

Optimización Semidefinida - Segundo Cuatrimestre 2021

Problema dual 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:

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

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

$$\begin{aligned} \text{maximizar: } & \mathbf{y} \bullet \mathbf{b}, \\ \text{sujeto a: } & \sum_{i=0}^{n-1} y_i \mathbf{P}_i \preceq \mathbf{D}, \end{aligned}$$

para matrices $\mathbf{D}, \mathbf{P}_i (1 \leq i \leq s) \in \mathbb{R}^{d \times d}$ (utilizamos distinta notación para diferenciar de la notación de Mosek).

Observamos que en la forma de plantear el problema en Mosek no tenemos forma de ingresar una desigualdad lineal matricial como aparece en el problema dual. Lo que podemos hacer es definir una matrix

$$\mathbf{X} = \mathbf{D} - \sum_{i=1}^s y_i \mathbf{P}_i,$$

que debe cumplir $\mathbf{X} \succeq 0$.

Esta igualdad es la que nos da las restricciones que ingresamos en Mosek. Tenemos una restricción para cada casilla de \mathbf{X} en la parte triangular inferior.

Considremos entonces:

- p es la cantidad de matrices semidefinidas positivas \mathbf{X} . Por lo tanto, tenemos $p = 1$ y una única matrix $\bar{\mathbf{X}}_0$.
- El vector \mathbf{x} es un vector de variables auxiliares, que en este caso corresponde a las n variables y_0, \dots, y_{n-1} del problema.
- En la función a maximizar no aparecen las coordenadas de $\bar{\mathbf{X}}_0$, por lo tanto tomamos $\bar{\mathbf{C}}_0 = \mathbf{0}$.
- La cantidad de restricciones es $m = \frac{d(d+1)}{2}$.

- No hay cotas para las variables x_j .

Por lo tanto, las restricciones corresponden a las coordenadas de la ecuación $\bar{\mathbf{X}}_0 + \sum_{i=1}^s y_i \mathbf{P}_i = \mathbf{D}$ el problema dual planteado en el formato de Mosek es

$$\begin{aligned} & \text{maximizar: } \sum_{j=0}^{n-1} b_j y_j \\ & \text{sujeto a: } \bar{\mathbf{X}}_0 + \sum_{j=0}^{n-1} y_j \mathbf{P}_j = \mathbf{D}, \\ & \bar{\mathbf{X}}_0 \succeq 0 \end{aligned}$$

donde la restricción $\bar{\mathbf{X}}_0 + \sum_{j=0}^{n-1} y_j \mathbf{P}_j = \mathbf{D}$ debemos ingresarla como $\frac{d(d+1)}{2}$ restricciones.

2. Ejemplo

Para calcular el mínimo autovalor de una matriz $\mathbf{A} \in \mathcal{S}^n$, planteamos el problema

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

Tomamos como ejemplo

$$\mathbf{A} = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix}.$$

Renombrando α como y_0 , la función a maximizar es y_0 . La ecuación $\mathbf{X} + y_0 \mathbf{I} = \mathbf{A}$ es

$$\begin{pmatrix} x_{11} & x_{12} \\ x_{12} & x_{22} \end{pmatrix} + y_0 \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 3 & 2 \\ 2 & 1 \end{pmatrix},$$

que nos da las ecuaciones:

$$\begin{aligned} x_{11} + y_0 &= 3 \\ x_{21} &= 2 \\ x_{22} + y_0 &= 1 \end{aligned}$$

3. Implementación

Veamos cómo implementar el problema dual en Mosek. Debemos ingresar:

1. la dimensión d de la matrix $\bar{\mathbf{X}}_0$,
2. la matrix $\bar{\mathbf{C}}_0 = \mathbf{0}$,
3. la cantidad n de variables auxiliares y_j (que en la notación de Mosek son las variables x_j),
4. los coeficientes c_j de las variables auxiliares y_j en la función objetivo,
5. los coeficientes a_{ij} de las variables y_j en las restricciones, $i = 0, \dots, \frac{d(d+1)}{2} - 1$,
6. las matrices $\bar{\mathbf{A}}_{i0}, i = 0, \dots, \frac{d(d+1)}{2} - 1$ de las restricciones,
7. las cotas $l_i^c = u_i^c = b_i, i = 0, \dots, \frac{d(d+1)}{2} - 1$
8. por último si buscamos el máximo o el mínimo de la función objetivo.

3.1. Dimension d de la matrix \bar{X}_0

Definimos una variable `BARVARDIM` como una lista de los tamaños de las matrices \bar{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)
```

3.2. Matriz \bar{C}_0 de coeficientes

Como tenemos $\bar{C}_0 = \mathbf{0}$, no necesitamos ingresar nada en este paso.

3.3. Variables auxiliares $y_j, 1 \leq j \leq n$ y cotas de estas variables

Definimos la cantidad de variables auxiliares:

```
numvar = n
```

(reemplazando n por la cantidad correcta) y las agregamos al problema:

```
task.appendvars(numvar)
```

Podemos fijar cotas para estas variables. Como en el problema dual, las variables y_j no están acotadas, ingresamos

```
for j in range(numvar):
    # Set the bounds on variable j
    # blx[j] <= x_j <= bux[j]
    task.putvarbound(j, mosek.boundkey.fr, -inf, +inf)
```

3.4. Coeficientes c_j de las variables auxiliares y_j en la función objetivo

Para cada variable auxiliar y_j que aparece en la función objetivo, indicamos el coeficiente c_j mediante el comando `putcj`:

```
task.putcj(j, c_j)
```

En el ejemplo, la única variable auxiliar en la función objetivo es y_0 con coeficiente 1. Ingresamos

```
task.putcj(0, 1.0)
```

3.5. Matrices \bar{A}_i de restricciones

Ingresamos las matrices \bar{A}_i también en forma esparsa, ingresando las coordenadas de la parte triangular inferior de la matriz. Si hay m matrices \bar{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{A}_i :

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

Ejemplo. Para las ecuaciones

$$x_{11} + y_0 = 3$$

$$x_{21} = 2$$

$$x_{22} + y_0 = 1$$

las matrices son

$$\bar{A}_0 = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}, \quad \bar{A}_1 = \begin{pmatrix} 0 & 0,5 \\ 0,5 & 0 \end{pmatrix} \text{ y } \bar{A}_2 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}.$$

Ingresamos tres listas en cada variable, la i -ésima lista en cada variable corresponde a la matriz \bar{A}_i :

```
barai = [[0], [1], [1] ]
baraj = [[0], [0], [1] ]
baraval = [[1.0], [0.5], [1.0]]
```

Cargamos las matrices \bar{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])
```

3.6. Coeficientes a_{ij} de la variable y_j en la restricción i

Definimos dos listas `asub` y `aval`. Cada lista contiene m listas, la i -ésima lista corresponde a las variables de la i -ésima restricción:

- en la i -ésima lista de `asub` ingresamos los índices de las variables auxiliares que aparecen en la i -ésima restricción,
- en la i -ésima lista de `aval` ingresamos los coeficientes de las respectivas variables auxiliares.

En el **ejemplo** aparece la variable y_0 en las restricciones $i = 0$ y $i = 2$, con coeficiente 1 en ambas ecuaciones. Ingresamos:

```
# Below is the sparse representation of the A
# matrix stored by row.
asub = [[0] , [ ], [0] ]
aval = [[1.0], [ ], [1.0]]
```

Cargamos estos coeficientes en el problema:

```
for i in range(numcon):
    # Input row i of A
    task.putarow(i, # Constraint (row) index.
                 asub[i], # Column index of non-zeros in constraint i.
                 aval[i]) # Non-zero values of row i.
```

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

Para definir las cotas en las restricciones

$$l_i^c \leq \sum_{j=0}^{n-1} a_{ij}x_j + \sum_{j=0}^{p-1} \bar{A}_{ij} \bullet \bar{X}_j \leq u_i^c, \quad i = 0, \dots, m - 1,$$

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 las restricciones

$$\begin{aligned} x_{11} + y_0 &= 3 \\ x_{21} &= 2 \\ x_{22} + y_0 &= 1 \end{aligned}$$

ingresamos

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

# Bound values for constraints
blc = [3.0, 2.0, 1.0]
buc = [3.0, 2.0, 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])
```

3.8. 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:

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

(numeramos filas y columnas a partir de 0 para compatibilidad con Python).

Comparando con la forma general del problema SDP en Mosek descrito en <https://docs.mosek.com/latest/pythonapi/tutorial-sdo-shared.html>, tenemos:

1. $p = 1$, hay una sola matriz \mathbf{X} sobre la que tenemos la condición $\mathbf{X} \succeq 0$.
2. Por lo tanto, el único valor de j es $j = 0$ y $\bar{\mathbf{X}}_0 = \begin{pmatrix} x_{00} & x_{01} \\ x_{01} & x_{11} \end{pmatrix}$.
3. $r_0 = 2$, es la dimensión de la matrix \mathbf{X}_0 .
4. La función a minimizar es $0x_{00} + 0x_{01} + 1x_{11}$, que con la notación $\langle \mathbf{A}, \mathbf{B} \rangle = \sum \sum a_{ij}b_{ij}$, 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, $\bar{\mathbf{C}}_0 = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$.

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 \mathbf{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

$$\bar{\mathbf{A}}_{00} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \text{y} \quad \bar{\mathbf{A}}_{10} = \begin{pmatrix} 0 & 1/2 \\ 1/2 & 0 \end{pmatrix}.$$

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 $\bar{\mathbf{C}}_0$ 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 \mathbf{C} por la fórmula

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

En nuestro ejemplo hay un solo coeficiente no nulo en $\bar{\mathbf{C}}_0$: $c_{11} = 1$ y por lo tanto definimos

```
barci = [1]
barcj = [1]
barcval = [1.0]
```

2. De la misma forma definimos las matrices $\bar{\mathbf{A}}_{i0}$. 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 \mathbf{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

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

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 \mathbf{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, [syms0], [1.0])
task.putbaraij(1, 0, [syms1], [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)

```

4. Ejemplo 2

Calcular el menor autovalor de

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

resolviendo el problema SDP:

maximizar: α
 sujeto a: $\mathbf{A} - \alpha \mathbf{I} \succeq 0$.

En este caso tenemos

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

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.