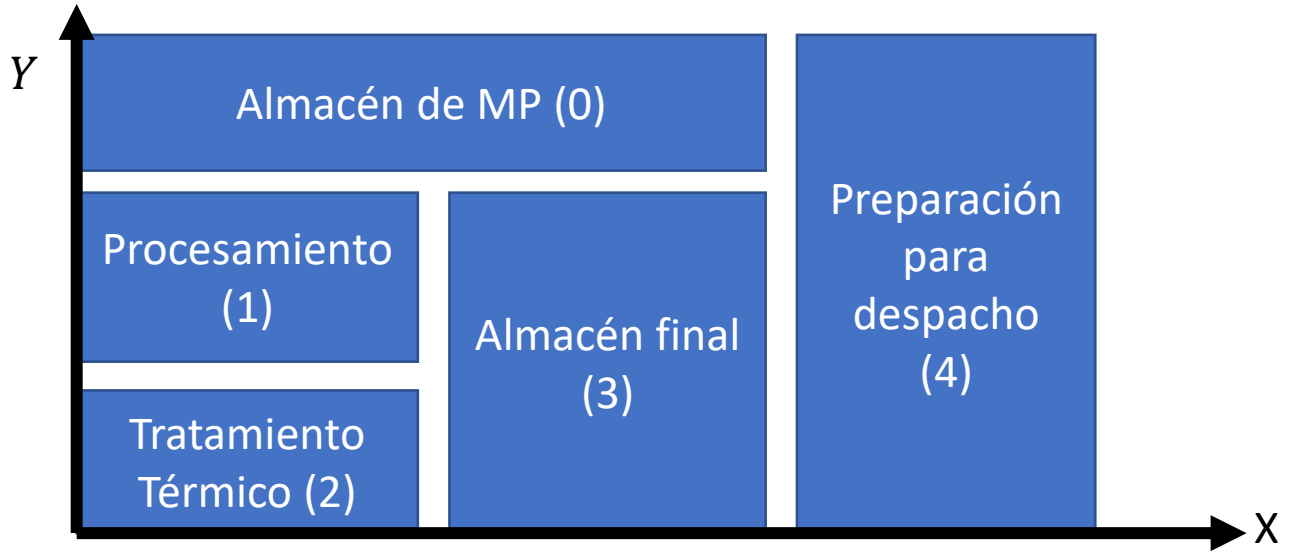
## Restricciones de posición relativa de áreas:

En el eje x:

$$x_0 + w_0 + \rho \leq x_4$$
$$x_1 + w_1 + \rho \leq x_3$$
$$x_2 + w_2 + \rho \leq x_3$$
$$x_3 + w_3 + \rho \leq x_4$$
$$x_4 + w_4 \leq W$$

En el eje y:

$$y_2 + h_2 + \rho \leq y_1$$
$$y_1 + h_1 + \rho \leq y_0$$
$$y_3 + h_3 + \rho \leq y_0$$
$$y_0 + h_0 \leq H$$
$$y_4 + h_4 \leq H$$



# Ejemplo: modelo de floorplanning convexo

## Restricciones de área:

### Área mínima:

$$Amin_i \leq w_i * h_i \qquad \forall i$$

convertimos a restricción convexa

$$\log(Amin_i) \leq \log(w_i) + \log(h_i) \qquad \forall i$$

### Límites de relación de aspecto:

$$h_i \leq \gamma_{max} * w_i \qquad \forall i$$
$$h_i \geq \gamma_{min} * w_i \qquad \forall i$$

# Ejemplo: modelo de floorplanning convexo

$$Min\ W + H$$

**s.t.**

$$\begin{aligned}
x_0 + w_0 + \rho &\le x_4 \\
x_1 + w_1 + \rho &\le x_3 \\
x_2 + w_2 + \rho &\le x_3 \\
x_3 + w_3 + \rho &\le x_4 \\
x_4 + w_4 &\le W \\
y_2 + h_2 + \rho &\le y_1 \\
y_1 + h_1 + \rho &\le y_0 \\
y_3 + h_3 + \rho &\le y_0 \\
y_0 + h_0 &\le H \\
y_4 + h_4 &\le H \\
\log(Amin_i) \le \log(w_i) + \log(h_i) &\qquad \forall i \\
h_i \le \gamma_{max} * w_i &\qquad \forall i \\
h_i \ge \gamma_{min} * w_i &\qquad \forall i
\end{aligned}$$

$$x_i \ge 0, \qquad y_i \ge 0, \qquad w_i \ge 0, \qquad h_i \ge 0 \qquad \forall i,$$
$$W \ge 0, \qquad H \ge 0$$

# Resolución en Python con librería cvxpy

CVXPY es una librería open-source de optimización convexa.

Permite modelizar problemas, resolverlos y hacer chequeos de convexidad en las ecuaciones.

Documentación: https://www.cvxpy.org/

Repositorio: https://github.com/cvxpy/cvxpy

# Resolución en Python con librería cvxpy

```python
import cvxpy as cp
import numpy as np

# Parametros
rho = 2.5 # Espacio entre cajas
gamma_max = 5 # Aspect ratio superior
gamma_min = 1/5 # Aspect ratio inferior
Amin = np.array([350., 300., 200., 300., 300.])

# Variables
W = cp.Variable(shape=(1)) # Ancho
H = cp.Variable(shape=(1)) # Largo
x = cp.Variable(shape=(5)) # Posición x del vértice inferior izquierdo
y = cp.Variable(shape=(5)) # Posición y del vértice inferior izquierdo
w = cp.Variable(shape=(5)) # Ancho de la caj
h = cp.Variable(shape=(5)) # Largo de la caja
```

# Resolución en Python con librería cvxpy

```python
# Indicamos la función objetivo:
objective = cp.Minimize(H + W) # Minimizar perímetro total

# Indicamos las restricciones:
## Restricciones de dependencia entre cajas en x:
constraints = [
    x[0] + w[0] + rho <= x[4],
    x[1] + w[1] + rho <= x[3],
    x[2] + w[2] + rho <= x[3],
    x[3] + w[3] + rho <= x[4],
    x[4] + w[4]       <= W
]

## Restricciones de dependencia entre cajas de y:
constraints += [
    y[2] + h[2] + rho <= y[1],
    y[1] + h[1] + rho <= y[0],
    y[3] + h[3] + rho <= y[0],
    y[0] + h[0] <= H,
    y[4] + h[4] <= H
]
```

$$Min\ W + H$$

$$\begin{aligned}
x_0 + w_0 + \rho &\leq x_4 \\
x_1 + w_1 + \rho &\leq x_3 \\
x_2 + w_2 + \rho &\leq x_3 \\
x_3 + w_3 + \rho &\leq x_4 \\
x_4 + w_4 &\leq W \\
y_2 + h_2 + \rho &\leq y_1 \\
y_1 + h_1 + \rho &\leq y_0 \\
y_3 + h_3 + \rho &\leq y_0 \\
y_0 + h_0 &\leq H \\
y_4 + h_4 &\leq H
\end{aligned}$$

# Resolución en Python con librería cvxpy

```python
# Restricción de área mínima para cada caja:
constraints.append(cp.log(Amin) ≤ cp.log(w) + cp.log(h))
# multiplicacion elemento por elemento

# Restricción de aspect ratio:
constraints.append(h ≤ cp.multiply(gamma_max, w))
constraints.append(h ≥ cp.multiply(gamma_min, w))

# Restricciones de positividad:
constraints += [
    W ≥ 0,
    H ≥ 0,
    x ≥ 0,
    y ≥ 0,
    w ≥ 0,
    h ≥ 0,
]
```

$$\log(Amin_i) \leq \log(w_i) + \log(h_i) \qquad \forall i$$

$$h_i \leq \gamma_{max} * w_i \qquad \forall i$$
$$h_i \geq \gamma_{min} * w_i \qquad \forall i$$

$$x_i \geq 0, \quad y_i \geq 0, \quad w_i \geq 0, \quad h_i \geq 0 \quad \forall i,$$
$$W \geq 0, \quad H \geq 0$$

# Resolución en Python con librería cvxpy

```python
# Armamos el problema:
prob = cp.Problem(objective, constraints)

# Resolvemos
result = prob.solve()

# Obtenemos los valores optimos de perímetro y área:
print('Perímetro', 2 * prob.value)
print('Área', H.value[0] * W.value[0])
```

```
>> Perímetro: 167.96
>> Área: 1752.59
```