Slalom_Std

April 17, 2019

0.0.1 Slalom Problem

```
In [2]: from cvxpy import *
        import numpy as np
```

Here, we give you the inputs for the problem : the x, y, and c coordinates as given in the problem statement (refer to the table)

```
In [3]: x0, y0 = 0, 4
        x1, y1, c1 = 4, 5, 3
        x2, y2, c2 = 8, 4, 2
        x3, y3, c3 = 12, 6, 2
        x4, y4, c4 = 16, 5, 1
        x5, y5, c5 = 20, 7, 2
        x6, y6 = 24, 4
```

Initialize the variables we are optimizing over here! You should be using cvx.Variable() to create the variables you are optimizing over.

```
In [4]: # Initialize any and all cvxpy variables that you will use
        t_1 = Variable()
        t_2 = Variable()
        t_3 = Variable()
        t_4 = Variable()
        t_5 = Variable()
        t_6 = Variable()

        x_0 = np.array([x0, y0])
        x_1 = Variable(2)
        x_2 = Variable(2)
        x_3 = Variable(2)
        x_4 = Variable(2)
        x_5 = Variable(2)
        x_6 = np.array([x6, y6])
        c_1_plus, c_1_minus = np.array([x1, y1 + c1]), np.array([x1, y1 - c1])
        c_2_plus, c_2_minus = np.array([x2, y2 + c2]), np.array([x2, y2 - c2])
        c_3_plus, c_3_minus = np.array([x3, y3 + c3]), np.array([x3, y3 - c3])
        c_4_plus, c_4_minus = np.array([x4, y4 + c4]), np.array([x4, y4 - c4])
```

```
        c_5_plus, c_5_minus = np.array([x5, y5 + c5]), np.array([x5, y5 - c5])

        norm_1 = norm(x_0 - x_1)
        norm_2 = norm(x_1 - x_2)
        norm_3 = norm(x_2 - x_3)
        norm_4 = norm(x_3 - x_4)
        norm_5 = norm(x_4 - x_5)
        norm_6 = norm(x_5 - x_6)
```

In [5]:
```
        # Now, we put in our constraints: the format should be as follows.

        constraints = [
            norm_1 <= t_1,
            norm_2 <= t_2,
            norm_3 <= t_3,
            norm_4 <= t_4,
            norm_5 <= t_5,
            norm_6 <= t_6,
            x_1 <= c_1_plus,
            x_1 >= c_1_minus,
            x_2 <= c_2_plus,
            x_2 >= c_2_minus,
            x_3 <= c_3_plus,
            x_3 >= c_3_minus,
            x_4 <= c_4_plus,
            x_4 >= c_4_minus,
            x_5 <= c_5_plus,
            x_5 >= c_5_minus,
        ]
```

In [6]:
```
        # Here, input your objective function. It should be of the form:
        objective_fn = sum([t_1,t_2, t_3, t_4, t_5,t_6])
        objective = Minimize(objective_fn)
```

In [7]:
```
        # creating the optimization problem here, putting together the objective and the const
        prob = Problem(objective, constraints)

        optimal_path_length = prob.solve() # this will output your optimal path length
        print(optimal_path_length)
```

```
24.14809001515744
```

Just check that your optimization variables respect the constraints here (OPTIONAL, but good for debugging)

In [8]:
```
        # Constraints Respected?
        vector_variables = [x_1.value, x_2.value, x_3.value, x_4.value, x_5.value]
        norm_variables = [t_1.value, t_2.value, t_3.value, t_4.value, t_5.value, t_6.value]
```

```
norms = [norm_1, norm_2, norm_3, norm_4, norm_5, norm_6]
minus = [c_1_minus, c_2_minus, c_3_minus, c_4_minus, c_5_minus]
plus = [c_1_plus, c_2_plus, c_3_plus, c_4_plus, c_5_plus]


for i in range(len(vector_variables)):
    print(minus[i], vector_variables[i], plus[i], "The middle quantity should be in the


for i in range(len(norm_variables)):
    print(norm_variables[i], norms[i].value, "The left should be less than or equal to
```

```
[4 2] [4.         4.19998991] [4 8] The middle quantity should be in the middle of the two
[8 2] [8.         4.39999153] [8 6] The middle quantity should be in the middle of the two
[12  4] [12.          4.59998546] [12  8] The middle quantity should be in the middle of the tw
[16  4] [16.          4.80001536] [16  6] The middle quantity should be in the middle of the tw
[20  5] [20.  5.] [20  9] The middle quantity should be in the middle of the two
4.004996374196873 4.004996374836674 The left should be less than or equal to the right
4.00499695928816 4.0049969602229565 The left should be less than or equal to the right
4.004996574644494 4.004996575398124 The left should be less than or equal to the right
4.004998371103788 4.004998372341634 The left should be less than or equal to the right
4.004996110780737 4.004996111958084 The left should be less than or equal to the right
4.123105625143389 4.123105626149096 The left should be less than or equal to the right
```

0.0.2 Print out the coordinates of the path (this should be an array with 7 tuples denoting the (x,y) position of where the skiier should cross

```
In [11]: print(np.array([x0,y0]))
         for crossing in vector_variables:
             print(crossing)
         print(np.array([x6, y6]))
```

```
[0 4]
[4.         4.19998991]
[8.         4.39999153]
[12.          4.59998546]
[16.          4.80001536]
[20.  5.]
[24  4]
```

UNIVERSITY OF CALIFORNIA, BERKELEY

HW10 : EE127

Oscar Ortega

April 18, 2019

1 CVX INSTALLATION

- a: done
- b: done

2 A SLALOM PROBLEM

- a: In general, the slalom problem can be formulated as the following optimization problem:

$$\min_{x_i \in R^2} \sum_{i=1}^{n-1} \|x_i - x_{i+1}\|_2$$

$$\text{s.t } \begin{bmatrix} x_i & y_i \end{bmatrix}^T \leq \begin{bmatrix} x_i & y_i + \frac{c}{2} \end{bmatrix}^T : i = 1, ..., n \tag{2.1}$$

$$-\begin{bmatrix} x_i & y_i \end{bmatrix}^T \leq -\begin{bmatrix} x_i & y_i - \frac{c}{2} \end{bmatrix}^T : i = 1, ..., n \tag{2.2}$$

We can also convert to a standard form by using an epigraphic reformulation:

$$\min_{x_i \in \mathbf{R}^2, t \in \mathbf{R}} \sum_{i=1}^{n-1} t_i$$

$$\text{s.t } \begin{bmatrix} x_i & y_i \end{bmatrix}^T \leq \begin{bmatrix} x_i & y_i + \frac{c}{2} \end{bmatrix}^T : i = 1, ..., n \tag{2.3}$$

$$-\begin{bmatrix} x_i & y_i \end{bmatrix}^T \leq -\begin{bmatrix} x_i & y_i - \frac{c}{2} \end{bmatrix}^T : i = 1, ..., n \tag{2.4}$$

$$\|x_i - x_{i+1}\|_2 \leq t_i : i = 1, ..., n \tag{2.5}$$

b:on iPython Notebook

3 Formulating Problems as LPs or QPs

- a: Consider the following optimization problem:

$$\min_x \|Ax - y\|_\infty + \|x\|_1$$

This is equivalent to

$$\min_{x,z} \|z\|_\infty + \|x\|_1$$

$$\text{s.t } Ax - y = z$$

Which is equivalent to:

$$\min_{x,z} \max_{i=1,...,n} z_i + \sum_{i=1}^{n} m_i$$

$$\text{s.t } Ax - y = z$$

$$m_i \leq |x_i| : i = 1, ..., n$$

Which is equivalent to the following:

$$\min_{x,t,z,m} t + \sum_{i=1}^{n} m_i$$

$$\text{s.t } Ax - y = z$$

$$m_i \leq |x_i| : i = 1, ..., n$$

$$t \leq |z_i| : i = 1, ..., n$$

Where replacing the absolute values constraints with the appropriate transformation $m_i \leq x_i$ and $-m_i \leq -x_i$ and moving everything over i.e $m_i \leq x_i \rightarrow m_i - x_i \leq 0$, will result in the standard form of an Linear Program.

- b:

$$\min_x \|Ax - y\|_2^2 + \|x\|_1$$

Is equivalent to the following QP:

$$\min_x x^T Ax - 2x^T Ay + \sum_{i=1}^{n} m_i$$

$$\text{s.t } m_i \leq |x_i| : i = 1, ..., n$$

Here, we can apply the same transformation described on the inequality constraints to put this in the standard form for a QP. Note that this objective function is convex as the sum of convex functions is a convex function.

- c: If we consider the following optimization problem:

$$\min_x \|Ax - y\|_2^2 - \|x\|_1$$

This will be equivalent to the following optimization problem:

$$\min_x x^T Ax - 2x^T Ay - \sum_{i=1}^{n} m_i$$

$$\text{s.t } m_i \leq |x_i| : i = 1, ..., n$$

However, if we note that the objective, a difference of two convex function, is not convex we can conclude that this cannot be considered a QP.

.

- d: Consider the form of $\|x\|_1^2$

$$\|x\|_1^2 = \sum_{i,j} |x_i||x_j|$$

As we can see, the function is not convex over its domain. So if we consider the form of the optimization problem:

$$\min_x \|Ax - y\|_2 + \|x\|_1^2$$

We can see that the objective is not convex, and cannot be formulated into a convex QP.

4 An LP with wide matrix

$$p^* = \min_x c^T x : l \leq Ax \leq u$$

- a: let $l \leq y \leq u$. Because the matrix A is full rank, this implies that there exists an $x \in \mathscr{R}(A)$ s.t $Ax = y$. Therefore, the feasible set cannot be empty.

- b:
  $c \notin \mathscr{R}(A^T) \rightarrow \forall \gamma : A^T \gamma \neq c$

  Therefore, by the orthogonal decomposition theorem, we know

  $\exists \gamma, z : c = A^T \gamma + z : z \neq 0, z \in \mathscr{N}(A)$
  let $r \in \mathscr{N}(A)$

  $$c^T r = (A^T \gamma + z)^T r \tag{4.1}$$
  $$= \gamma^T A r + z^T r \tag{4.2}$$
  $$= 0 + z^T r \tag{4.3}$$

  Where we can chose z such that $z^T r$ is negative.

  Let x_0 be a feasible point and let $x = x_0 + \rho$

  $$Ax = Ax_0$$

  and is therefore feasible. Our cost is now as follows

  $$c^T x = c^T x_0 + \rho z^T r \tag{4.4}$$
  $$\lim_{\rho \rightarrow \infty} = -\infty \tag{4.5}$$

  And our function is unbounded from below.

- c:
  Let $c \in \mathscr{R}(A^T)$ $\exists d \rightarrow A^T d = c$
  By the orthogonal decomposition theorem, we know there exists γ, z s.t $x = A^T \gamma + z$: $z \in \mathscr{N}(A)$ for every $x \in \mathbb{R}^n$. Furthermore, because we know the $c = A^T d$ for some d this shows that orthogonal component z in this representation should be 0.
  Let x_0 be a feasible point in the set.

  $$c^T x_0 = (A^T d)^T x_0 \tag{4.6}$$
  $$= d^T A x_0 \tag{4.7}$$

  Using the orthogonal decomposition theorem for x_0

  $$= d^T A(A^T y + z) \tag{4.8}$$
  $$= d^T A A^T y + d^T A z \tag{4.9}$$
  $$= d^T (A A^T) y = c^T x \tag{4.10}$$

And our optimization problem is now the following:

$$\min_y d^T AA^T y$$

$$\text{s.t } l \leq AA^T y \leq u$$

- d: Defining $y' = AA^T y$ We can reformulate the following minimization problem as follows:

$$\min_{y'} d^T y'$$

$$\text{s.t } l \leq y' \leq u$$

We can solve for the optimal value d^* as follows:

1. if $c \notin \mathscr{R}(A^T) : d^* = -\infty$
2. else: find d s.t $A^T d = c$ and y s.t $A^T y = x$
   a) let $y' = AA^T y$
   b) end up with optimization of the following: $\max_{y' \in \{l,u\}} d^T y' = d^*$

5 SPHERE ENCLOSURE

Credit to the Boyd Book: Recall the definition of the Euclidean Ball:

$$B(x_c, r) = \{x : \|x - x_c\|_2 \leq r\}$$

Because we want to find the radius euclidean ball of smallest radius that encompasses all the other euclidean balls $B_i((x_c)_i, r_i) \in \mathscr{B} = \{B_1, ..., B_m\}$ Therefore we can format this in SOCP Form as follows:

$$\min_{x_c, t} t$$

$$\text{s.t } \|x_c\| \leq t \tag{5.1}$$

$$\|x_c - (x_c)_i\| \leq t - r_i : i = 1, ..., m \tag{5.2}$$

Where the first constraint ensures that the encompassing ball B is smaller than the radius which is equal to t and the other constraints make sure that the ball B_i is also within the encompassing ball B. It helps to view this problem from the viewpoint of x_c as the origin.