Proyecto final Optimización Avanzada

Eduardo Moreno, Leonardo Ceja, Cecilia Avilés, Carolina Acosta

Prefacio

★ En la siguiente presentación se explicará el paquete ***mex*** desarrollado por los autores de este proyecto, así como una breve introducción a la *Programación Lineal* y al *método símplex*.

Programación Lineal - Historia

- La Programación Lineal es un conjunto de técnicas que nace de una de las ramas de la matemática, con mayor auge a lo largo de los Siglos XX y XXI ,denominada como *Optimización.*
- Tiene su cénit a mediados del año 1947 al finalizar la Segunda Guerra Mundial cuando se da a conocer el método símplex desarrollado por George B. Dantzig.

Programación Lineal

Como su nombre lo indica, busca resolver problemas lineales en su función objetivo y restricciones, tal como se muestra a continuación:

$$\min_{x \in \mathbb{R}^n} c^T x$$

sujeto a

$$h(x) = Ax \leq b$$

$$x \geq 0$$

$$x = (x_1, x_2, \ldots, x_n)^T$$

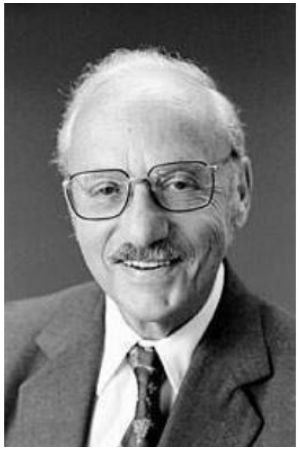
$$c = (c_1, c_2, \ldots, c_n)^T$$

$$b = (b_1, b_2, \ldots, b_m)^T$$

$$A = \begin{bmatrix} a11 & a12 & \ldots & a1n \\ a21 & a22 & \ldots & a2n \\ \vdots & \vdots & \ldots & \vdots \\ am1 & am2 & \ldots & amn \end{bmatrix}; A \in \mathbb{R}^{mxn}$$

Método Símplex

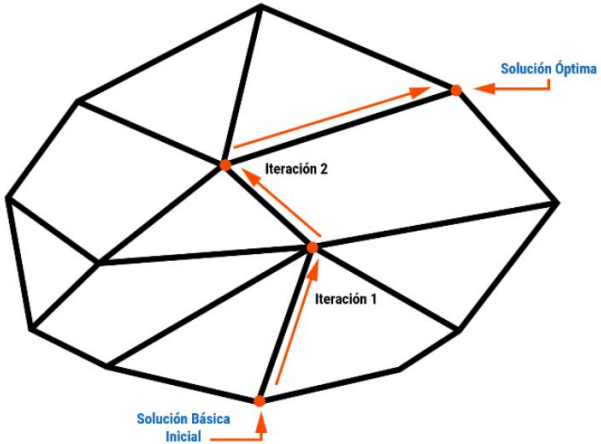
Creado por el matemático George B. Dantzig en el año 1947. Tiene como objetivo resolver problemas de programación lineal permitiendo mejorar las respuestas paso a paso, logrando así alcanzar la solución óptima.



Método Símplex - Intuición

Las siguientes definiciones y resultados nos ayudarán a mejorar nuestra percepción de cómo funciona el método símplex:

- El conjunto factible puede representarse por medio de un **poliedro convexo** (éste es un resultado aplicable a cualquier *LP*)
- Si un *LP* tiene una solución óptima y finita, entonces dicha solución se encuentra en uno de los **vértices** del poliedro convexo.



Método Símplex - Problema

El método símplex soluciona problemas en su forma estándar, la cual es la siguiente:

$$\min_{x \in \mathbb{R}^n} c^T x$$

$$\text{sujeto a}$$
$$Ax = b$$
$$x \geq 0 \qquad A \in \mathbb{R}^{mxn}, b \in \mathbb{R}^m.$$

Para transformar cualquier problema lineal a su forma estándar se añaden **variables sintéticas**.

Método Símplex - Condiciones

Para encontrar la solución óptima de un *LP*, el método símplex utiliza las siguientes 2 condiciones:

1. Condición de optimalidad
2. Condición de factibilidad

Método Símplex - Condición de optimalidad

¿La solución actual es óptima o se puede mejorar?

- *Problema de **maximización**:* verificar que los valores del vector de costos reducidos son **mayores o iguales a cero**; esto indica que se ha alcanzado la solución óptima.
- *Problema de **minimización**:* verificar que los valores del vector de costos reducidos son **menores o iguales a cero**; esto indica que se ha alcanzado la solución óptima.

Método Símplex - Condición de factibilidad

¿El problema tiene solución no acotada?

- Si el problema tiene solución acotada o finita, entonces al menos uno de los valores de la columna pivote es mayor que 0.

Por ejemplo, en la siguiente tabla en todos se cumple la condición de factibilidad.

Tabla 1	C_j	3	2	0	0	0	
C_b	Base	X_1	X_2	S_1	S_2	S_3	R
0	S_1	2	5	1	0	0	35
0	S_2	3	-2	0	1	0	18
0	S_3	2	4	0	0	1	26
	Z	-3	-2	0	0	0	0

Método Símplex - Pasos

$$\min_{x \in \mathbb{R}^n} c^T x \quad \text{sujeto a} \quad h(x) = Ax \leq b \quad x \geq 0$$

$$\min_{x \in \mathbb{R}^n} c^T x \quad \text{sujeto a} \quad Ax = b \quad x \geq 0$$

Solución inicial

S_B

Condición de optimalidad: variable de entrada S_B

Condición de factibilidad: variable de salida S_N

Actualización por medio de operaciones Gauss-Jordan

$$\begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} & 0 \\ 5 & -2 & 0 & 1 \end{bmatrix} \xrightarrow{R_2 \to R_2 - 5R_1} \begin{bmatrix} 1 & -\frac{1}{2} & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & -\frac{5}{2} & 1 \end{bmatrix} \xrightarrow{R_2 \to 2R_2}$$

mex - Descripción

mex es un paquete desarrollado en Python que tiene como objetivo resolver problemas de programación lineal (*LP*) utilizando el método símplex.

Las funciones básicas son las siguientes:

- ***create_matrix***: genera matriz de dimensiones necesarias para el *tableau*.
- ***constraint***: agrega las restricciones, ya sean del tipo “menor o igual”, “mayor o igual” o “igual”.
- ***obj***: agrega la función objetivo del problema.
- ***maxz, minz:*** resuelve el problema a optimizar, ya sea de maximización o minimización, respectivamente.

Implementación - Descripción dataset

Se realizará la implementación del paquete con un problema de pequeña escala, el cual tiene las siguientes dimensiones:

- Número de restricciones: **488**
- Número de variables: **615**

El dataset fue obtenido del repositorio de Netlib, el cual se encarga de desarrollo de software, papers y dataset. El dataset se encuentra en formato *MPS*, el cual es un formato específico que cumplen en su mayoría los problemas de programación lineal comerciales.

Implementación - Lectura de datos

```
import scipy.io as sio

# mex
from mex.simplex.minimizer_class import Minimizer
from mex.simplex.maximizer_class import Maximizer
```

```
mat = sio.loadmat('../data/AGG.mat')
```

```
A = mat['A']
b = mat['b']
c = mat['c']
lbounds = mat['lbounds']
ubounds = mat['ubounds']
A.shape
```

```
(488, 615)
```

Implementación - Resolución con mex

```
A = A.toarray()
```

```
minim = Minimizer(A, b, c)
```

```
minim.add_constraints(lbounds,ubounds)
```

```
tableau_obj = minim.matrix
```

```
start_time = timeit.default_timer()
minim.solve()
secs_mex = timeit.default_timer() - start_time
print("Todo el proceso tomó",secs_mex,"segundos")
```

```
Todo el proceso tomó 110.79066195200357 segundos
```

Valores aproximados

```
min_approx = minim.get_min()
coeff_approx = minim.get_coeff()
```

Implementación - Valores objetivo

Valores objetivos

```
tableau_obj = minim.matrix

# Dimensiones originales del problema
n_restr = tableau_obj.shape[0]-1
n_vars = A.shape[1]

# Variables sin restricciones de cotas superiores ni inferiores
c_min_obj = tableau_obj[-1,0:n_vars]
A_min_obj = tableau_obj[0:n_restr, 0:n_vars]
b_min_obj = tableau_obj[0:n_restr, -1]
```

```
start_time_scipy = timeit.default_timer()
min_obj = linprog(c_min_obj, A_ub=A_min_obj, b_ub=b_min_obj).fun
secs_scipy = timeit.default_timer() - start_time_scipy
print("Todo el proceso tomó",secs_scipy,"segundos" )
```

```
Todo el proceso tomó 92.96342534699943 segundos
```

Implementación - Comprobación resultados

Aproximación de resultados

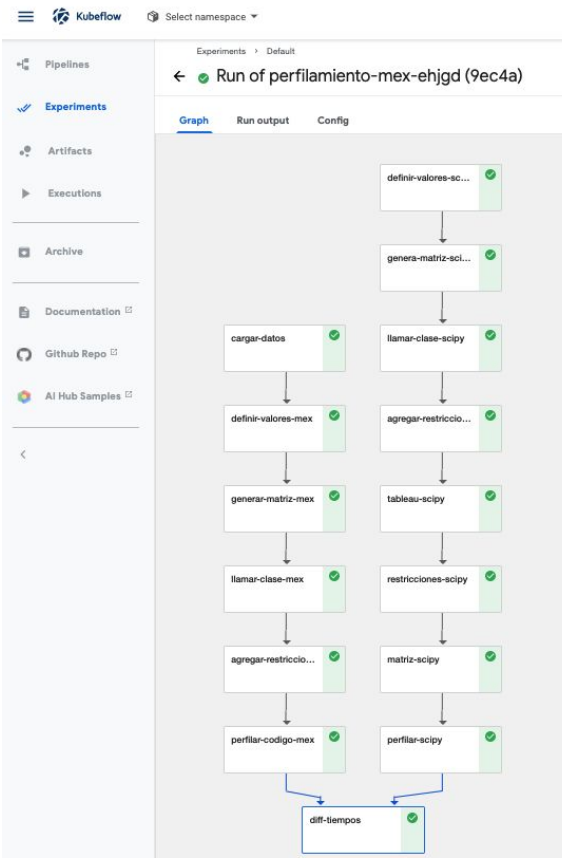
```
assert min_obj == approx(min_approx, rel=1e-1), "El valor aproximado es incorrecto"

print("El valor objetivo obtenido con scipy es: {0:.2g}".format(min_obj))
print("El valor aproximado obtenido con mex es: {0:.2g}".format(min_approx))
```

```
El valor objetivo obtenido con scipy es: -3.6e+35
El valor aproximado obtenido con mex es: -3.6e+35
```

Integración con Kale y Minikube

- Ejecutamos pipelines utilizando nuestro paquete, levantando una instancia con Kale/Minikube, y que además contara con una imagen de Docker con nuestro paquete instalado.



Compilación a C - Ejecución

```
#mex_c
from mex.mex_c.minimizer_class_c import Minimizer_c
from mex.mex_c.maximizer_class_c import Maximizer_c
```

```
minim_c = Minimizer_c(A,b,c)
```

```
minim_c.add_constraints(lbounds,ubounds)
```

```
minim_c.matrix.shape
```

```
(1719, 2335)
```

```
start_time_c = timeit.default_timer()
minim_c.solve()
secs_mex_c = timeit.default_timer() - start_time_c
print("Todo el proceso tomó",secs_mex_c,"segundos")
```

```
Todo el proceso tomó 81.64939518999745 segundos
```

Compilación a C - Resultados

Valores aproximados

```
min_approx_c = minim_c.get_min()
coeff_approx_c = minim_c.get_coeff()
```

Comprobación

```
assert min_obj == approx(min_approx_c, rel=1e-1), "El valor aproximado es incorrecto"

print("El valor objetivo obtenido con scipy es: {0:.2g}".format(min_obj))
print("El valor aproximado obtenido con mex es: {0:.2g}".format(min_approx_c))
```

```
El valor objetivo obtenido con scipy es: -3.6e+35
El valor aproximado obtenido con mex es: -3.6e+35
```

Reducción en tiempo

```
print("La resolución con el paquete mex base tomó {:.2f}".format(secs_mex),"segundos.")
print("La resolución con el paquete mex con compilación a C tomó {:.2f}".format(secs_mex_c),"segundos.")
print("La resolución con scipy tomó {:.2f}".format(secs_scipy),"segundos.")
```

```
La resolución con el paquete mex base tomó 110.79 segundos.
La resolución con el paquete mex con compilación a C tomó 81.65 segundos.
La resolución con scipy tomó 92.96 segundos.
```

```
secs_diff = secs_scipy - secs_mex_c
print("¡El paquete mex compilado a C necesitó {:.2f}".format(secs_diff),"segundos menos que el paquete s
```

```
¡El paquete mex compilado a C necesitó 11.31 segundos menos que el paquete scipy para la ejecución!
```

Conclusiones

- La estructura del *tableau* es muy utilizada para resolver problemas lineales de pequeña escala (decenas, cientos y miles de variables), sin embargo, no es eficiente para problemas de mediana y/o grande escala.
- Como alternativas para la resolución de problemas de mayor escala, se tienen por ejemplo los métodos por puntos interiores, como el método primal-dual de barrera logarítmica.
- La principal diferencia entre los métodos de puntos interiores y el método símplex es que, como su nombre lo indica, comienzan en un punto interior de la región factible, y en cada iteración se van aproximando a los puntos óptimos en el límite.
- Cada iteración en el método de puntos interiores es costosa de calcular (comparando con el método símplex), sin embargo, para problemas de gran escala no se requieren muchas más iteraciones respecto a un problema de pequeña escala.

Preguntas