Geometric Problems

Arpit Bhatia

June 20, 2019

These problems in this tutorial are drawn from Chapter 8 of the book Convex Optimization by Boyd and Vandenberghe

```julia
using JuMP
using Ipopt
```

1 Euclidean Projection on a Hyperplane

For a given point x_0 and a set C, we refer to any point $z \in C$ which is closest to x_0 as a projection of x_0 on C. The projection of a point x_0 on a hyperplane $C = \{x | a' \cdot x = b\}$ is given by

$$\begin{array}{cc} \min & ||x - x_0|| \\ s.t. & a' \cdot x = b \end{array}$$

```julia
x = rand(10)
a = rand(10)
b = rand()

model = Model(with_optimizer(Ipopt.Optimizer, print_level=0))
@variable(model, x0[1:10])
@objective(model, Min, sum((x - x0) .* (x - x0))) # We minimize the square of the
    distance here
@constraint(model, x0' * a == b)                  # Point must lie on the hyperplane

optimize!(model)
@show objective_value(model)
```

```
objective_value(model) = 1.2454262479448035
```

```julia
@show value.(x0)
```

```
value.(x0) = [0.671528, -0.0342765, 0.486667, -0.0719938, 0.367341, 0.48575
8, 0.66609, 0.0593366, 0.338023, 0.0899162]
10-element Array{Float64,1}:
  0.6715279749396132
 -0.03427653682879572
  0.4866673308029252
 -0.0719937814801509
```

```
0.3673412721199711
0.4857580784037865
0.6660897125987228
0.05933660548921381
0.33802321178746164
0.08991624512589541
```

2 Euclidean Distance Between Polyhedra

Given two polyhedra $C = \{x|A_1 \cdot x \leq b1\}$ and $D = \{x|A_2 \cdot x \leq b_2\}$, the distance between them is the optimal value of the problem:

$$\begin{aligned} \min \quad & ||x - y|| \\ s.t. \quad & A_1 \cdot x \leq b_1 \\ & A_2 \cdot y \leq b_2 \end{aligned}$$

```
A_1 = rand(10,10)
A_2 = rand(10,10)
b_1 = rand(10)
b_2 = rand(10)

model = Model(with_optimizer(Ipopt.Optimizer, print_level=0))
@variable(model, x[1:10])                          # Point closest on the first polyhedron
@variable(model, y[1:10])                          # Point closest on the second polyhedron
@objective(model, Min, sum((x - y) .* (x - y))) # We minimize the square of the distance
    here as above
@constraint(model, A_1 * x .<= b_1)                # Point x must lie on the first
    polyhedron
@constraint(model, A_2 * y .<= b_2)                # Point y must lie on the second
    polyhedron

optimize!(model)
@show objective_value(model)

objective_value(model) = 2.7755575615628914e-17
2.7755575615628914e-17
```

3 Linear Placement Problem

We have N points in $\mathbb{R}^2$, and a list of pairs of points that must be connected by links. The positions of some of the N points are fixed; our task is to determine the positions of the remaining points, i.e., to place the remaining points. The objective is to place the points so that the distance between the links is minimized, i.e. our objective is:

$$\min \quad \sum_{(i,j) \in A} ||x_i - x_j||$$

```
fixed = [ 1    1   -1  -1     1    -1   -0.2   0.1;          # coordinates of fixed points
          1   -1   -1   1  -0.5  -0.2    -1      1]

M = size(fixed,2)                                             # number of fixed points
N = 6                                                         # number of free points

A = [ 1  0  0 -1  0  0    0  0  0  0  0  0  0  0;             # Matrix on links
      1  0 -1  0  0  0    0  0  0  0  0  0  0  0;
      1  0  0  0 -1  0    0  0  0  0  0  0  0  0;
      1  0  0  0  0  0   -1  0  0  0  0  0  0  0;
      1  0  0  0  0  0    0 -1  0  0  0  0  0  0;
      1  0  0  0  0  0    0  0  0  0 -1  0  0  0;
      1  0  0  0  0  0    0  0  0  0  0  0  0 -1;
      0  1 -1  0  0  0    0  0  0  0  0  0  0  0;
      0  1  0 -1  0  0    0  0  0  0  0  0  0  0;
      0  1  0  0  0 -1    0  0  0  0  0  0  0  0;
      0  1  0  0  0  0    0 -1  0  0  0  0  0  0;
      0  1  0  0  0  0    0  0 -1  0  0  0  0  0;
      0  1  0  0  0  0    0  0  0  0  0  0 -1  0;
      0  0  1 -1  0  0    0  0  0  0  0  0  0  0;
      0  0  1  0  0  0    0 -1  0  0  0  0  0  0;
      0  0  1  0  0  0    0  0  0  0 -1  0  0  0;
      0  0  0  1 -1  0    0  0  0  0  0  0  0  0;
      0  0  0  1  0  0    0  0 -1  0  0  0  0  0;
      0  0  0  1  0  0    0  0  0 -1  0  0  0  0;
      0  0  0  1  0  0    0  0  0  0  0 -1  0  0;
      0  0  0  1  0  0    0  0  0  0  0 -1  0  0;
      0  0  0  0  1 -1    0  0  0  0  0  0  0  0;
      0  0  0  0  1  0   -1  0  0  0  0  0  0  0;
      0  0  0  0  1  0    0  0  0 -1  0  0  0  0;
      0  0  0  0  1  0    0  0  0  0  0  0  0 -1;
      0  0  0  0  0  1    0  0 -1  0  0  0  0  0;
      0  0  0  0  0  1    0  0  0  0 -1  0  0  0;]

model = Model(with_optimizer(Ipopt.Optimizer, print_level=0))
@variable(model, x[1:M + N,1:2])                              # A variable array for the
    coordinates of each point
@constraint(model, x[N + 1:N + M,:] .== fixed')               # We had a constraint for the fixed
    points
dist = A * x                                                  # Matrix of differences between
    coordinates of 2 points with a link
@objective(model, Min, sum(dist .* dist))                     # We minimize the sum of the square
    of the distances

optimize!(model)
@show value.(x)

value.(x) = [0.44791 0.0468982; -0.0319353 -0.670614; 0.404336 -0.451308; -
0.394295 -0.132825; 0.025327 0.412421; -0.00165206 -0.439548; 1.0 1.0; 1.0
-1.0; -1.0 -1.0; -1.0 1.0; 1.0 -0.5; -1.0 -0.2; -0.2 -1.0; 0.1 1.0]

@show objective_value(model)

objective_value(model) = 20.44474039109911
20.44474039109911
```

4 Floor Planning

A floor planning problem consists of rectangles or boxes aligned with the axes which must be placed, within some limits such that they do not overlap. The objective is usually to minimize the size (e.g., area, volume, perimeter) of the bounding box, which is the smallest box that contains the boxes to be configured and placed. We model this problem as follows:

We have N boxes $B_1, ..., B_N$ that are to be configured and placed in a rectangle with width W and height H, and lower left corner at the position $(0,0)$. The geometry and position of the ith box is specified by its width w_i and height h_i, and the coordinates (x_i, y_i) of its lower left corner.

The variables in the problem are x_i, y_i, w_i, h_i for $i = 1, \ldots, N$, and the width W and height H of the bounding rectangle. In all floor planning problems, we require that the cells lie inside the bounding rectangle, *i.e.*

$$x_i \geq 0, \quad y_i \geq 0, \quad x_i + w_i \leq W, \quad y_i + h_i \leq H, \quad i = 1, \ldots, N$$

We also require that the cells do not overlap, except possibly on their boundaries, *i.e.*

$$x_i + w_i \leq x_j,$$

or $x_j + w_j \leq x_i$, or $y_i + h_j \leq y_j$, or $y_j + h_i \leq y_i$

```
n = 5;

Amin = [                                        # We'll try this problem with 4 times
    with different minimum area constraints
100 100 100 100 100;
 20  50  80 150 200;
180  80  80  80  80;
 20 150  20 200 110]

r = 1

for i = 1:4
    A = Amin[i,:]

    model = Model(with_optimizer(Ipopt.Optimizer, print_level=0))

    @variable(model, x[1:n] >= r)
    @variable(model, y[1:n] >= r)
    @variable(model, w[1:n] >= 0)
    @variable(model, h[1:n] >= 0)
    @variable(model, W)
    @variable(model, H)

    @constraint(model, x[5] + w[5] + r <= W)     # No rectangles at the right of Rectangle
    5
    @constraint(model, x[1] + w[1] + r <= x[3]) # Rectangle 1 is at the left of Rectangle
    3
    @constraint(model, x[2] + w[2] + r <= x[3]) # Rectangle 2 is at the left of Rectangle
    3
    @constraint(model, x[3] + w[3] + r <= x[5]) # Rectangle 3 is at the left of Rectangle
    5
    @constraint(model, x[4] + w[4] + r <= x[5]) # Rectangle 4 is at the left of Rectangle
    5
```

```julia
    @constraint(model, y[4] + h[4] + r <= H)     # No rectangles on top of Rectangle 4
    @constraint(model, y[5] + h[5] + r <= H)     # No rectangles on top of Rectangle 5
    @constraint(model, y[2] + h[2] + r <= y[1]) # Rectangle 2 is below Rectangle 1
    @constraint(model, y[1] + h[1] + r <= y[4]) # Rectangle 1 is below Rectangle 4
    @constraint(model, y[3] + h[3] + r <= y[4]) # Rectangle 3 is below Rectangle 4
    @constraint(model, w .<= 5*h)                # Aspect ratio constraint
    @constraint(model, h .<= 5*w)                # Aspect ratio constraint
    @constraint(model, A .<= h .*  w)            # Area constraint

    @objective(model, Min, W + H)

    optimize!(model)
    @show objective_value(model)
end
```

```
objective_value(model) = 51.93446154361733
objective_value(model) = 51.1562218755639
objective_value(model) = 52.669203332919835
objective_value(model) = 52.54574619621182
```