Quadprog

Seccia Ruggiero
PhD candidate in Operations Research
ruggiero.seccia@uniroma1.it

While solving an optimization problem is essential to take into consideration the specific structure of the problem we are solving so that the optimization framework can leverage the specific structure to speed up the optimization process.

Different libraries for optimization:

- scipy.optimize
- CVX
- quadprog
- lapack...

In this tutorial we will see how to use quadprog for optimizing **quadratic strictly convex** optimization problems (does it reminds you anything?).

We want to solve the following optimization problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \qquad & \frac{1}{2}(x^2 + y^2) + 2x + y \\ & 2x + y = 200 \\ & x + y \geq 100 \\ & x, y \geq 0 \end{aligned}$$

Let's import the useful packages

In [2]:

```
import numpy as np
import quadprog
```

Let's have a look on how quadprog (https://github.com/rmcgibbo/quadprog/blob/master/quadprog/quadprog.pyx) works and what it wants as input

 quadprog%20input.PNG

We need to write the problem in a matrix form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \frac{1}{2} x^T G x - a^T x \\ & C^T x \geq b \end{aligned}$$

where

$$G = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad a = \begin{bmatrix} -2 \\ -1 \end{bmatrix} \qquad C^T = \begin{bmatrix} 2 & 1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Note that if you want to add equality constraints, they have to be the first rows in the matrix C and when calling the solver you must specify how many rows must be considered as equality constraints (*meq*)

In [3]:

```
G=np.eye(2)
a=np.array([-2,-1],dtype=float).reshape(-1,)

# equality constraints
C_eq=np.array([2,1],dtype=float).reshape(1,2)
b_eq=np.array([200],dtype=float)

# inequality constraints
C_ineq=np.array([[1,1],[1,0],[0,1]],dtype=float)
b_ineq=np.array([100,0,0],dtype=float)

# put all together
C_tot=np.concatenate((C_eq,C_ineq)).T
b=np.concatenate((b_eq,b_ineq)).reshape(-1,)
print('shape of C', C_tot.shape, 'b',b.shape)
```

```
shape of C (2, 4) b (4,)
```

In [4]:

```
res=quadprog.solve_qp(G, a,C_tot, b, meq=1)
print('optimal solution:',res[0])
```

```
optimal solution: [80. 40.]
```

Some important things to note:

- we need to specify the type of the arrays (the source code accepts ony float numbers)
- watch out to the dimension of the constraints
- we do not need to specify the gradient of the function. Given a quadratic function, quadprog already knows its gradient: $\nabla f(x) = Gx - a$