

Programación no lineal

Clase 24

Investigación Operativa UTN FRBA

Curso: I4051

Docente: Martín Palazzo



UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

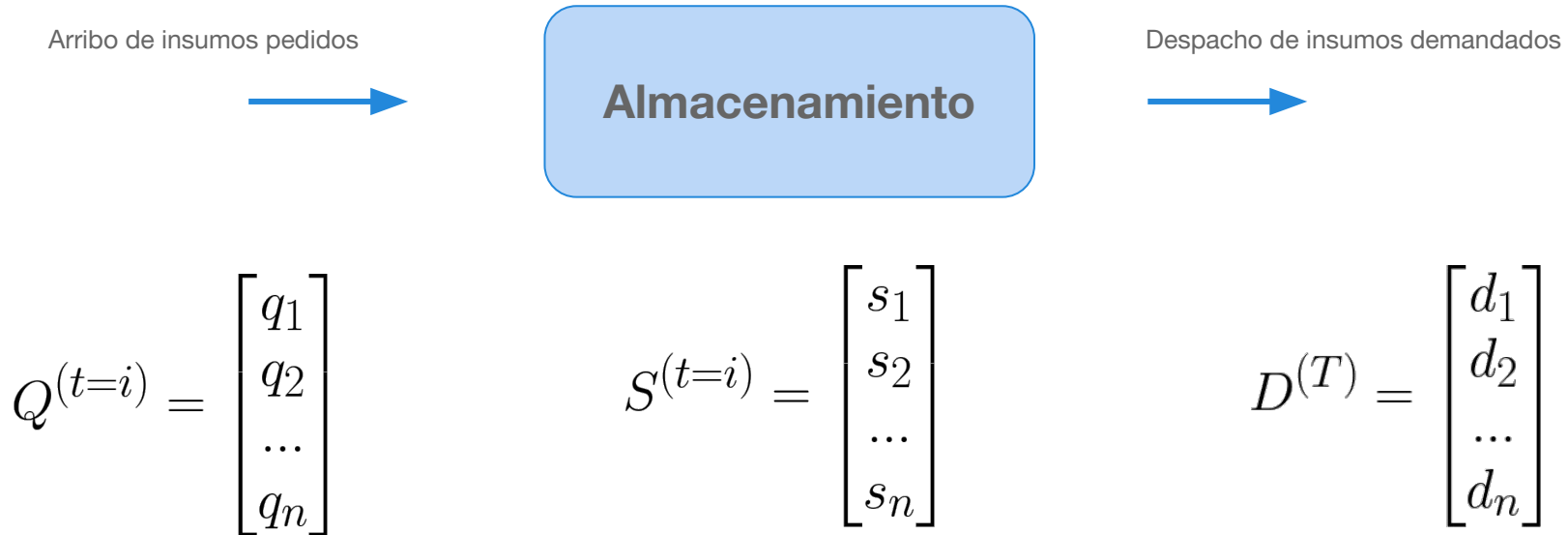
Agenda clase 24

- Repaso inventarios
- Optimización por gradiente descendiente
- Ejercicio Inventarios costo variable por cantidad
- Caso de aplicación: optimización del portfolio de inversiones

Modelos de Inventario con optimización convexa

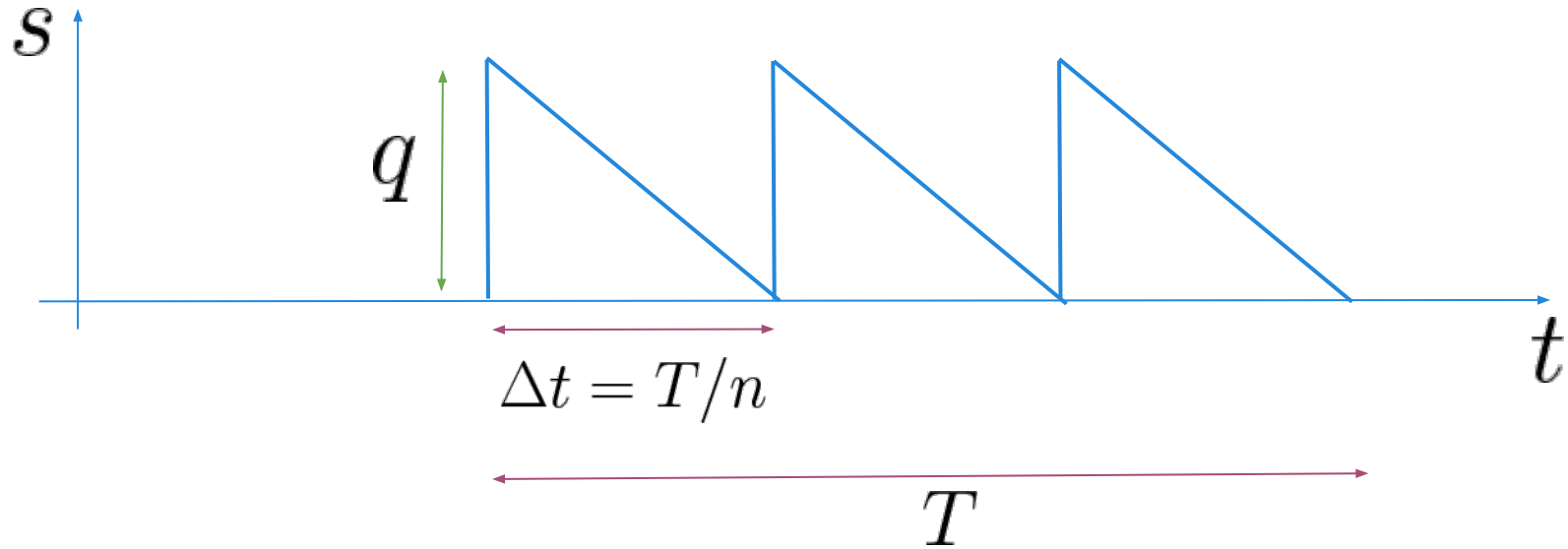
Caso de estudio

Sistemas de inventarios



Podemos modelizar el problema pensando que la cantidad a pedir regularmente en el tiempo de cada ítem es “q”, la cantidad de inventario de cada ítem en un tiempo t es “s” y la demanda de cada ítem en una ventana de tiempo es “d”. Si tenemos “n” ítems entonces nuestro problema es n-dimensional.

Modelo de inventario fundamental



En este modelo fundamental suponemos que hay un solo ítem $n=1$, la reposición de insumos es instantánea y que no existe stock de seguridad. El inventario se consume a una tasa D de unidades/tiempo (demanda).

Modelizar un problema de inventarios

$$f(q) = z \rightarrow \min$$

Simplificando el problema, si suponemos que la demanda en un periodo dado T es un dato conocido y constante entonces podemos modelizar al **costo** del inventario en función de la cantidad “ q ” de unidades a solicitar en cada pedido a realizar en el periodo T . Por esta razón, vamos a utilizar herramientas de optimización para encontrar el valor de “ q ” que minimiza la función objetivo costo “ f ”.

Costo del inventario

Costo Inventarios = Costo Compra/reposición + Costo Adquisición + Costo de almacenamiento + Costo faltante

El costo de inventario es una función con 3 componentes. El costo de compra/reposición se basa en el precio unitario de cada producto y la cantidad de producto a solicitar en cada orden. El costo de adquisición representa el costo fijo que requiere **cada** orden de compra (departamento de compras + recepción + control de calidad, etc). El costo de almacenamiento representa el costo de mantener una existencia de inventario disponible, incluye el interés sobre el capital inmovilizado, el costo de manipulación en el depósito y el mantenimiento del depósito.

Costo del inventario

$$\text{Costo Adq.} = b_i \cdot D$$

$$\text{Costo pedido} = k \cdot \frac{D_i}{q_i}$$

$$\text{Costo alm.} = \frac{1}{2} \cdot q_i \cdot i_i \cdot b_i$$

$$\frac{D_i}{q_i} = n$$

$$z_i = b_i \cdot D_i + k \cdot \frac{D_i}{q_i} + \frac{1}{2} \cdot q_i \cdot i_i \cdot b_i$$

n : cant. veces a pedir en T

D_i : Demanda del producto i en T

b_i : Precio de adquisicion por unidad de i

i_i : Tasa Anual de almacenamiento

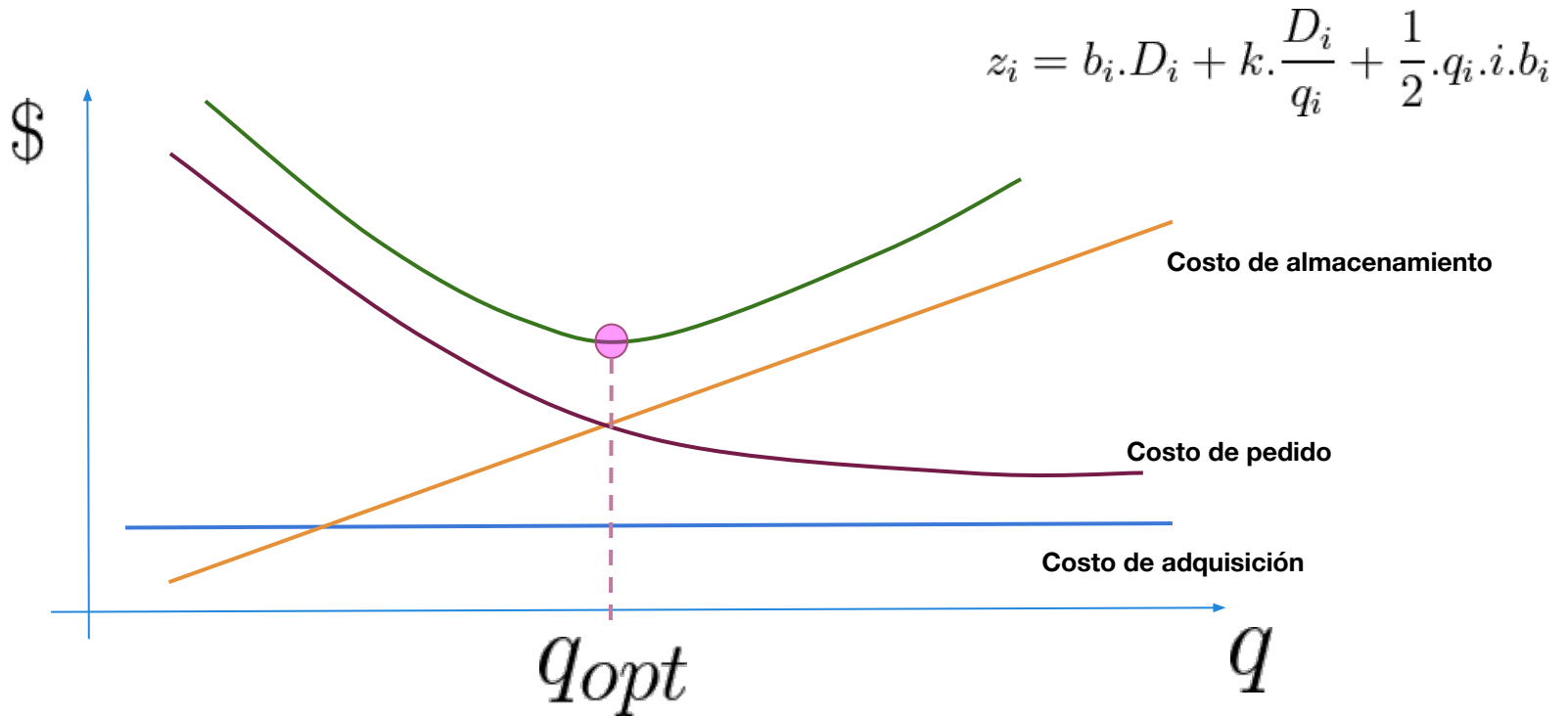
k : Costo de orden de compra

q_i : Cantidad a pedir en cada orden

T : Periodo de analisis

Función objetivo a minimizar -> costo total de almacenamiento en todo el periodo T

Costo Total Esperado



Minimo Costo Total Esperado

Para calcular el **mínimo** Costo Total Esperado (CTE) en el periodo T del sistema de inventarios tendremos las siguientes consideraciones:

- Suponemos que la función de costo es continua, diferenciable y convexa. Podemos entonces suponer que existe un mínimo global y encontrarlo en la coordenada donde la derivada es = 0.

$$\frac{dz}{dq} = \frac{d(b.D)}{dq} + \frac{d(k.\frac{D}{q})}{dq} + \frac{d(\frac{1}{2}.q.i.b)}{dq} = 0$$

- El costo de compra en el periodo T es una constante ya que no importa la cantidad de pedidos que se hagan ni cuanto se pida en cada uno, la cantidad a comprar en el periodo de análisis siempre será la misma y será la necesaria para satisfacer la demanda. Además su derivada en función de q (cantidad a pedir) es cero.
- Entonces el **mínimo** del CTE dependerá del costo de adquisición y el costo de almacenamiento.

Lote optimo

$$q_{\text{opt}} = \sqrt{\frac{2Dk}{b \cdot i}}$$

Entonces para un solo item con el modelo de inventarios fundamental más básico podemos calcular determinísticamente cual es el tamaño del lote de reposición óptimo que minimiza el costo total esperado.

Inventarios: multi-item sin restricción

- En el caso de tener “m” items sin restricción de volumen la solución que determine el Costo Total Esperado óptimo estará dada por un vector de “m” posiciones donde cada una determinará la cantidad óptima a solicitar de cada item.
- Para calcular el vector de items que minimice el costo lo que haremos es calcular de manera univariada/independiente el lote optimo de cada item y su respectivo costo.
- Una vez calculados los costos de cada item sumaremos cada término para obtener un costo total.

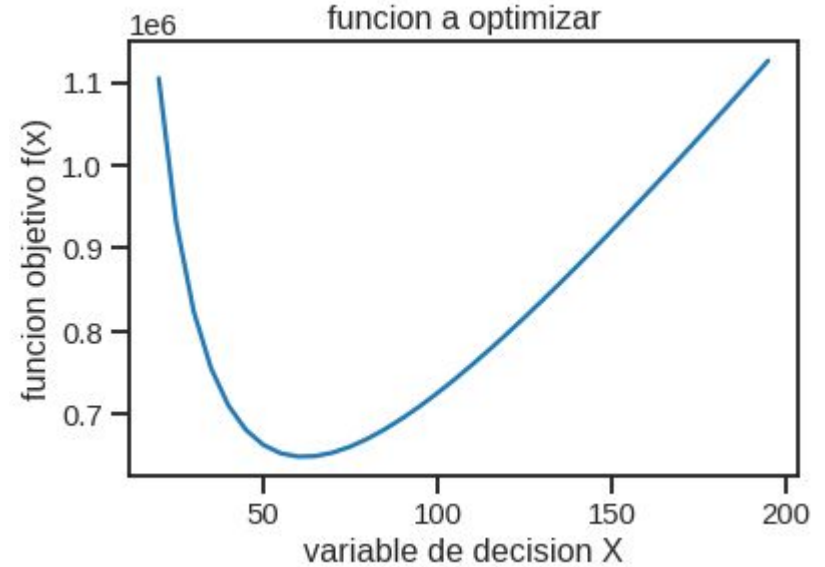
$$CTE_o = \sum_{i=1}^m b_i D_i + \frac{1}{2} \cdot q_i \cdot c_{1i} \cdot T + k_i \frac{D_i}{q_i}$$

$$Q_o = [q_{o1}, q_{o2}, \dots, q_{om}]$$

Función de costo de inventarios de 1 ítem



```
def costo_stock(p, x1, k, d, b):  
    return p*x1 + k*d*(1/x1) + 0.5*x1*b*p
```



Calculo de lote optimo analiticamente

```

# definimos la ecuacion de lote optimo obtenida analiticamente
# para minimizar el costo
def calcular_q_opt (k, d, b, i, ba = 0):
    return np.sqrt((2 * k * d)/(b * i + ba))

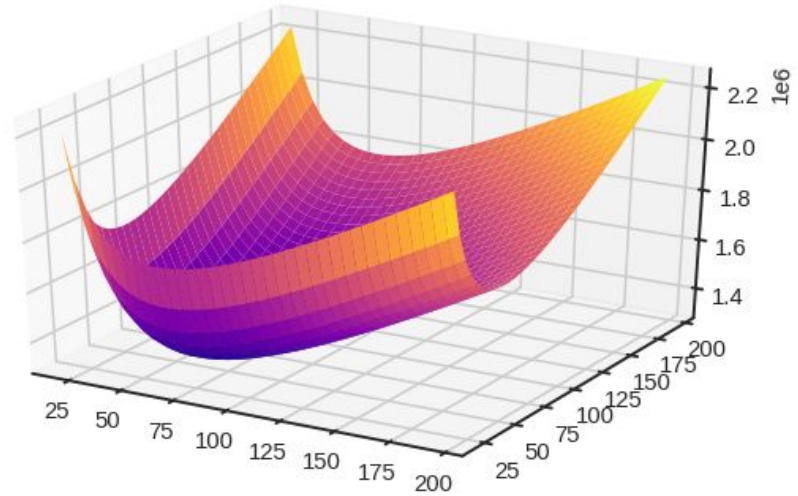
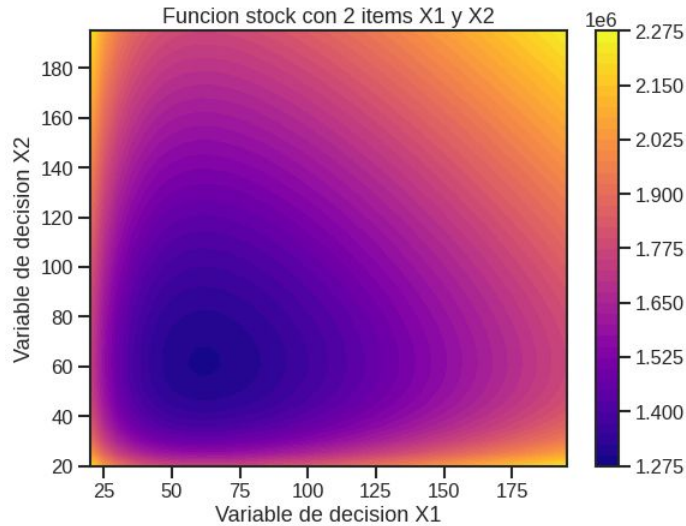
b = 5000 # costo por cada producto
k = 2000 # costo por cada orden de compra
d = 100 # demanda en el delta T
i = 0.01 # interes por almacenamiento

# usamos la funcion definida para calcular el lote optimo
q_optimo = calcular_q_opt (k, d, b, i, ba = 0)

print(q_optimo)
89.44271909999159
```

$$q_{\text{opt}} = \sqrt{\frac{2Dk}{b \cdot i}}$$

Función de costo de inventarios de 2 item

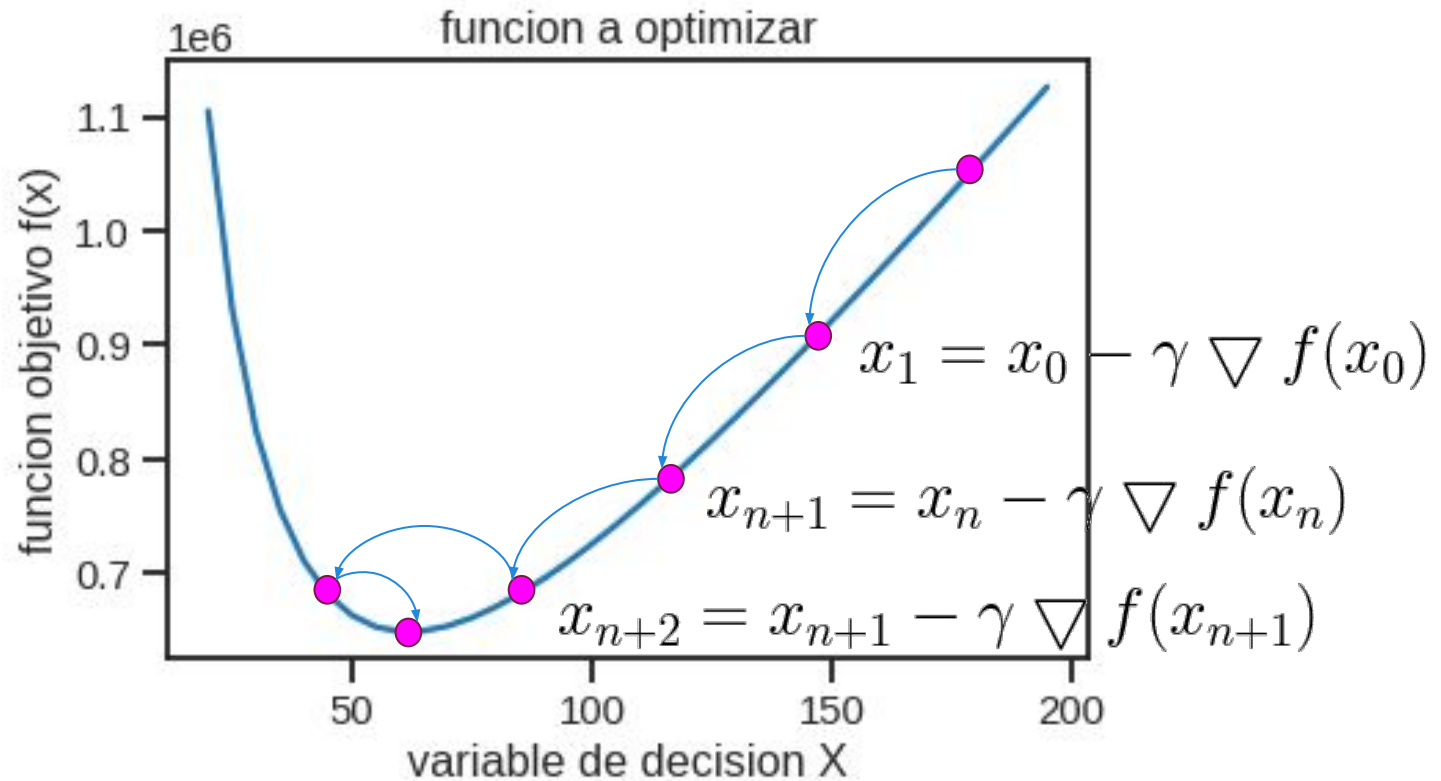


```
def costo_stock_2d(p1,p2, x1, x2, k1, k2, d1, d2, b1, b2):  
    costo_item1 = p*x1 + k*d*(1/x1) + 0.5*x1*b*p  
    costo_item2 = p*x2 + k*d*(1/x2) + 0.5*x2*b*p  
    return
```

Descenso por gradiente

Metodo para resolver: optimización no lineal

Gradiente descendiente



Optimización por gradiente descendiente

$$x \in \mathbb{R}^d \quad \min_x f(x) \quad \nabla f(x_0) \quad \gamma$$

Gradiente de $F(x)$	Paso de optimización
---------------------	----------------------

$$\begin{aligned} x_1 &= x_0 - \gamma \nabla f(x_0) \\ x_{n+1} &= x_n - \gamma \nabla f(x_n) \\ x_{n+2} &= x_{n+1} - \gamma \nabla f(x_{n+1}) \end{aligned} \quad f(x_n) > f(x_{n+1})$$

Supongamos que queremos optimizar la función no lineal, continua y diferenciable $f(x)$ definida en el dominio de X . Se partirá desde la solución inicial x_0 (llamado semilla) actualizando iterativamente la solución del problema en dirección al gradiente descendiente con un paso *gamma* de manera tal que la nueva solución presente un valor de la función objetivo menor.

Gradiente descendiente en Python

```
# definimos la funcion de gradiente descendiente
def gradient_descent(gradient, x0, learn_rate, n_iter=50, tolerance=1e-06):
    # semilla
    xi = x0

    # vector vacio para ir guardando el valor de la variable de decision en cada paso
    x_pasos = np.zeros(n_iter)
    # vector vacio para ir guardando el valor de la funcion objetivo en cada paso
    func_pasos = np.zeros(n_iter)

    # guardo la variable de decision en el paso 1
    x_pasos[0] = xi

    # guardo la funcion objetivo con el xi actual
    func_pasos[0] = costo_stock(xi)

    # for loop para iterar paso-a-paso
    for i in range(n_iter):

        # diff calcula cuanto optimizo la funcion objetivo
        diff = -learn_rate * gradient(xi)

        # si la diff es menor a un umbral interrumpo la optimizacion
        if np.all(np.abs(diff) <= tolerance):
            break

        # actualizo la variable de decision actual con la mejora del ultimo paso
        xi += diff

        # guardo la variable de decision en el paso 1
        x_pasos[i] = xi
        # guardo la funcion objetivo con el xi actual
        func_pasos[i] = costo_stock(xi)

    return xi, x_pasos, func_pasos
```

$$x_1 = x_0 - \gamma \nabla f(x_0)$$

$$x_{n+1} = x_n - \gamma \nabla f(x_n)$$

$$x_{n+2} = x_{n+1} - \gamma \nabla f(x_{n+1})$$

Solucion Lote Óptimo con Gradiente descendiente

$$z_i = b_i.D_i + k.\frac{D_i}{q_i} + \frac{1}{2}.q_i.i.b_i$$

Función objetivo a optimizar

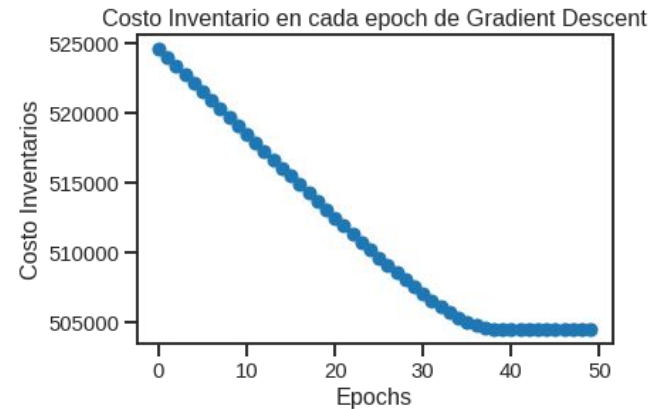
$$\frac{dz}{dq} = \frac{d(b.D)}{dq} + \frac{d(k.\frac{D}{q})}{dq} + \frac{d(\frac{1}{2}.q.i.b)}{dq}$$

$$\frac{dz}{dq} = -k \cdot d \cdot \frac{1}{q^2} + \frac{1}{2} \cdot i \cdot b$$

Gradiente de la función objetivo

Calculo del lote óptimo por gradiente descendiente

```
def grad_stocks_litem(x1):  
    return 0.5*i*b-k*d*x1*(-2)  
  
# defino la semilla  
x0 = 1000  
  
# ejecuto el gradiente descendiente en esta funcion  
x_opt, x_pasos, func_pasos = gradient_descent(grad_stocks_litem, x0, learn_rate = 1, n_iter=50,  
tolerance=1e-06)  
  
#  
print(x_opt)  
89.44720717143278
```



Optimizando con scipy

Usando `scipy.optimize()`

scipy.optimize()

Función objetivo
definida en python

$$f(x)$$

Variables de decisión
y conjunto factible

$$x$$

$$g(x) \leq 0$$

$$h(x) = 0$$

scipy.optimize()

Punto X optimo

$$x^*$$

$$\nabla f(x)$$

Gradiente de $f(x)$


- Determina la pendiente de la función

$$\nabla^2 f(x)$$

Hessiano de $f(x)$

- Determina la curvatura de la función

Calculo del lote optimo con gradiente descendiente



```
# bounds de las variables de decision, condiciones de no-negatividad
bounds = [(1, None)]

# semilla
#x0 = np.array([300])
x0 = 300
import scipy
# solucion con scipy
sol = scipy.optimize.minimize(costo_stock, x0, bounds = bounds)

# imprimimos la solucion
print(sol.x[0])
89.44526047
```


Ejercicio Portfolio de Inversión

Optimización Convexa

Programación cuadrática no lineal

Portfolio Management

Supongamos que tenemos un capital fijo X para invertir y 3 posibles inversiones



La primer decisión a tomar en este caso es que % del monto total X se invertirá en cada empresa. Comúnmente existe un compromiso entre retorno y riesgo donde un mayor retorno suele estar asociado a una inversión de mayor riesgo. En el caso que se desee lograr un retorno objetivo $r_{\text{(min)}} = 0.1$ entonces el problema queda determinado por lograr el mix de inversión de menor riesgo que satisfaga dicho retorno.

Si la inversion a cada empresa “i” la modelamos con X_i y sabemos que la suma de todas las inversiones X_i llega al 100% entonces

$$X = [x_1, x_2, x_3] \quad \sum x_i = 1 \quad x_i \geq 0$$

Portfolio Management

Tomando algunas ideas de la ciencia de Finanzas consideraremos al retorno de invertir en cada empresa como una variable aleatoria $R_i = [r_1, \dots, r_n]$, por ejemplo una gaussiana. Entonces utilizando datos históricos de los retornos de invertir en cada empresa podemos obtener los parámetros que caracterizan a cada R_i , es decir μ (media) y su co-varianza Σ .

- **Retorno esperado de cada inversión:** utilizando el histórico de inversión determinaremos con la media de los datos el valor esperado de inversión.

$$\bar{r} = [\bar{r}_1, \bar{r}_2, \bar{r}_3] \qquad \bar{r}_i = \frac{\sum_{i=1}^n r_i}{n}$$

- **Covarianza de retornos entre todas las inversiones:** utilizando el histórico de inversión determinamos la matriz de covarianza entre cada par de empresas a invertir. Es decir la dispersión de múltiples variables entre si mismas.

$$\Sigma_{ij} = \text{COV}[R_i, R_j] = E[(R_i - E[R_i])(R_j - E[R_j])] \quad \longrightarrow \quad \Sigma = \begin{pmatrix} \sigma_{x_1x_1} & \sigma_{x_1x_2} & \sigma_{x_1x_3} \\ \sigma_{x_2x_1} & \sigma_{x_2x_2} & \sigma_{x_2x_3} \\ \sigma_{x_3x_1} & \sigma_{x_3x_2} & \sigma_{x_3x_3} \end{pmatrix}$$
$$\Sigma = \text{COV}[\mathbf{R}, \mathbf{R}] = E[(\mathbf{R} - \mu_{\mathbf{R}})(\mathbf{R} - \mu_{\mathbf{R}})^T]$$

Modelización del problema

Vamos a definir nuestra función de pérdida L (Loss) como una medida del riesgo en base a la cantidad de inversión afectada por la variabilidad de retornos de la acción invertida.

$$L = x^T \Sigma x$$

Se supone que en una inversión el riesgo quiere ser minimizado, por eso en este problema es utilizado como función objetivo a minimizar.

$$\min_x x^T \Sigma x$$

Modelización del problema

Vamos a definir nuestra función de pérdida L (Loss) como una medida del riesgo en base a la cantidad de inversión afectada por la variabilidad de retornos de la acción invertida.

$$L = x^T \Sigma x$$

Se supone que en una inversión el riesgo quiere ser minimizado, por eso en este problema es utilizado como función objetivo a minimizar.

$$\min_x x^T \Sigma x$$

Además el problema plantea un retorno mínimo determinado por el quien realiza la inversión. De esta manera se genera una restricción de mayor o igual a R_{min} (retorno minimo a recuperar).

$$\begin{aligned} r^T x &\geq r_{min} \\ x_i &\geq 0 \\ \sum x_i &= 1 \end{aligned}$$

Modelización del problema

$$\min_x x^T \Sigma x$$

$$r^T x \geq r_{min}$$

$$x_i \geq 0$$

$$\sum x_i = 1$$

El problema de optimización queda determinado como un problema no lineal, convexo (siempre que la matriz Sigma sea positiva semi definida) y con restricciones lineales.

Construyendo el problema desde los datos

$$\Sigma = \begin{pmatrix} \sigma_{x_1x_1} & \sigma_{x_1x_2} & \sigma_{x_1x_3} \\ \sigma_{x_2x_1} & \sigma_{x_2x_2} & \sigma_{x_2x_3} \\ \sigma_{x_3x_1} & \sigma_{x_3x_2} & \sigma_{x_3x_3} \end{pmatrix}$$

T	Tesla	MELI	23M
Q1	6	7	10
Q2	3	8	1
Q3	4	9	7

```
# Matriz de datos historicos
hist = np.array([[6,7,10],
                 [3,8,1],
                 [4,9,7]])

# calculamos matriz covarianza retorno
r_cov_hist = np.cov(hist.T)

# calculamos vector media retorno
r_avg_hist = np.mean(hist, axis = 0)
```

```
# imprimimos matriz covarianza retorno
print(r_cov_hist)

array([[ 2.33333333, -1.          ,  6.5         ],
       [-1.          ,  1.          , -1.5        ],
       [ 6.5         , -1.5        , 21.         ]])

# imprimimos vector media retorno
print(r_avg_hist)

array([4.33, 8.   , 6.   ])
```

$$\bar{r} = [\bar{r}_1, \bar{r}_2, \bar{r}_3]$$

Supongamos que tenemos una tabla de retornos históricos de cada empresa de los últimos tres trimestres.

Modelamos el problema en Python

```
import scipy

# objective function
def objective(x):
    f = np.dot(np.dot(x, r_cov_hist), x)
    #f = -(30*x[0]**0.5 + 20*x[1]**0.5)
    return f

# funcion de restricciones mayor o igual
def c1(x):
    c1 = np.dot(r_avg_hist, x)
    return c1

# funcion de restricciones igualdad
def c2(x):
    c2 = np.sum(x)
    return c2

# Minimo retorno a obtener
r_min = 0.1

# formalizo restricciones en el formato de scipy
con1 = scipy.optimize.NonlinearConstraint(c1, lb = r_min, ub = np.inf)
con2 = scipy.optimize.NonlinearConstraint(c2, lb = 1, ub = 1)

# bounds de las variables de decision, condiciones de no-negatividad
bounds = [(0, None), (0, None), (0, None)]

# semilla
x0 = np.array([0.5, 0.4, 0.1])

# solucion con scipy
scipy.optimize.minimize(objective, x0, bounds = bounds, constraints = ([con1, con2]))
```

$$\min_x x^T \Sigma x$$

$$r^T x \geq r_{min}$$

$$x_i \geq 0$$

$$\sum x_i = 1$$

Solucion al problema de optimizacion

```
fun: 0.25000000000000006  
jac: array([0.50000002, 0.50000002, 3.00000028])  
message: 'Optimization terminated successfully.'  
nfev: 26  
nit: 5  
njev: 5  
status: 0  
success: True  
x: array([0.375, 0.625, 0.  ])
```

