Semidefinite and Second Order Cone Programming Seminar
Fall 2012
Lecture 1

Instructor: Farid Alizadeh
Scribe: Gyorgy Matyasfalvi

9/10/2012

1 Overview

First we introduce Semi Definite Programming (SDP) and show that Linear Programming (LP) and Second Order Cone Programming (SOCP) are special cases of SDP. Secondly we go over applications of SDP namely the MAXCUT, the Shape Constraint Regression and the Single Facility Plant Location Problem.

2 Program Formulations

The linear and semidefinite programming problems are formulated as follows:

2.1 Standard Form Linear Programming

$$\begin{array}{ll} \text{min:} & c_1x_1 + \ldots + c_nx_n \\ \text{s.t.} & a_{11}x_1 + \ldots + a_{1n}x_n = b_1 \\ & \vdots \\ & a_{m1}x_1 + \ldots + a_{mn}x_n = b_m \\ & x_i \geq 0, \quad \forall i \end{array} \tag{1}$$

2.2 Semidefinite Programming

Here instead of a_{ij} we use symmetric matrices $A_{ij} \in \mathbb{S}^{n\times n}$ (the set of $n \times n$ *symmetric matrices*), $i = 1, ..., m$, $j = 1, ..., n$ $C_j \in \mathbb{S}^{n\times n}$ and $X_j \in \mathbb{S}^{n\times n}$ instead

of c_j and x_j. The matrix X_j is positive semidefinite. The inner product is defined as

$$\begin{aligned} A \bullet B &= \sum_{i,j} A_{ij} B_{ij} \\ &= \text{Trace}(AB^T) \\ &= \text{Trace}(AB) = \text{Trace}(BA). \end{aligned}$$

We get the second equation from the definition of the product, the last one from the observation that the diagonal entries of AB and BA are equal and thus their traces are equal as well (in general matrix product is not commutative, i.e. $AB \neq BA$). The standard form of semidefinite programming is :

$$\begin{array}{ll} \min & C_1 \bullet X_1 + \ldots C_n \bullet X_n \\ \text{s.t.} & A_{11} \bullet X_1 + \ldots + A_{1n} \bullet X_n = b_1 \\ & \vdots \\ & A_{m1} \bullet X_1 + \ldots + A_{mn} \bullet X_n = b_m \\ & X_i \succcurlyeq 0, \quad \forall i \end{array}$$

3 Linear Algebra

Let V be a nonzero vector-space of finite dimension over the field T and $\mathcal{X} \in \text{Hom}V$ an arbitrary linear transformation.

Definition 1 (Eigenvalue) *A $\lambda \in T$ scalar is called the Eigenvalue of the linear transformation X if $\exists \boldsymbol{v} \in V(\boldsymbol{v} \neq 0) : X\boldsymbol{v} = \lambda \boldsymbol{v}$.*

Definition 2 (Eigenvector) *A $\boldsymbol{v} \in V(\boldsymbol{v} \neq 0)$ vector is called the Eigenvector of linear transformation X if $\exists \lambda \in T$ scalar : $X\boldsymbol{v} = \lambda \boldsymbol{v}$.*

Definition 3 (Positive semidefinite matrix (PSD)) *A matrix $X \in \mathbb{R}^{n \times n}$ is positive semidefinite if:*

1. *$\mathbf{a}^T X \mathbf{a} \geq 0 \; \forall \mathbf{a} \in \mathbb{R}^n$ (i.e. it is characterized by infinitely many inequalities) or*
2. *$\exists B \in \mathbb{R}^{n \times k} : X = BB^T$ or*
3. *all λ Eigenvalues of X are real.*

We denote: $X \succcurlyeq 0$.

Proposition 4 (Equivalence of PSD definitions) *Definitions 1, 2, and 3 are equivalent.*

Proof:

- $2 \Rightarrow 1$: Since $X = BB^T$ we can rewrite $\mathbf{a}^T X \mathbf{a} = \mathbf{a}^T B B^T \mathbf{a}$, let $\mathbf{a}^T B = \mathbf{y}^T$ then $\mathbf{a}^T X \mathbf{a} = \mathbf{a}^T B B^T \mathbf{a} = \mathbf{y}^T \mathbf{y}$ but $\mathbf{y}^T \mathbf{y} \geq 0$ always thus if $X = BB^T \Rightarrow \mathbf{a}^T X \mathbf{a} \geq 0$.

- $1 \Rightarrow 3$: (Recall that eigenvalues of a Hermitian–and thus a symmetric–matrix is always real.

$$\begin{array}{rcl} X\boldsymbol{v} = & \lambda \boldsymbol{v} & /\boldsymbol{v}^T \cdot \\ \boldsymbol{v}^T X \boldsymbol{v} = & \boldsymbol{v}^T \lambda \boldsymbol{v} & \\ \underbrace{\boldsymbol{v}^T X \boldsymbol{v}} = & \lambda \underbrace{\boldsymbol{v}^T \boldsymbol{v}} & \\ \geq 0 & \geq 0 & \\ & \Downarrow & \\ & \lambda \geq 0 & \end{array}$$

- $3 \Rightarrow 2$: We know that a matrix with real Eigenvalues and orthogonal Eigenvectors is symmetric. The finite-dimensional spectral theorem says that any symmetric matrix whose entries are real can be diagonalized by an orthogonal matrix. Thus $X = Q \Lambda Q^T$, where $Q^T Q = I$ and

$$\Lambda = \begin{pmatrix} \lambda_1 & & 0 \\ & \ddots & \\ 0 & & \lambda_n \end{pmatrix} \tag{2}$$

Thus we can define:

$$\underbrace{Q \begin{pmatrix} \sqrt{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda_n} \end{pmatrix}}_{B} \underbrace{\begin{pmatrix} \sqrt{\lambda_1} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda_n} \end{pmatrix} Q^T}_{B^T} \tag{3}$$

$$\tag{4}$$

and express $X = BB^T$.

■

4 Semidefinite Programming

Definition 5 (Cone) *A set $\mathcal{K}$ is called a cone if $\alpha \mathbf{x} \in \mathcal{K}$ for each $\mathbf{x} \in \mathcal{K}$ and for each $\alpha \geq 0$.*

Definition 6 (convex Cone) *A convex cone $\mathcal{K}$ is a cone with the additional property that $\mathbf{x} + \mathbf{y} \in \mathcal{K}$ for each $\mathbf{x}, \mathbf{y} \in \mathcal{K}$.*

Definition 7 (pointed Cone) *A pointed cone $\mathcal{K}$ is a cone with the property that $\mathcal{K} \cap (-\mathcal{K}) = \{0\}$. (Thus $\mathcal{K}$ does not contain any lines.)*

We require that our matrices $X_i \succcurlyeq 0$. The set of PSD matrices is a closed pointed convex cone. Let $P = \{X \in \mathbb{S}_n,\ X \succcurlyeq 0\}$.

- CONE:

$$X \in P \Rightarrow \alpha X \in P\ \forall \alpha \geq 0 \text{ since } a^T \alpha X a = \alpha a^T X a \geq 0$$

- CONVEX CONE:

$$X_1, X_2 \in P a^T (X_1 + X_2) a \quad = \quad \underbrace{a^T X_1 a}_{\geq 0} + \underbrace{a^T X_2 a}_{\geq 0} \geq 0$$

- POINTED CONE:

$$X \in P \text{ and } X \in -P \Leftrightarrow X = 0$$

The feasible set of a semidefinite program as a geometrical object is the intersection of hyperplanes by a convex cone. Linear Programs are special case of semidefinite progra,s This can be established by identifying $x_i \geq 0$ as 1×1 matrices. The cone we operate in is the non-negative orthant.

4.1 Second Order Cone Programming (SOCP)

$$\begin{array}{ll} \text{min:} & c_1 x_1 + \ldots + c_n x_n \\ \text{s.t.} & a_{11} x_1 + \ldots + a_{1n} x_n = b_1 \\ & \vdots \\ & a_{m1} x_1 + \ldots + a_{mn} x_n = b_m \\ & x_i \in Q_i, \quad \forall i \end{array} \tag{5}$$

c_i, x_i, a_i-s are vectors i and j are not necessarily of the same dimension. Q_i is the Second Order Cone (SOC) aka Lorentz Cone.

$$Q = \left\{ x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{pmatrix} : x_0 \geq \sqrt{x_1^2 + \ldots + x_n^2} \right\} \tag{6}$$

The relation between the LP problem and the SOCP problem can be established by setting $x^T = (x_0, 0, \ldots, 0)$ then our constraint becomes: $x_0 \geq 0$. For SDP problems we have to consider:

$$x \in Q:\ x_0 \geq \sqrt{x_1^2 + \ldots + x_n^2} \Leftrightarrow \mathrm{Arw}(x) = \begin{pmatrix} x_0 & x_1 & \ldots & x_n \\ x_1 & x_0 & 0 & 0 \\ \vdots & 0 & \ddots & 0 \\ x_n & 0 & 0 & x_0 \end{pmatrix} \succcurlyeq 0$$

$\mathrm{Arw}(x) \succcurlyeq 0 \ \Leftrightarrow\ x = 0$ or $x_0 > 0$ and the Schur complement $x_0 - \overline{x}^T (x_0 I)^{-1} \overline{x} \geq 0$, where $\overline{x}^T = (x_1, x_2, \ldots, x_n)$ [1].

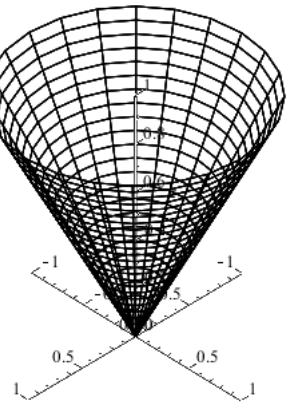

Figure 1: 3D Lorentz Cone

In linear programming, when we have only one decision variable, the problem becomes trivial. In SOCP, with only one (vector) decision variable, the problem is not as trivial, but it turns out that it can be reduced to least squares problem and solved by solving a system of equations. In SDP, a one (matrix) variable problem is as hard as multiple matrix problems.

5 Applications

5.1 MAXCUT

We know that the MINCUT problem is solvable in polynomial time it turns out that the MAXCUT problem is NP-Complete. Thus we will look at approximation algorithms for the MAXCUT problem.

Definition 8 (MAXCUT) *For a graph* $\mathsf{G} = (\mathsf{V}, \mathsf{E}, w : \mathsf{E} \to \mathbb{R}^+)$ *find a partition of* V *into two nontrivial sets (cut) such that the weight of the edges crossing the cut is maximum.*

As mentioned before there are no known good (poly. time) algorithms for the MAXCUT problem thus we will rely on approximation algorithms.

5.1.1 A Greedy Algorithm

Our greedy algorithm:

Data: Graph $G = (V, E, w)$
Result: Approximate MAXCUT: C_G
initialization $Z = V$, $X = \emptyset$, $Y = \emptyset$ pick an arbitrary vertex $v \in Z$ $v \rightarrow X$;
while $Z \neq \emptyset$ **do**
  pick $v \in Z : \exists\, e_{vX}$ or $e_{vY} \in E$;
  if $w_{vX} \geq w_{vY}$ **then**
    $v \rightarrow Y$
  else
    $v \rightarrow X$
  end
end

Where e_{vX} represents the set of edges between vertex v and the set X and w_{vX} represents the sum of all edge weights between v and X, $w_{vX} = \sum_{i \in X} w_{vi}$.

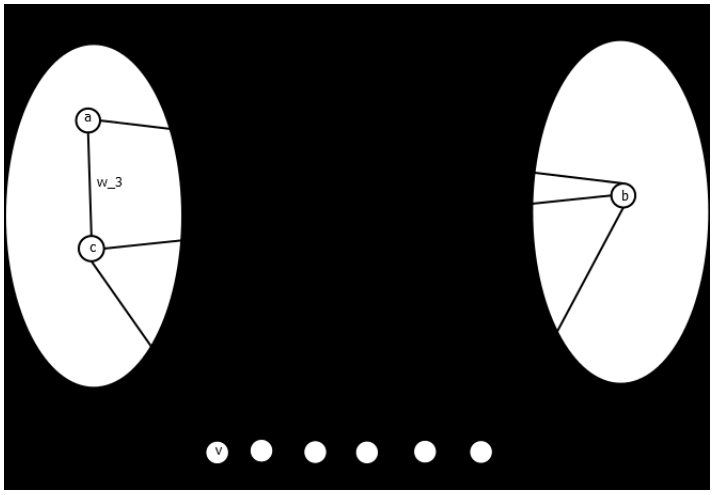

Figure 2: MAXCUT greedy

In Figure 2 we first picked vertex a placed it in X then picked vertex b since b did not have edges going to Y we placed it in Y then we picked vertex c, c had edges going to both X and Y but since $w_2 \geq w_3$ we placed it in X. The next vertex is v, which we will put in Y and so on.
This algorithm gives a $\frac{1}{2}$ approximation of the optimum MAXCUT value $C_G \geq \frac{1}{2} C_{OPT}$. Since at each iteration at least half of the edge weights emanating from the selected vertex end up in the cut.

5.1.2 Improving the $\frac{1}{2}$ bound by using SDP

Let's assign $+1$ to each vertex that we place in set X and -1 to each vertex that we place in set Y as shown in Figure 3.

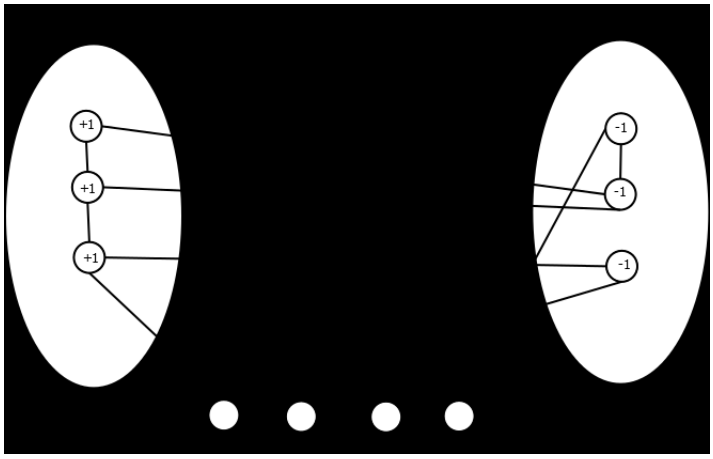

Figure 3: $+1/-1$ Assignment

Our variable x_i will correspond to vertex i and have value:

$$x_i = \begin{cases} +1 & \text{if vertex } i \in X \\ -1 & \text{if vertex } i \in Y \end{cases}$$

Thus:

$$\frac{1 - x_i x_j}{2} = \begin{cases} 0 & \text{if it doesn't contribute to the cut} \\ 1 & \text{if it contributes to the cut} \end{cases}$$

The weight of the cut can be defined in the following manner:

$$\sum_{ij \in E} w_{ij} \left(\frac{1 - x_i x_j}{2} \right)$$

Our MAXCUT formulation becomes as a quadratic programming problem:

$$\begin{array}{ll} \text{max:} & \sum_{ij \in E} w_{ij} \left(\frac{1 - x_i x_j}{2} \right) \\ \text{s.t.} & x_i^2 = 1 \quad \forall i \in V \end{array} \tag{7}$$

Let's relax our problem by associating vector decision variables to our vertices $\boldsymbol{v}_i \in \mathbb{R}^n$:

$$\begin{array}{ll} \text{max:} & \sum_{ij \in E} w_{ij} \left(\frac{1 - \boldsymbol{v}_i^T \boldsymbol{v}_j}{2} \right) \\ \text{s.t.} & \|\boldsymbol{v}_i\| = 1 \quad \forall i \in V \end{array} \tag{8}$$

We can observe that $\boldsymbol{v}_i = (1, 0, \ldots, 0)$ and $\boldsymbol{v}_j = (-1, 0, \ldots, 0)$ is a feasible solution to (8) thus we can transform any feasible solution of (7) into a feasible

solution of (8). We observe that the feasible solutions of (7) once transformed are rotation invariant i.e. if $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are solutions then for $Q^TQ = I$ $\boldsymbol{u}_i = Q\boldsymbol{v}_i$ $\boldsymbol{u}_i$ is a solution as well $\boldsymbol{u}_i^T\boldsymbol{u}_i \equiv \boldsymbol{v}_i^TQ^TQ\boldsymbol{v}_i = \boldsymbol{v}_i^T\boldsymbol{v}_i$. Another way to look at this is that all our solutions have to be on a line. Let's create a matrix V and X:

$$V = [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \ldots \boldsymbol{v}_n]\,,\ X = V^TV \tag{9}$$

thus $x_{ij} = \boldsymbol{v}_i^T\boldsymbol{v}_j$. This way we can get a slightly different however equivalent formulation of (8).

$$\begin{array}{ll} \text{max:} & \sum_{ij\in V} w_{ij}\left(\frac{1-x_{ij}}{2}\right) \\ \text{s.t.} & x_{ij} = 1 \quad \forall ij \in V \\ & X \succcurlyeq 0 \end{array} \tag{10}$$

(10) is the SDP formulation of our problem. If we imposed a rank constraint on X, $\text{rank}(X) = 1$ our formulation would be equivalent with (7). However solving problems with rank constraint are just as difficult as our original formulation. Our solutions $X = V^TV$ to (10) are unique up to rotation. We can imagine our solution vectors to (10) as depicted in Figure 4.

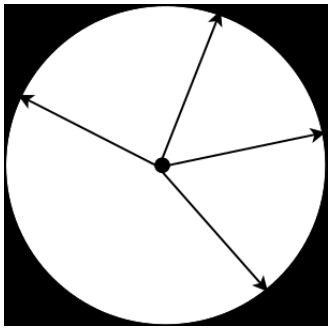

Figure 4: Solution vectors of (10)

Nonetheless as already mentioned we will need solution vectors that are on a line Figure 5.

We will collapse our solution vectors of (10) by separating them with a random hyperplane i.e. vertices corresponding to $\boldsymbol{v}_i$s on one side of the hyperplane go into X vertices on the other-side are placed in Y. Formally if the equation of our hyperplane is $\boldsymbol{r}^T\boldsymbol{x} = 0$:

$$\begin{array}{ll} \text{if } \boldsymbol{r}^T\boldsymbol{v}_i < 0 & \rightarrow x_i = -1 \quad x_i \text{ of (7)} \\ \text{if } \boldsymbol{r}^T\boldsymbol{v}_j > 0 & \rightarrow x_j = +1 \quad x_j \text{ of (7)} \end{array}$$

We will choose the coordinates of the normal vector to our plane by drawing each coordinate independently from a normal distribution and then normalize it (this way the points on our unit sphere are distributed uniformly).
Now that we have a cut all that is left is to decide how well our solution approximates the optimal.

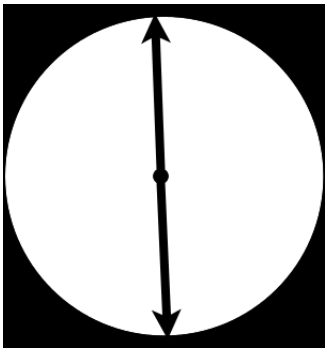

Figure 5: Solution vectors of (10) feasible for (7)

- $W(C_{SDP})$ weight of our cut obtained by SDP.
- $W(C_{OPT})$ weight of the optimal cut.
- $W(C_R)$ weight of cut obtained by random collapse.

We know that

$$W(C_{SDP}) \geq W(C_{OPT}) \geq E(W(C_R)) \tag{11}$$

We will show that the expected value of our random cuts $E(W(C_R))$ will be better than the average cut with probability close to 1. By linearity of the expected value we get the following relation:

$$\begin{aligned} E(W(C_R)) &= \sum_{ij\in E} w_{ij} \Pr(\boldsymbol{v}_i \text{ and } \boldsymbol{v}_j \text{ are on different sides of the hyperplane}) \\ &= \sum_{ij\in E} w_{ij} \frac{\arccos(\boldsymbol{v}_i^T \boldsymbol{v}_j)}{\pi} \quad \text{since } \|\boldsymbol{v}_i\| = 1 \end{aligned}$$

Question: Does there exist an $\alpha > \frac{1}{2}$ such that:

$$\begin{aligned} W(C_{SDP}) &\geq W(C_{OPT}) \geq E(W(C_R)) \geq \alpha W(C_{SDP}) \\ \sum_{ij\in E} w_{ij} \left(\frac{1-\boldsymbol{v}_i^T \boldsymbol{v}_j}{2}\right) &\geq \sum_{ij\in E} w_{ij} \frac{\arccos(\boldsymbol{v}_i^T \boldsymbol{v}_j)}{\pi} \geq \alpha \sum_{ij\in E} w_{ij} \left(\frac{1-\boldsymbol{v}_i^T \boldsymbol{v}_j}{2}\right) \end{aligned}$$

In general it turns out that the following is true:

$$\begin{aligned} \frac{\arccos(y)}{\pi} &\geq \alpha \frac{1-y}{2} \\ &\Downarrow \\ \alpha = \min_{-1\leq y\leq 1} \frac{2\arccos(y)}{\pi(1-y)} &\approx 0.878567 \end{aligned}$$

We can conclude that our randomized algorithm is an 0.87 approximation algorithm to the MAXCUT problem.

5.1.3 Shape Constraint Regression

The problem is to estimate a function f based on a finite set of noisy observations. We have $y = f(t)$ we observe (y_i, t_i)s where $y_i = f(t_i) + \epsilon$. Nonetheless we have some information about f:

- We know what class of functions f belongs to e.g. polynomial of degree r.
- We have information about its shape.

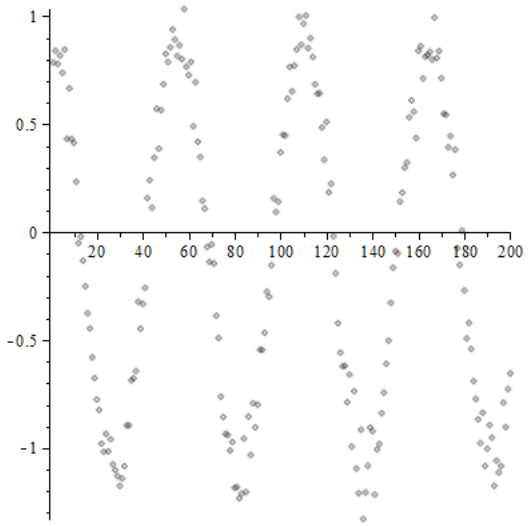

Figure 6: Scatter Plot of observed function values

If f is of class polynomial degree r we basically want to solve the following problem with some shape constraints:

$$\min \sum_i (y_i - f(t_i))^2 = \min_{p_0,\dots,p_r} \sum_i (y_i - p_0 - p_1 t_i - \cdots - p_r t_i^r) \tag{12}$$

Therefore our constrained problem can be stated as:

$$\begin{array}{ll} \min_{p_0,\dots,p_r} & \sum_i (y_i - p_0 - p_1 t_i - \cdots - p_r t_i^r) \\ \text{s.t.} & f(t) \geq 0 \quad \forall t \\ & f(t) \uparrow \\ & f(t) \cup \end{array} \tag{13}$$

The problem is that for example the non-negativity constraint makes our problem somewhat messy, if we insist on expressing this constraint in the form of $g_i(p) \geq 0$ for some functions g_i. Even with $r = 2$ $p_0 + p_1 t + p_2 t^2$ implies $p_1^2 - 4p_0p_2 \leq 0$.
However non-negative polynomials of degree r are a convex cone.

- CONE: $\alpha \geq 0$, $\alpha\times$ non-negative polynomial = non-negative polynomial
- CONVEX: non-negative polynomial + non-negative polynomial = non-negative polynomial

Let $\mathcal{P}_{2r}$ denote the convex cone of non-negative polynomials of degree $2r$ (obviously the degree has to be even). We can say the following:

$$\begin{aligned} p_0 + p_1 t + \cdots + p_r t^r \in \mathcal{P}_{2r} \quad \Leftrightarrow \quad & \exists Y \succcurlyeq 0,\ Y \in \mathbb{R}^{r+1\times r+1} : \\ p_0 &= y_{00} \\ p_1 &= y_{01} + y_{10} \\ p_2 &= y_{02} + y_{11} + y_{20} \\ &\vdots \\ p_{2r} &= y_{rr} \end{aligned}$$

where y_{ij}s are entries of

$$Y = \begin{pmatrix} y_{00} & y_{01} & \cdots & y_{0r} \\ \vdots & & & \\ y_{r0} & y_{r1} & \cdots & y_{rr} \end{pmatrix}$$

With this in mind our optimization problem becomes the following:

$$\begin{array}{ll} \min & \sum_i (y_i - p_0 - p_1 t_i - \cdots - p_r t_i^r)^2 \\ \text{s.t.} & y_{00} - p_0 = 0 \\ & y_{01} - y_{10} - p_1 = 0 \\ & \vdots \\ & y_{rr} - p_{2r} = 0 \\ & Y \succcurlyeq 0 \end{array} \tag{14}$$

where the y_i in the objective function are data readings. The quadratic terms in the objective can be replaced by linear terms, at the expense of adding second order cone constraints.

5.1.4 Single Facility Plant Location Problem (aka Fermat-Weber Problem)

Given a set of points find a point such that the sum of distances is minimal i.e. determine the location of $\mathbf{P}$ such that the sum of the distances from $\mathbf{P}$ to $\mathbf{x}$ is minimal.

Our problem translates to the following:

$$\min \quad w_1 \underbrace{\|\mathbf{p} - \mathbf{x}_1\|}_{z_1} + w_2 \underbrace{\|\mathbf{p} - \mathbf{x}_2\|}_{z_2} + \cdots + w_n \underbrace{\|\mathbf{p} - \mathbf{x}_n\|}_{z_n} \tag{15}$$

Figure 7: The Fermat-Weber Problem

Our optimization problem:

$$\begin{array}{ll} \min & w_1 z_1 + w_2 z_2 + \cdots + w_n z_n \\ \text{s.t} & z_i \geq \|\mathbf{p} - \mathbf{x}_i\| \end{array} \tag{16}$$

It turns out:

$$\begin{pmatrix} z_i \\ \mathbf{p} - \mathbf{x}_i \end{pmatrix} \in \mathcal{Q} \tag{17}$$

where $\mathcal{Q}$ is the Second Order Cone.
A mechanical analogue to the problem is illustrated in Figure 8. The points $\mathbf{x}$

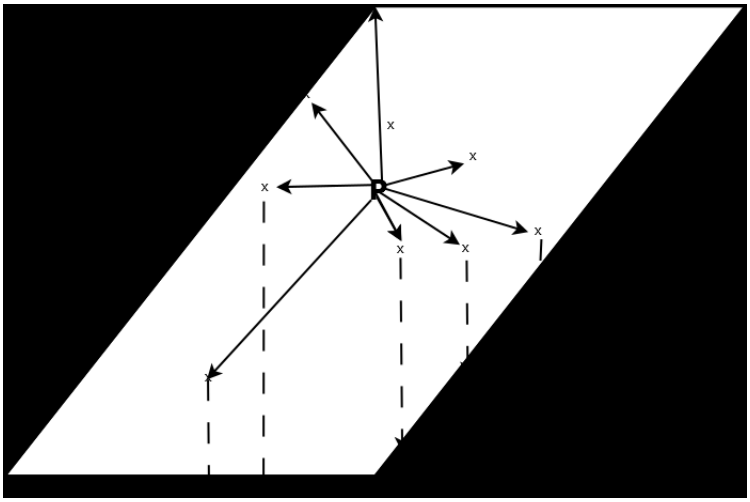

Figure 8: The Fermat-Weber Problem Mechanical Analogue

are holes in a table we tie a bunch of lines together and hang weights on their loose end the question is where will **P** be on the table?

References

[1] Alizadeh, F., Goldfarb, D., *Second-Order Cone Programming*, Mathematical Programming manuscript, 2010.