Gestión de inventarios Restricción de espacio: caso en Python
Clase 26

Investigación Operativa UTN FRBA 2021
Curso: I4051(Palazzo)
Docente: Rodrigo Maranzana

Modelo restringido por espacio

Siendo:

S: espacio total.
s_i: espacio unitario utilizado por cada producto.
q_i: cantidad de cada producto.
CTE: Costo total esperado.

$$\min_{q_i} \quad Z = CTE(q_1, q_2, \ldots, q_m)$$

$$s.t.$$

$$q_1 s_1 + q_2 s_2 + \ldots + q_m s_m \leq S$$

Modelo relajado

Siendo:

λ: multiplicador de Lagrange.

L: Lagrangiano.

$$f = CTE(q_1, q_2, \ldots, q_m)$$

$$g = (q_1 s_1 + q_2 s_2 + \ldots + q_m s_m - S)$$

El modelo relajado será el siguiente:

$$L(\lambda) = min_{q_i} \quad Z_{relax} = f + \lambda g$$

$$L^*(\lambda^*) = max_{\lambda} min_{q_i} f + \lambda g$$

Cantidad óptima

$$q_i^* = \sqrt{\frac{2 \ * K_i * D_i}{T \ * \ c_{ui} + \mathbf{2} \ * \ \boldsymbol{\lambda} \ * \boldsymbol{s_i}}}$$

```
def calcular_qopt(K, D, T, c1, s_i, lmbd):
    return math.sqrt((2 * K * D) / (T * c1 + 2 * lmbd *s_i))
```

Restricción de espacio

$$g = (q_1 s_1 + q_2 s_2 + \ldots + q_m s_m - S)$$

$$g = \begin{bmatrix} q_1 & q_2 & \cdots & q_m \end{bmatrix} \times \begin{bmatrix} s_1 \\ s_2 \\ \cdots \\ s_m \end{bmatrix} - S$$ (Forma vectorial)

```python
def calcular_g(vect_s, vect_q, S):
    return vect_s @ vect_q - S
```

Costo total esperado del producto *i*

$$f_i = CTE(q_i) = Cadq + Calm(q_i) + Cpedido(q_i)$$

```
def calcular_f_i(b_i, d_i, q_i, c1_i, t, k_i):
    return b_i * d_i + 0.5 * q_i * c1_i * t + k_i * (d_i / q_i)
```

Costo total esperado (vectorizado)

$$f = CTE(q_1, q_2, \ldots, q_m)$$

$$f = \sum_i f_i$$

```python
def calcular_f(vect_b, vect_d, vect_q, vect_c1, t, vect_k):
    # Vectorizar la funcion:
    calcular_f_i_vectorizada =  np.vectorize(calcular_f_i)

    # Calcularmos f_i con la función vectorizada, mismos inputs que calcular_f_i pero
    # en vectores con todos los valores.
    vector_f_i = calcular_f_i_vectorizada(vect_b, vect_d, vect_q, vect_c1, t, vect_k)

    # Nos devuelve un vector con cada f_i que tenemos que sumar y retornar:
    return np.sum(vector_f_i)
```

Lagrangiano $L(\lambda)$

$$L(\lambda) = min_{q_i} \quad Z_{relax} = f + \lambda g$$

```
# Lagrangiano:
def calcular_L(f_q, g_q, lmbd):
    return f_q + lmbd * g_q
```

Datos

```
# Ejemplo:
S = 150

diasmes = 30
t = 1 # período de análisis
interes = 0.1 # anual

# Datos producto 1:
b_1 = 30 #costo por producto
alquiler_1 = 30 # diario
compra_1 = 100 # unidad
calidadrecepcion_1 = 200 # pedido
demanda_1 = 3000 # por año
k_1 = calidadrecepcion_1 + compra_1 # costo de orden
d_1 = demanda_1 # demanda
c1_1 = b_1 * interes + (alquiler_1 * diasmes * 12) # costo unitario
s_1 = 10

# Datos producto 2:
b_2 = 40 #costo por producto
alquiler_2 = 40 # diario
compra_2 = 150 # unidad
calidadrecepcion_2 = 250 # pedido
demanda_2 = 4300 # por año
k_2 = calidadrecepcion_2 + compra_2 # costo de orden
d_2 = demanda_2 # demanda
c1_2 = b_2 * interes + (alquiler_2 * diasmes * 12) # costo unitario
s_2 = 15
```

Construcción de vectores de datos

```
# Construcción de vectores de datos:
vect_s = np.array([s_1, s_2])
vect_b = np.array([b_1, b_2])
vect_d = np.array([d_1, d_2])
vect_c1 = np.array([c1_1, c1_2])
vect_k = np.array([k_1, k_2])
```

Búsqueda de λ^* con Grid Search

Pseudocódigo:

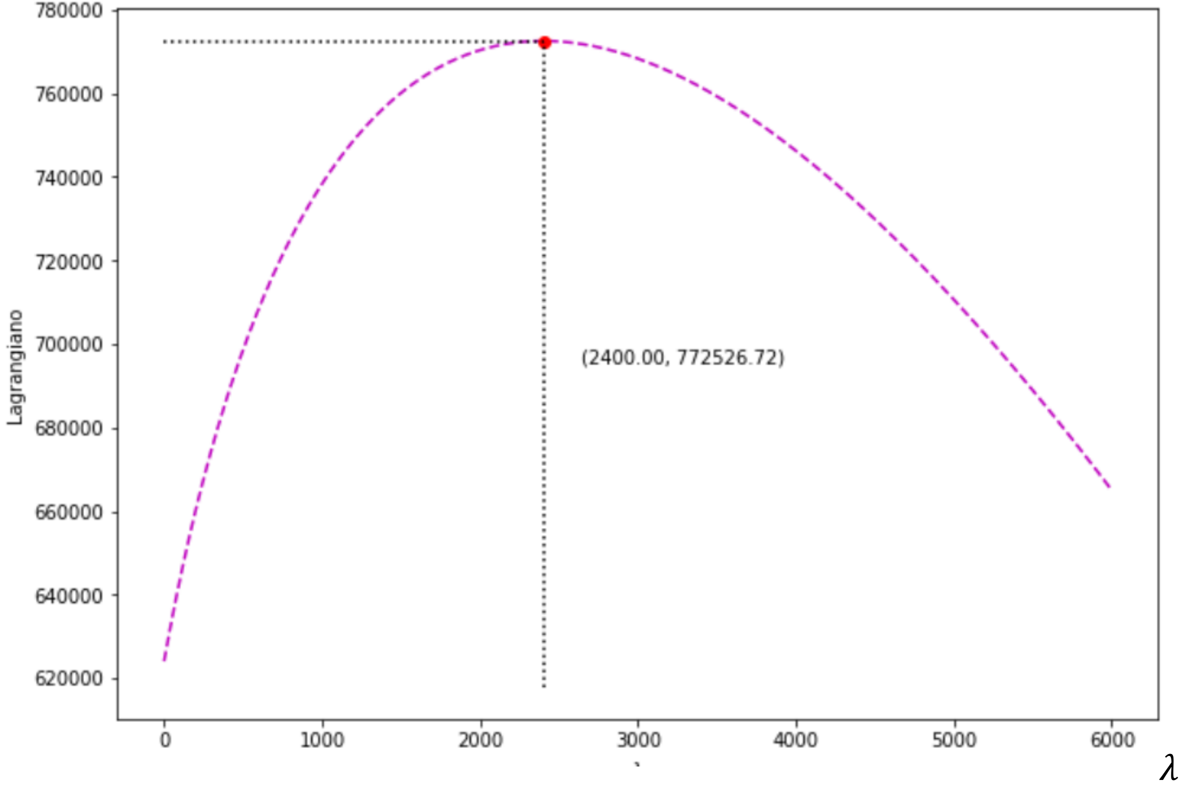
- Creamos un vector de lambdas $\lambda = [\lambda_1, \lambda_2, \dots \lambda_n]$

- Para cada λ_i:
    - Calculamos el óptimo q_i
    - Construimos el vector de óptimos $Q = [q_1, q_2, \dots, q_m]$
    - Calculamos g y f
    - Calculamos $L(\lambda_i)$ para el lambda actual.
    - Guardamos $L(\lambda_i)$ en un vector de soluciones $Lvector$

- Buscamos el máximo $L(\lambda_i)$ en $Lvector$

```
lmbds = np.array(range(0, 6000, 10))
L = np.zeros(lmbds.shape)

for i, lmbd_i in enumerate(lmbds):
    # Cálculo de cada cantidad óptima:
    q1_opt = calcular_qopt(k_1, d_1, t, c1_1, s_1, lmbd_i)
    q2_opt = calcular_qopt(k_2, d_2, t, c1_2, s_2, lmbd_i)

    # Construcción de arrays de q:
    vect_q_opt = np.array([q1_opt, q2_opt])

    # Cálculo de g y f:
    g = calcular_g(vect_q_opt, vect_s, S)
    f = calcular_f(vect_b, vect_d, vect_q_opt, vect_c1, t, vect_k)

    # Cálculo del Lagrangiano:
    L[i] = calcular_L(f, g, lmbd_i)

# Máximo:
max_index = np.argmax(L)
L_max = L[max_index]
lmbd_max = lmbds[max_index]
```

Plot $L(\lambda_i)$




```
RESULTADOS:
El lambda óptimo es: 2400.00
Las cantidades óptimas son: 5.53, 6.31
El CTE óptimo es: 772591.03
```

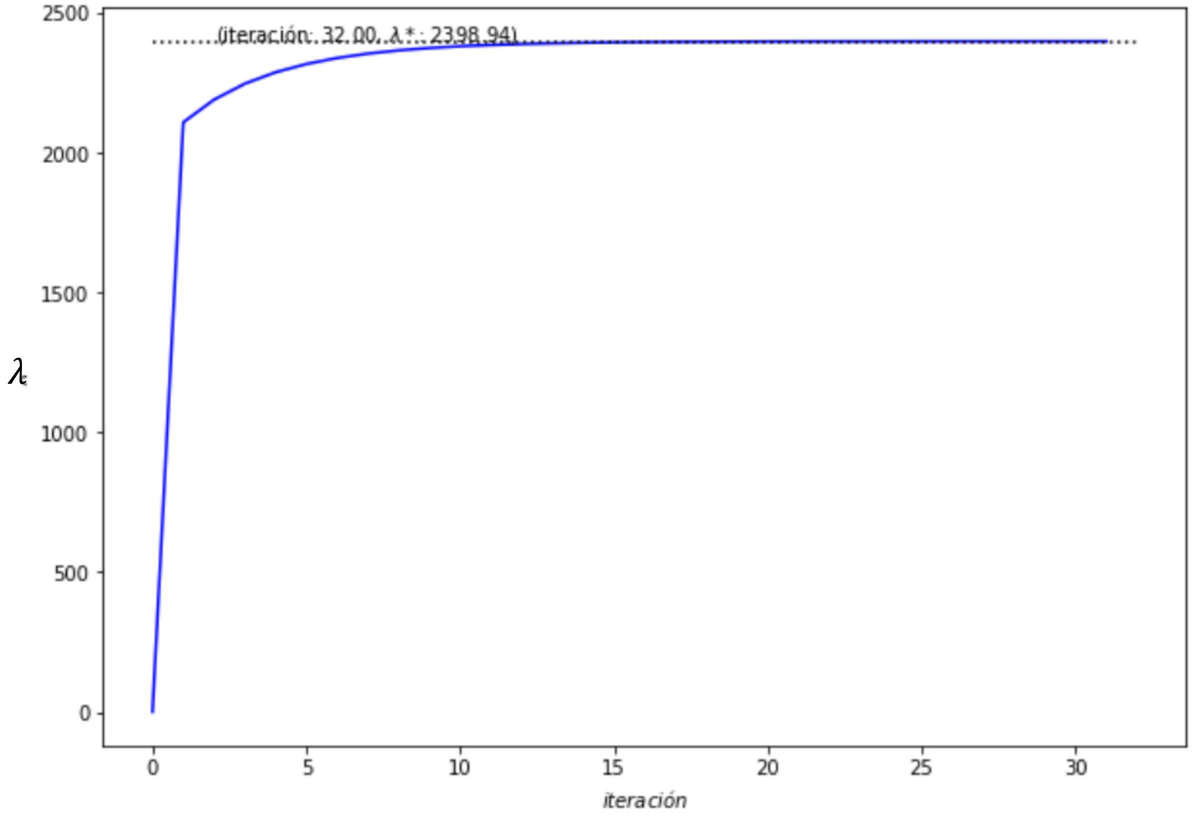
Búsqueda de λ^* con método del SubGradiente

Pseudocódigo:

-Inicializar λ_0
-Calcular $L(\lambda)$
-Calcular $\nabla L(\lambda)$
-Actualizar λ: $\lambda_{i+1} = \lambda_i + step * \nabla L(\lambda)$
-Calcular $\Delta\lambda = |\lambda_{i+1} - \lambda_i|$
-Revisar si $\Delta\lambda > tol$, continuar; sino parar.

```python
def gradiente_L(vect_s, vect_q, S):
    return vect_s @ vect_q - S

lmbd_0 = 0
step = 10 # paso fijo, sólo funciona en estos casos simples. Referirse a "métodos de paso adaptativo"
tol = 10e-3
i = 0
lmbd_i = lmbd_0
lmbd_array = []
L = []
diff = np.inf

while diff > tol:
    # Cálculo de cada cantidad óptima:
    q1_opt = calcular_qopt(k_1, d_1, t, c1_1, s_1, lmbd_i)
    q2_opt = calcular_qopt(k_2, d_2, t, c1_2, s_2, lmbd_i)

    # Construcción de arrays de q:
    vect_q_opt = np.array([q1_opt, q2_opt])

    # Cálculo de g y f:
    g = calcular_g(vect_q_opt, vect_s, S)
    f = calcular_f(vect_b, vect_d, vect_q_opt, vect_c1, t, vect_k)

    # Cálculo del Lagrangiano:
    L.append(calcular_L(f, g, lmbd_i))

    # Nuevo Lambda:
    lmbd_old = lmbd_i
    lmbd_array.append(lmbd_old)
    lmbd_i = max(lmbd_i + step * gradiente_L(vect_q_opt, vect_s, S), 0) # Lambda positivo

    # Chequeo de convergencia:
    diff = abs(lmbd_i - lmbd_old)

    i+=1
```

Búsqueda de λ^* con método del SubGradiente




```
RESULTADOS:
El lambda óptimo es: 2398.94
Las cantidades óptimas son: 5.53, 6.31
El CTE óptimo es: 772524.68
```