Optimización Semidefinida - Segundo Cuatrimestre 2021

Problema primal SDP 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. Introducción

Mosek puede resolver problemas de optimización semidefinida de la forma:

$$\begin{aligned}
\text{minimizar: } & \sum_{j=0}^{n-1} c_j x_j + \sum_{j=0}^{p-1} \bar{\boldsymbol{C}}_j \bullet \bar{\boldsymbol{X}}_j \\
\text{sujeto a: } & l_i^c \leq \sum_{j=0}^{n-1} a_{ij} x_j + \sum_{j=0}^{p-1} \bar{\boldsymbol{A}}_{ij} \bullet \bar{\boldsymbol{X}}_j \leq u_i^c, \quad i = 0, \ldots, m-1, \\
& l_i^x \leq x_j \leq u_i^x, \quad j = 0, \ldots, n-1, \\
& \boldsymbol{x} \in \mathbb{R}^n, \bar{\boldsymbol{X}}_j \succeq 0, j = 0, \ldots, p-1
\end{aligned}$$

Vamos a implementar un problema primal en Mosek, es decir un problema de la forma:

$$\begin{aligned}
\text{minimizar: } \quad & \boldsymbol{C} \bullet \boldsymbol{X} \\
\text{sujeto a: } \quad & \boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad i = 0, \ldots, m-1, \\
& \boldsymbol{X} \succeq 0,
\end{aligned} \tag{SDP-P}$$

para una matriz $\boldsymbol{X} \in \mathbb{R}^{d \times d}$.
Observamos que:

- p es la cantidad de matrices semidefinidas positivas $\boldsymbol{X}$. Por lo tanto, tenemos $p = 1$ y una única matrix $\bar{\boldsymbol{X}}_0$.
- El vector $\boldsymbol{x}$ es un vector de variables auxiliares que no aparecen en el problema primal, por lo tanto $n = 0$.

Por lo tanto, el problema primal planteado en el formato de Mosek es

$$\begin{aligned}
\text{minimizar: } & \bar{\boldsymbol{C}}_0 \bullet \bar{\boldsymbol{X}}_0 \\
\text{sujeto a: } \quad & l_i^c \leq \bar{\boldsymbol{A}}_{i0} \bullet \bar{\boldsymbol{X}}_0 \leq u_i^c, \quad i = 0, \ldots, m-1, \\
& \bar{\boldsymbol{X}}_0 \succeq 0
\end{aligned}$$

donde $l_i^c = u_i^c = b_i$ para todo $i = 0, \ldots, m-1$.

2. Implementación

Debemos ingresar en Mosek:

1. la dimensión d de la matrix $\bar{\boldsymbol{X}}_0$,
2. la matrix $\bar{\boldsymbol{C}}_0$,
3. las matrices $\bar{\boldsymbol{A}}_{i0}, i = 0, \ldots, m-1$,
4. las cotas $l_i^c = u_i^c = b_i$, $i = 0, \ldots, m-1$
5. la cantidad de variables auxiliares $n = 0$
6. por último si buscamos el máximo o el mínimo de la función objetivo.

2.1. Dimension d de la matrix $\bar{\boldsymbol{X}}_0$

Definimos una variable `BARVARDIM` como una lista de los tamaños de las matrices $\bar{\boldsymbol{X}}_i$. Como en nuestro caso, hay una sola matriz, ingresamos

```
BARVARDIM = [d]
```

reemplazando d por el valor correspondiente
Agregamos al problema las variables de optimización correspondientes a las coordenadas de las matrices con el comando `appendbarvars`:

```
task.appendbarvars(BARVARDIM)
```

2.2. Matriz $\bar{\boldsymbol{C}}_0$ de coeficientes

Las matrices las ingresamos en forma esparsa, ingresando las coordenadas de la parte triangular inferior de la matriz (es decir, las coordenadas c_{ij} con $i \geq j$). Si hay k coordenadas no-nulas, definimos tres listas de longitud k:

- `barci`, contiene las coordenadas i de las casillas no nulas de $\bar{\boldsymbol{C}}_0$.
- `barcj`, contiene las coordenadas j de las casillas no nulas de $\bar{\boldsymbol{C}}_0$.
- `barcval`, contiene los valores de las coordenadas $\bar{c}_{ij}$ para los índices definidos en `barci` y `barcj`.

Para cada índice a, $0 \leq a \leq k$, se asigna una coordenada de $\bar{\boldsymbol{C}}_0$ por la fórmula

$$c_{(\texttt{barci[a]},\texttt{barcj[a]})} = \texttt{barcval[a]}.$$

Ejemplo. Para la matriz $\bar{\boldsymbol{C}}_0 = \begin{pmatrix} 2 & 1 \\ 1 & 0 \end{pmatrix}$ ingresamos

```
barci = [0, 1]
barcj = [0, 0]
barcval = [2.0, 1.0]
```

Cargamos esta matriz en el problema mediante los comandos siguientes (no es necesaria ninguna modificación):

```
symc = task.appendsparsesymmat(BARVARDIM[0],
                               barci,
                               barcj,
                               barcval)
task.putbarcj(0, [symc], [1.0])
```

2.3. Matrices $\bar{\boldsymbol{A}}_i$ de restricciones

Ingresamos las matrices $\bar{\boldsymbol{A}}_i$ también en forma esparsa, ingresando las coordenadas de la parte triangular inferior de la matriz. Si hay m matrices $\bar{\boldsymbol{A}}_i$, definimos tres listas de longitud m, donde el elemento i-ésimo de la lista es una lista de valores correspondiente a la definición esparsa de la matriz $\bar{\boldsymbol{A}}_i$:

- `barai`, contiene las coordenadas i de las casillas no nulas de las matrices $\bar{\boldsymbol{A}}_i$, $0 \leq i \leq m-1$.
- `baraj`, contiene las coordenadas j de las casillas no nulas de las matrices $\bar{\boldsymbol{A}}_i$, $0 \leq i \leq m-1$.
- `barcval`, contiene los valores de las coordenadas a_{ij} de las matrices $\bar{\boldsymbol{A}}_i$, $0 \leq i \leq m-1$, para los índices definidos en `barai` y `baraj`.

Ejemplo. Para la matriz $\bar{\boldsymbol{A}}_0 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ingresamos

```
barai = [[0, 1]]
baraj = [[0, 1]]
baraval = [[1.0, 1.0]]
```

Cargamos las matrices $\bar{\boldsymbol{A}}_i$, las cargamos mediante los comandos

```
numcon = len(barai)
task.appendcons(numcon)

syma = []
for i in range(len(barai)):
    syma.append(task.appendsparsesymmat(BARVARDIM[0],
                                        barai[i],
                                        baraj[i],
                                        baraval[i]))
    task.putbaraij(i, 0, [syma[i]], [1.0])
```

2.4. Cotas $l_i^c = u_i^c = b_i$, $i = 0, \ldots, m-1$.

Para definir las restricciones $\boldsymbol{A}_i \bullet \boldsymbol{X} = b_i, \quad i = 0, \ldots, m-1$, definimos tres listas de longitud m:

- `bkc`: lista de constantes, la i-ésima constante indica el tipo de cota correspondiente a la i-ésima restricción. Para restricciones de igualdad, utilizamos la constante `mosek.boundkey.fx`.
- `blc`: lista de cotas inferiores, el i-ésimo valor de la lista indica la cota inferior de la i-ésima restricción. Para restricciones de igualdad, ingresamos el valor b_i.
- `buc`: lista de cotas superiores, el i-ésimo valor de la lista indica la cota superior de la i-ésima restricción. Para restricciones de igualdad, ingresamos el valor b_i.

Ejemplo. Para la restricción $\boldsymbol{A}_0 \bullet \boldsymbol{X} = 1$ ingresamos

```
# Bound keys for constraints
bkc = [mosek.boundkey.fx]

# Bound values for constraints
blc = [1.0]
buc = [1.0]
```

Y cargamos todas las variables en el problema

```
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])
```

2.5. No hay variables auxiliares

La cantidad de variables auxiliares la indicamos en la variable `numvar`.

```
numvar = 0
```

2.6. Función objetivo

Finalmente, indicamos el sentido de la función objetivo (`maximize` o `minimize`):

```
task.putobjsense(mosek.objsense.minimize)
```

Con todos estos datos ingresado, ya podemos correr la optimización.
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, $0 \le a \le k$, 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 $i \ge j$, 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

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

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.