Optimización Semidefinida - Segundo Cuatrimestre 2021

Primeros pasos en Mosek

Referencia principal MOSEK Optimizer API for Python 9.3.6
https://docs.mosek.com/latest/pythonapi/index.html
Ver Sección 6.6 - Semidefinite optimization.

1. Instalación

Seguir los siguientes pasos:

1. Obtener el archivo de licencia mosek.lic en la página

   `https://www.mosek.com/products/academic-licenses/`

   solicitando la licencia personal académica. Se requiere cuenta de correo electrónica académica.

2. En Windows, crear una carpeta con nombre mosek en la carpeta principal del usuario. Por ejemplo, si el usuario de Windows es jperez, crear la carpeta mosek en `C:\Users\jperez`

3. Guardar el archivo de licencia en la carpeta creada.

4. Instalar mosek utilizando Anaconda Power Shell Prompt mediante el comando

   ```
   conda install -c mosek mosek
   ```

   (ver `https://docs.mosek.com/latest/pythonapi/install-interface.html` para otras opciones)

5. Probar la instalación en Jupyter Notebook, ejecutando por ejemplo el código disponible en la página

   `https://docs.mosek.com/latest/pythonapi/tutorial-sdo-shared.html`

2. Ejemplo 1

Vamos a resolver el siguiente ejemplo:

$$\begin{aligned} \text{minimizar: } & x_{11} \\ \text{sujeto a: } & x_{00} = 1 \\ & x_{01} = 3 \\ & \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix} \succeq 0. \end{aligned}$$

(numeramos filas y columnas a partir de 0 para compatbilidad con Python).
Comparando con la forma general del problema SDP en Mosek descripto en `https://docs.mosek.com/latest/pythonapi/tutorial-sdo-shared.html`, tenemos:

1. $p = 1$, hay una sola matriz $\boldsymbol{X}$ sobre la que tenemos la condición $\boldsymbol{X} \succeq 0$.

2. Por lo tanto, el único valor de j es $j = 0$ y $\bar{\boldsymbol{X}}_0 = \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix}$.

3. $r_0 = 2$, es la dimensión de la matrix $\boldsymbol{X}_0$.

4. La función a minimizar es $0x_{00} + 0x_{01} + 1x_{11}$, que con la notación $\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \sum\sum a_{ij} b_{ij}$, corresponde a
$$\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix},$$
por lo tanto, $\bar{\boldsymbol{C}}_0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$.

5. No hay variables adicionales x_j en la función a minimizar, por lo tanto, $n = 0$.

6. Tenemos dos restricciones sobre los coeficientes de $\boldsymbol{X}$, por lo tanto $m = 2$ y las dos restricciones van indexadas con la variable $i = 0, 1$.

7. La restricción $x_{00} = 1$ la escribimos como
$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix} = 1$$
y la restricción $x_{01} = 3$ la escribimos como
$$\begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix} \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix} = 3$$
(observar que todas las matrices deben ser simétrica). Obtenemos
$$\bar{\boldsymbol{A}}_{00} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{y } \bar{\boldsymbol{A}}_{10} = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}.$$

Ahora estamos en condiciones de escribir el programa en Mosek, reemplazando el código correspondiente en el primer ejemplo de `https://docs.mosek.com/latest/pythonapi/tutorial-sdo-shared.html`.

1. Construimos la matriz $\bar{C}_0$ definiéndola en forma esparsa con tres listas barci, barcj y barcval de la misma longitud k. Para cada índice a, $0 \leq a \leq k$, se asigna una coordenada de C por la fórmula
$$c_{(\texttt{barci[a]},\texttt{barcj[a]})} = \texttt{barcval[a]}.$$
En nuestro ejemplo hay un solo coeficiente no nulo en $\bar{C}_0$: $c_{11} = 1$ y por lo tanto definimos
```
barci = [1]
barcj = [1]
barcval = [1.0]
```

2. De la misma forma definimos las matrices $\bar{A}_{i0}$. En este caso como son varias matrices, las definimos mediante listas de listas. Cada lista indica los coeficientes de una matriz. Obtenemos
```
barai = [[0], [1]]
baraj = [[0], [0]]
baraval = [[1.0], [0.5]]
```
Notar que solo definimos las matrices para las coordenadas $i \geq j$, y las otras coordenadas quedan definidas por simetría (si definimos alguna coordenada $j > i$ nos tira error).

3. Ahora definimos las restricciones que corresponden a cada matriz $\boldsymbol{A}_i$. En nuestro caso tenemos dos restricciones de igualdad, por lo tanto definimos

```
bkc = [mosek.boundkey.fx, mosek.boundkey.fx]
```

En esta definición podemos modificar fx por otras restricciones, ver la tabla `https://docs.mosek.com/latest/javaapi/tutorial-lo-shared.html#doc-optimizer-tab-boundkeys2`.

4. Y fijamos las restricciones

```
blc = [1.0, 3.0]
buc = [1.0, 3.0]
```

que nos indica

$$\texttt{blc[i]} \leq \langle \boldsymbol{A}_{i0}, \boldsymbol{X}_0 \rangle \leq \texttt{buc[i]}, i = 0, 1.$$

5. Definimos los tamaños:

```
numcon = len(bkc)
BARVARDIM = [2]
```

donde *numcon* es la cantidad m de restricciones y $BARVARDIM$ es una lista con los tamaños de la matrice $\boldsymbol{X}_j$, que en nuestro caso es una sola matriz de tamaño 2.

6. Finalmente, cargamos todas las matrices y restricciones en las variables apropiadas

```
task.appendcons(numcon)

# Append one symmetric variables of dimension 3 (3x3 matrix)
task.appendbarvars(BARVARDIM)

symc = task.appendsparsesymmat(BARVARDIM[0],
                               barci,
                               barcj,
                               barcval)

syma0 = task.appendsparsesymmat(BARVARDIM[0],
                                barai[0],
                                baraj[0],
                                baraval[0])

syma1 = task.appendsparsesymmat(BARVARDIM[0],
                                barai[1],
                                baraj[1],
                                baraval[1])

task.putbarcj(0, [symc], [1.0])
task.putbaraij(0, 0, [syma0], [1.0])
task.putbaraij(1, 0, [syma1], [1.0])

for i in range(numcon):
  # Set the bounds on constraints.
  # blc[i] <= constraint_i <= buc[i]
  task.putconbound(i, bkc[i], blc[i], buc[i])
```

En nuestro ejemplo no hay variables x_j por lo tanto fijamos `numvar = 0` y todo lo que tiene que ver con `numvar`, `aval`, `asub` lo eliminamos.

7. Y especificamos que queremos minimizar la función objetivo.

```
# Input the objective sense (minimize/maximize)
task.putobjsense(mosek.objsense.minimize)
```

3. Ejemplo 2

Calcular el menor autovalor de

$$\boldsymbol{A} = \begin{pmatrix} 6 & 2 & 2 \\ 2 & 4 & 2 \\ 2 & 2 & 2 \end{pmatrix}$$

resolviendo el problema SDP:

$$\begin{aligned} &\text{maximizar: } \alpha \\ &\text{sujeto a:} \quad \boldsymbol{A} - \alpha \boldsymbol{I} \succeq 0. \end{aligned}$$

En este caso tenemos

$$\boldsymbol{X} = \begin{pmatrix} x_{00} & x_{01} & x_{02} \\ x_{01} & x_{11} & x_{12} \\ x_{02} & x_{12} & x_{22} \end{pmatrix} = \begin{pmatrix} 6 & 2 & 2 \\ 2 & 4 & 2 \\ 2 & 2 & 2 \end{pmatrix} - \begin{pmatrix} \alpha & 0 & 0 \\ 0 & \alpha & 0 \\ 0 & 0 & \alpha \end{pmatrix}$$

Por lo tanto en el problema modelo de `https://docs.mosek.com/latest/pythonapi/tutorial-sdo-shared.html`, tomamos $n = 1$ y $x_0 = \alpha$, siendo x_0 la función a maximizar.