MATH-566 HW 5

Due **Oct 5** before class. Just bring it before the class and it will be collected there.

1: (*Analytic center*)
Let S be defined as intersection of halfspaces $x_i \geq 0$ and $(1-x_i)^k \geq 0$. Suppose $i \in \{1, 2, \ldots, d\}$ and $k \geq 1$ is odd. Compute the analytic center of S. Notice that for x satisfying $(1-x_i)^k \geq 0$, the function $(1-x_i)^k$ is convex.

2: (*Central path*)
Compute central path for the following problem

$$(P) \begin{cases} \text{minimize} & -x_1 \\ \text{subject to} & x_1 \leq 1 \\ & x_2 \leq 1 \\ & x_1 \geq 0 \\ & x_2 \geq 0 \end{cases}$$

and find the optimal solution using the central path. Plot (sketch) the set of feasible solutions and the computed central path. Lot of calculus...
Hint: The central path is formed by points that are optimal solutions to

$$min\ h_t(x_1, x_2) = -tx_1 + \Phi(x_1, x_2),$$

where $t \geq 0$ and $\Phi(x_1, x_2)$ is the barrier function. Take partial derivatives of $h_t(x_1, x_2)$ to obtain optimal solution $(x_1, x_2)_t$. For $t \in [0, \infty]$ these points of optimal solutions will form a curve. Plot or describe the curve. This curve is the central path.

3: (*Alternative attempt to define minimum spanning tree as LP*)
Let $G = (V, E)$ and $|V| = n$.

Recall that the spanning tree polytope was created by constraints *tree has $n-1$ edges* and *tree has no cycles*. Formally,

$$STP = \left\{ \mathbf{x} \in [0,1]^{E(G)} : \sum_{e \in E} x_e = n-1, \sum_{uv \in E, u \in X, v \in X} x_{(u,v)} \leq |X| - 1 \text{ for } \emptyset \subset X \subset V \right\}.$$

Suppose we try to characterize the spanning tree by assuming that by constraints *tree has $n-1$ edges* and *tree is connected*. The tree is connected can be formulated by saying that for every cut, the sum x_e of edges e in the cut is at least one. Formally,

$$P = \left\{ \mathbf{x} \in [0,1]^{E(G)} : \sum_{e \in E} x_e = n-1, \sum_{uv \in E, u \in X, v \notin X} x_{(u,v)} \geq 1 \text{ for } \emptyset \subset X \subset V \right\}.$$

1. Prove that the spanning tree polytope is a subset of P. That is, $STP \subseteq P$.

2. Show P does NOT have to be the same as the spanning tree polytope. To do this, show that the polytope P does NOT have to be integral (i.e., P contains a vertex that does not have all coordinates integers).

Hint: See the book Combinatorial Optimization from Korte and Vygen on the bottom of the page 150. Free PDF available from ISU library.

4: (*Programming spanning tree*)
Implement any minimum spanning tree algorithm and test it on random data. You can pick any algorithm you like. You can use ANY programming language but you have to IMPLEMENT the method yourself (calling a library function `RunKruskal` is not acceptable). Obtain data by randomly generating 10 points in range $[0, 10]^2$ and the cost of every edge is the Euclidean distance in $\mathbb{R}^2$. We consider all 45 edges of K_{10}. Finally, create the plot of of the random points and draw edges picked to the spanning tree. You should provide: Name of the algorithm you implemented and short description of implementation, printout of the source code, pictures of two solutions.
Template is provided for Sage, you do not have to use it.
Time complexity DOES NOT matter.

```
# MATH 566 - Minimum spanning tree algorithm
# Notes:
#  - pick any algorithm for minimum spanning three you like
#  - no need to optimize the running time
#

# This plots vertices as red dots and blue edges connecting them
def plot_vertices_edges(vertices, edges):
    drawing =  line([])
    for x in vertices:
        drawing = drawing + disk(x, 0.1, (0,2*pi), color='red')
    for e in edges:
        drawing =  drawing + line([vertices[e[0]], vertices[e[1]]])
    drawing.show()

# Generate 10 random vertices in 10x10 grid
def generate_random_vertices():
    vertices=[]
    for i in range(10):
        vertices.append((random()*10, random()*10))
    return vertices

# This is the function that you need to write
def compute_minimum_spannig_tree(vertices):
    n = len(vertices)
    edges_in_tree = [[0,1],[1,2],[5,6]]
```

```
    edges_in_tree = []
    vertex_components = range(n)

    # Write your code here please....


    return edges_in_tree


vertices = generate_random_vertices()
edges = compute_minimum_spannig_tree(vertices)

plot_vertices_edges(vertices, edges)
```