INTERIOR POINT METHODS FOR LINEAR PROGRAMMING

Khachiyan, L.G. (1979) A Polynomial Algorithm in Linear Programming, *Soviet Mathematics Doklady*, **20**, 191-194.

Karmarkar, N. (1984) A New Polynomial-Time Algorithm for Linear Programming, *Combinatorica*, **4**, 373-395.

Barnes, E. R. (1986) A variation on Karmarkar's Algorithm for Solving Linear Programming Problems, *Mathematical Programming* **36**, 174-182.

Vanderbei, R. J., M. S. Meketon, and B. A. Freedman, (1986) A modification of Karmarkar's Linear Programming Algorithm, *Algorithmica*, **1**, 395-407.

Megiddo, N., (1989) Pathways to the Optimal Set in Linear Programming, in *Progress in Mathematical Programming*, N. Meggido (ed.), Springer-Verlag, NY, 131-158.

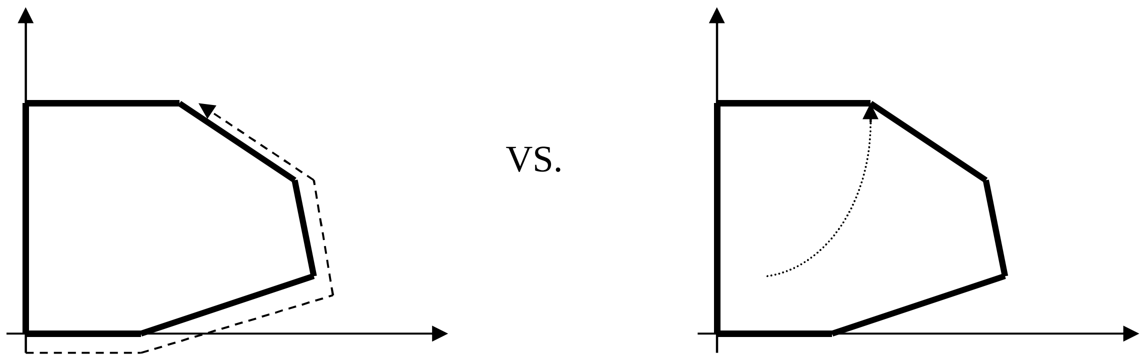
Monteiro, R. D. C., and I. Adler, (1989) Interior Path Following Primal-Dual Algorithms, Part I: Linear Programming, *Mathematical Programming*, **44**, 43-66.

Ye, Y. (1991) An $O(n^3L)$ Potential Reduction Algorithm for Linear Programming, *Mathematical Programming*, **50**, 239-258.

Lustig, I., R. E. Marsten, and D. Shanno, (1994) Interior Point Methods: Computational State of the Art, *ORSA Journal on Computing*, **6**, 1-14.

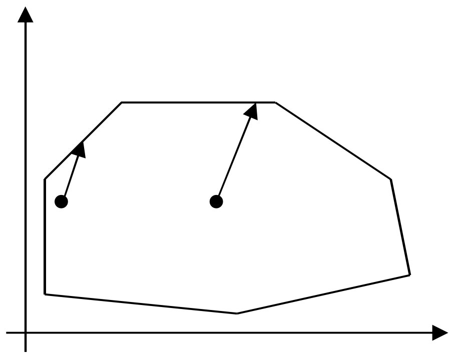
etc. etc. etc.

The basic difference:



TWO FUNDAMENTAL INSIGHTS:

1. If we are currently close to the “center” of the polytope representing the feasible region, a logical approach would be to move in the direction of maximum improvement, i.e., in the direction of steepest descent (assuming minimization).



2. The feasible region can be transformed so that the current solution is close to the center of the polytope *without changing the problem in any essential way*.

At each iteration we would like to move along a direction so that (1) there is improvement, (2) feasibility is maintained, and (3) there is "centrality" (whatever that means...).

Very broadly speaking, Interior Point Methods can be classified into four classes - however, there are algorithms that borrow ideas from several classes.

Affine-Scaling: These are the simplest methods and use a much simpler "affine scaling" for the transformation in Insight 2 on the previous page. They work surprisingly well, but tend to be sensitive to degeneracy and more importantly, lack proof of polynomial time convergence.

Projective Scaling: Karmarkar's original method falls into this category and the transformation in Insight 2 is done via a "projective transform." The procedure is rather elaborate and needs a lot of "extra" work, but this was the approach that provided the impetus for other interior point methods and has a proven polynomial time complexity.

Potential Reduction: In this approach, progress toward the optimum is measured by improvement in the value of a nonlinear "potential" function rather than the objective itself. The complexity is polynomial time. Karmarkar's method was also a potential reduction algorithm.

Path Following: These are currently considered the best methods and are based on a logarithmic "barrier" function that keeps one away from the boundary. The solutions follow a "central path" (hence the name) and the methods have polynomial time complexity as well.

THE (PRIMAL) AFFINE SCALING ALGORITHM

The simplest Interior Point approach is the Affine Scaling method. The method was originally proposed by Dikin (*Soviet Math. Doklady*, 1967) but his work went largely unnoticed until after Karmarkar's method was published, when it was independently proposed by Barnes (*Math. Prog.*, 1986) and Vanderbei et al. (*Algorithmica*, 1986). It is an efficient yet relatively simple variant of Karmarkar's general approach, i.e., a sequence of "centering" transforms followed by movement in a promising direction.

The main advantage of this method is that it uses a much simpler "affine" transform (rather than Karmarkar's projective transform) in order to center the current point; in fact, the transform this is merely a simple rescaling. It also does not have any artificial format requirements (unlike Karmarkar's method which requires initial reformulation in a very specific format, including the requirement that the optimum value for the reformulated problem is 0!) .

On the other hand, the disadvantages are that (1) nobody has been able to show that it converges in polynomial time (in fact, there seems to be evidence that it does **not**), and (2) it tends to be sensitive to degeneracy.

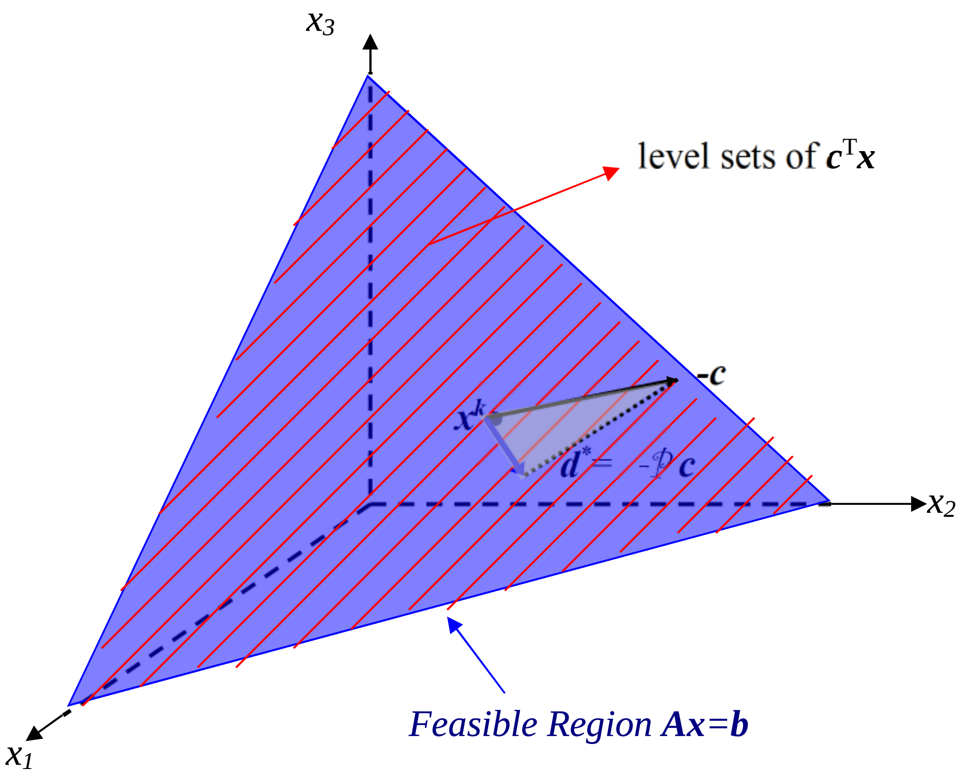
The problem solved is the following: Minimize $\{\boldsymbol{c}^T\boldsymbol{x}$, st $\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}$, $\boldsymbol{x}\geq\boldsymbol{0}\}$.

To see how the method works, consider an iteration where we are currently at the interior point $\boldsymbol{x}^k$ (i.e., >0) How do we move from this point so that (1) we remain feasible, and (2) we also improve the objective?

<u>NOTE</u>: This is a basic (direction finding) procedure that will be used in several Interior Point Algorithms so it is important to follow closely!!

Note that moving a distance s along some direction $\boldsymbol{d}$ changes the objective from $\boldsymbol{cx}^k$ to $\boldsymbol{c}(\boldsymbol{x}^k + s\boldsymbol{d})$. The direction $\boldsymbol{d}^*$ that maximizes the decrease in the objective [= $\boldsymbol{cx}^k - \boldsymbol{c}(\boldsymbol{x}^k + s\boldsymbol{d})$] is given by the negative of the gradient of $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^k$ (the direction of "steepest descent"), and thus $\boldsymbol{d}^* = -\boldsymbol{c}$.

HOWEVER...$\boldsymbol{d}^*$ may not be feasible! We may leave the feasible region even for an infinitesimally small move along the direction $\boldsymbol{d}^*$ (in other words, $\boldsymbol{A}(\boldsymbol{x}^k + s\boldsymbol{d}^*)=\boldsymbol{b}$ is violated for arbitrarily small values of the step size s. This is shown below:



x_3
level sets of $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$
$-\boldsymbol{c}$
x^k
$\boldsymbol{d}^* = -\mathcal{P}\boldsymbol{c}$
x_2
x_1
Feasible Region $\boldsymbol{Ax}=\boldsymbol{b}$

Given any direction $\boldsymbol{d}$, to *maintain feasibility* we want

$\boldsymbol{A}(\boldsymbol{x}^k + s\boldsymbol{d}) = \boldsymbol{b}$ for some $s>0$ $\Rightarrow$ $\boldsymbol{A}\boldsymbol{x}^k + s\boldsymbol{A}\boldsymbol{d} = \boldsymbol{b}$

$\Rightarrow$ $\boldsymbol{b} + s\boldsymbol{A}\boldsymbol{d} = \boldsymbol{b}$ $\Rightarrow$ $s\boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}$ $\Rightarrow$ $\boxed{\boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}}$

Thus, to maximize the decrease, we need to solve

$$\text{Max } \boldsymbol{c}\boldsymbol{x}^k - \boldsymbol{c}(\boldsymbol{x}^k + s\boldsymbol{d}), \text{ i.e., } \quad \text{Max } -\boldsymbol{c}\boldsymbol{d}$$

$$\text{st } \boldsymbol{A}\boldsymbol{d} = \boldsymbol{0}$$

for the optimal direction $\boldsymbol{d}^*$.

It is easily shown that $\boldsymbol{d}^* = -[\boldsymbol{c}^T - \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{c}^T] = -[\boldsymbol{I}-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}]\,\boldsymbol{c}^T$

Denoting $\mathcal{P} = [\boldsymbol{I}-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}]$, this implies that $\boldsymbol{d}^* = -\mathcal{P}\,\boldsymbol{c}$

The matrix $\mathcal{P}$ is called a *projection matrix*. Geometrically, this matrix projects the vector $\boldsymbol{c}$ (the direction of steepest descent) on to the space $\{\boldsymbol{d} \mid \boldsymbol{A}\boldsymbol{d}=\boldsymbol{0}\}$ - also called the null space of $\boldsymbol{A}$. This is illustrated in the picture on the previous page.

In practice, we don't explicitly compute $[\boldsymbol{I}-\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}]$. Instead, we first solve the system of linear equations $(\boldsymbol{A}\boldsymbol{A}^T)\boldsymbol{w} = \boldsymbol{A}\boldsymbol{c}^T$ for the vector $\boldsymbol{w}$ (which is thus equal to $(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}\boldsymbol{c}^T$)and then obtain $\boldsymbol{d}^*$ as $\boldsymbol{d}^* = -[\boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w}]$.

NOTE: For a maximization problem, $\boldsymbol{d}^* = [\boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w}]$.

While this direction $\boldsymbol{d}^*$ is good in the sense of maximizing the *rate* of improvement, the actual *amount* of improvement will depend on how far we can go in this direction. If $\boldsymbol{x}^k$ is close to a "wall" and we can't go to too far, the overall decrease in $\boldsymbol{cx}$ will be small. Thus we need to be at the "center" (whatever that means...).

The affine scaling method achieves this by (1) re-scaling the variables so that the current point $\boldsymbol{x}^k$ is "moved" to some "central" point $\boldsymbol{y}^k$ which is far from the "walls," (2) then moving (presumably a good distance) along the projected steepest descent direction for this re-scaled problem to $\boldsymbol{y}^{k+1}$, and (3) translating $\boldsymbol{y}^{k+1}$ back to $\boldsymbol{x}^{k+1}$ in the original scale.

QUESTION: How is this scaling done?

ANSWER: By an "affine" scaling!

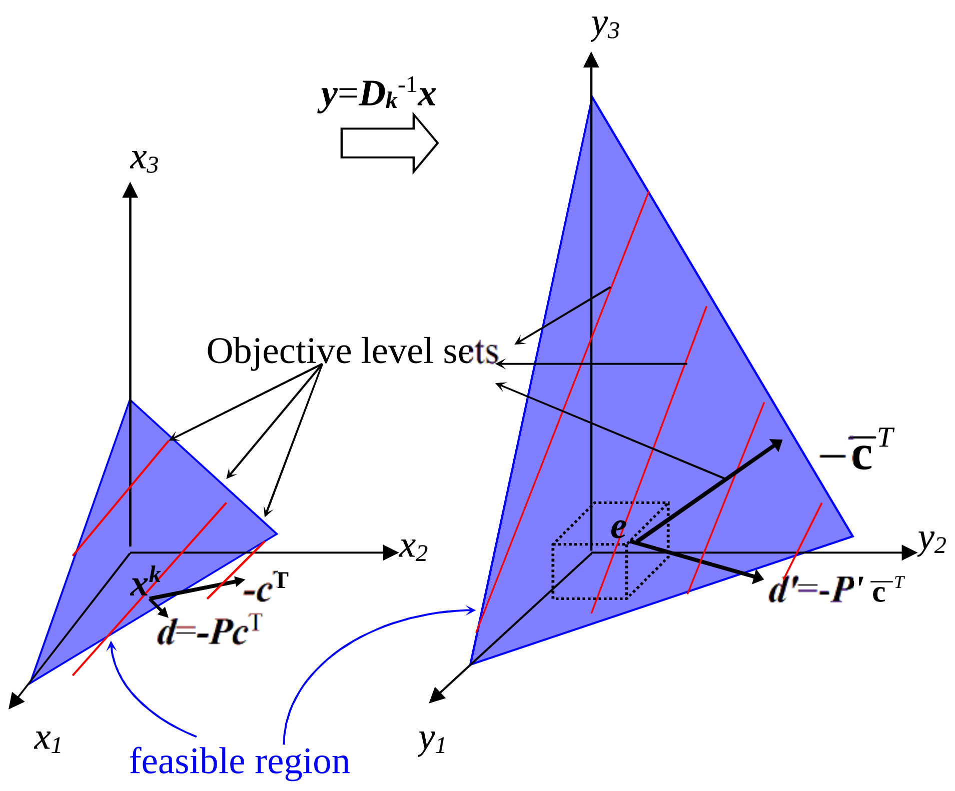
Consider the following transform that is defined using $\boldsymbol{x}^k$ as the reference point:

$$y_j = (x_j/x_j^k) \text{ for } j=1,2,...,n$$

Using matrix notation, if we define $\boldsymbol{D_k}$ = Diag($\boldsymbol{x}^k$), the transform may be written as $\boldsymbol{y}=\boldsymbol{D_k}^{-1}\boldsymbol{x}$. Note that $\boldsymbol{D_k}$ is a (constant) matrix that rescales any point $\boldsymbol{x} = (x_1, x_2,..., x_n)$ into the point $\boldsymbol{y}=(y_1, y_2,...,y_n)$. In particular, the current point $\boldsymbol{x}^k$ is transformed to the point $\boldsymbol{y}^k=\boldsymbol{e}$, where $\boldsymbol{e} =[1\ \ 1\ ...\ 1]^T$. Note also that the transform is trivial to undo via $x_j=y_jx_j^k$, i.e., $\boldsymbol{x=D_ky}$.

What does this affine transformation accomplish in terms of "centering?"

Consider the picture below:



The picture shows how the feasible region (and the current point) are transformed. Note that (since $\boldsymbol{x=D_ky}$) in terms of the transformed variables, the original problem, namely {Minimize $\boldsymbol{cx}$ | $\boldsymbol{Ax}=\boldsymbol{b}$, $\boldsymbol{x}\geq\boldsymbol{0}$} becomes: Minimize {$\boldsymbol{cD_ky}$ | $\boldsymbol{AD_ky} = \boldsymbol{b}$, $\boldsymbol{y}\geq\boldsymbol{0}$). If we define $\bar{c}=\boldsymbol{cD_k}$ and $\boldsymbol{P_k}=\boldsymbol{AD_k}$ this reduces the problem to minimizing $\bar{c}^T\boldsymbol{y}$, subject to $\boldsymbol{P_ky}=\boldsymbol{b}$, $\boldsymbol{y}\geq\boldsymbol{0}$. Also note that the point $\boldsymbol{x^k}$ ***moves to*** $\boldsymbol{y^k=e}$.

QUESTION 1**:** What does this accomplish?

ANSWER: If we had used projection matrix $\boldsymbol{P = I - A}^{\mathrm{T}}(\boldsymbol{AA}^{\mathrm{T}})^{-1}\boldsymbol{A}$ to project $-\boldsymbol{c}^{\mathrm{T}}$ in the (original) $\boldsymbol{x}$-space we might not have been able to move too far from $\boldsymbol{x}^{\boldsymbol{k}}$ along the direction $\boldsymbol{d}=-\boldsymbol{Pc}^{\mathrm{T}}$.

However, when we use $\boldsymbol{P'=I - P_k}^{\mathrm{T}}(\boldsymbol{P_kP_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}$ to project $-\bar{\boldsymbol{c}}^{\mathrm{T}}$ in the (transformed) $\boldsymbol{y}$-space we will in general be able to move a reasonable distance from $\boldsymbol{y}^{\boldsymbol{k}}=\boldsymbol{e}$ along the direction $\boldsymbol{d'}=-\boldsymbol{P'}\bar{\boldsymbol{c}}^{\mathrm{T}}$ as shown. This is because of the "re-centering" accomplished by the affine transform.

QUESTION 2: Having decided to move along $-\boldsymbol{P'}\bar{\boldsymbol{c}}^{\mathbf{T}}$ in the $\boldsymbol{y}$-space, how far to go?

ANSWER: Assume that we move a distance $1/v$ (>0). Then the new point $\boldsymbol{y}^{\boldsymbol{k+1}}$ in the transformed space is $\boldsymbol{y}^{\boldsymbol{k}}+(1/v)\boldsymbol{d'}$. But $\boldsymbol{y}^{\boldsymbol{k}}$ is equal to $\boldsymbol{e}$, so $\boldsymbol{y}^{\boldsymbol{k+1}} = \boldsymbol{e}+(1/v)\boldsymbol{d'}$. In the $\boldsymbol{x}$-space this point is given by $\boldsymbol{x}^{\boldsymbol{k+1}}= \boldsymbol{D_k y}^{\boldsymbol{k+1}} = \boldsymbol{D_k}(\boldsymbol{e}+(1/v)\boldsymbol{d'})$

i.e.,

$$\boldsymbol{x}^{\boldsymbol{k+1}} = \boldsymbol{D_k e} + (1/v)\boldsymbol{D_k d'} = \boldsymbol{x}^{\boldsymbol{k}} + (1/v)\boldsymbol{D_k d'}.$$

For this point to be feasible we want each element of $\boldsymbol{x}^{\boldsymbol{k+1}}$ to be nonnegative, i.e.,

$$\begin{bmatrix} x_1^{k+1} \\ x_2^{k+1} \\ : \\ : \\ x_n^{k+1} \end{bmatrix} = \begin{bmatrix} x_1^{k} \\ x_2^{k} \\ : \\ : \\ x_n^{k} \end{bmatrix} + \left(\frac{1}{v}\right) \begin{bmatrix} d'_1 x_1^{k} \\ d'_2 x_2^{k} \\ : \\ : \\ d'_n x_n^{k} \end{bmatrix} \geq \begin{bmatrix} 0 \\ 0 \\ : \\ : \\ 0 \end{bmatrix}$$

Since each element of $\boldsymbol{x^k}$ is strictly positive, we only have to worry about the negative elements of $\boldsymbol{d'}$. For all such j (corresponding to which $d_j' < 0$), we require that

$$x_j^k + \left(\frac{1}{v}\right) d_j' x_j^k \geq 0, \ \ i.e., \ \ 1 + \left(\frac{1}{v}\right) d_j' \geq 0, \ \ i.e., \ \ v \geq Abs[d_j'].$$

Hence, if we choose

$$v = \underset{j \ni d'_j < 0}{\text{Maximum}} \, |d_j'|$$

and move a distance $1/v$ along the direction $\boldsymbol{d'}$, we will remain feasible. Note that this same step size could have also been derived by requiring that each element of $\boldsymbol{y^{k+1}}$ be nonnegative (i.e., $1+(1/v)d_j' \geq 0$ for all j)...

In practice, since we always want each element of $\boldsymbol{x^{k+1}}$ at any iteration to be ***strictly positive*** (so that we can make the transformation $\boldsymbol{y}=\boldsymbol{D_k}^{-1}\boldsymbol{x}$), we move a distance which is which is a little less than $(1/v)$, say $\alpha(1/v)$ where $0<\alpha<1$.

The algorithm may thus be summarized as follows:

STEP 0: Set the iteration counter k=1, and start with some feasible point $\boldsymbol{x^1}$ with strictly positive elements. Define a value for α (a value of 0.97 is suggested), and a small tolerance ε.

STEP 1: Define the diagonal matrix $\boldsymbol{D_k}$ =Diag($\boldsymbol{x_k}$) and calculate $\boldsymbol{P_k} = \boldsymbol{AD_k}$ and $\bar{\boldsymbol{c}} = \boldsymbol{cD_k}$. Solve the system $(\boldsymbol{P_k P_k}^T)\boldsymbol{w^k} = \boldsymbol{P_k}\bar{\mathbf{c}}^T$ for the vector $\boldsymbol{w^k}$**.**

STEP 2: STOP if some suitable stopping criterion is met (more on this later...)

STEP 3: Compute $\boldsymbol{d'} = -[\bar{\mathbf{c}}^T - \boldsymbol{P_k}^T\boldsymbol{w^k}] = -\boldsymbol{D_k}[\boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w^k}]$ (if maximizing $\boldsymbol{d'} = [\bar{\mathbf{c}}^T - \boldsymbol{P_k}^T\boldsymbol{w^k}] = \boldsymbol{D_k}[\boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w^k}]$). If $\boldsymbol{d'}>0$, the problem is unbounded; so STOP. Otherwise, identify the negative component of $\boldsymbol{d'}$ with the largest absolute value and set v equal to this absolute value. Set k=k+1 and return to Step 1 after computing the new point as

$\boldsymbol{x^{k+1}} = \boldsymbol{D_k}[\boldsymbol{e} + \alpha(1/v)\boldsymbol{d'}] = \boldsymbol{x^k} + (\alpha/v)\boldsymbol{D_k d'}$

NOTE: It is possible to express the entire algorithm in terms of the original $\boldsymbol{x}$-space. To see this note that $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + (\alpha/v)\boldsymbol{D_k d'} = \boldsymbol{x^k} - (\alpha/v)\boldsymbol{D_k}[\bar{\mathbf{c}}^T - \boldsymbol{P_k}^T\boldsymbol{w^k}]$

$$
\begin{aligned}
&= \boldsymbol{x^k} - (\alpha/v)\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^T(\boldsymbol{P_k P_k}^T)^{-1}\boldsymbol{P_k}]\,\bar{\mathbf{c}}^T \\
&= \boldsymbol{x^k} - (\alpha/v)\boldsymbol{D_k}[\boldsymbol{I} - (\boldsymbol{AD_k})^T[\boldsymbol{AD_k}(\boldsymbol{AD_k})^T]^{-1}\boldsymbol{AD_k}]\,(\boldsymbol{cD_k})^T \\
&= \boldsymbol{x^k} - (\alpha/v)\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{D_k A}^T[\boldsymbol{AD_k}^2\boldsymbol{A}^T]^{-1}\boldsymbol{AD_k}]\,\boldsymbol{D_k c}^T \\
&= \boldsymbol{x^k} - (\alpha/v)\boldsymbol{D_k Q_k D_k c}^T \\
&= \boldsymbol{x^k} + (\alpha/v)\boldsymbol{\Delta x}
\end{aligned}
$$

where $\boldsymbol{Q_k} = [\boldsymbol{I} - \boldsymbol{D_k A}^T[\boldsymbol{AD_k}^2\boldsymbol{A}^T]^{-1}\boldsymbol{AD_k}]$ and $\boldsymbol{\Delta x} = -\boldsymbol{D_k Q_k D_k c}^T$ is called the ***primal affine-scaling direction***

STOPPING CRITERION (Step 2)

Consider the dual of the LP that we solve, namely Maximize $\{\boldsymbol{b}^{\mathrm{T}}\boldsymbol{w},\ \text{st } \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} \le \boldsymbol{c}^{\mathrm{T}}\}$.

A very elegant result is that under the assumption of nondegeneracy it may be shown that when in Step 2 we find

$$\boldsymbol{w}^{k} = (\boldsymbol{P}_k\boldsymbol{P}_k^{\mathrm{T}})^{-1}\boldsymbol{P}_k\,\overline{\mathbf{c}}^{T} = [\boldsymbol{A}\boldsymbol{D}_k(\boldsymbol{A}\boldsymbol{D}_k^{\mathrm{T}})]^{-1}\boldsymbol{A}\boldsymbol{D}_k(\boldsymbol{c}\boldsymbol{D}_k)^{\mathrm{T}} = [\boldsymbol{A}\boldsymbol{D}_k^{2}\boldsymbol{A}^{\mathrm{T}}]^{-1}\boldsymbol{A}\boldsymbol{D}_k^{2}\boldsymbol{c}^{\mathrm{T}},$$

these values of $\boldsymbol{w}^k$ converge to the optimum solution (say $\boldsymbol{w}^*$) to the dual of the LP that we just defined above!

Now, recall that at the optimum, the primal and dual optimum values must be identical, i.e., if the vectors $\boldsymbol{x}^* \in \mathrm{R}^n$ and $\boldsymbol{w}^* \in \mathrm{R}^m$ define the optimum primal and dual vectors we must have $\boldsymbol{c}\boldsymbol{x}^* = \boldsymbol{b}^{\mathrm{T}}\boldsymbol{w}^*$ at the optimum. Thus, in Step 2 a stopping criterion (that is commonly used in practice) is to stop if (1) the vector $\boldsymbol{w}^k$ is dual-feasible, and (2) the duality gap is zero (or "sufficiently" small), i.e., if

(1) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k \le \boldsymbol{c}^{\mathrm{T}}$ (this is equivalent to ensuring that $\boldsymbol{s}^k = \boldsymbol{c}^{\mathrm{T}} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k \ge \boldsymbol{0}$),

(2) $\boldsymbol{c}\boldsymbol{x}^k - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{w}^k < \varepsilon$. (since $\boldsymbol{A}\boldsymbol{x}^k = \boldsymbol{b}$, i.e., $\boldsymbol{b}^{\mathrm{T}} = (\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}$, this is equivalent to ensuring

that $\boldsymbol{c}\boldsymbol{x}^k - (\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k = (\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{c}^{\mathrm{T}} - (\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k = (\boldsymbol{x}^k)^{\mathrm{T}}[\boldsymbol{c}^{\mathrm{T}} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k] = (\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{s}^k < \varepsilon$).

Note that $\boldsymbol{s}^k$ is a vector of slack variables for the dual problem. In summary:

At Step 2 we use the current values of $\boldsymbol{x}^k$ and $\boldsymbol{w}^k$ to compute $\boldsymbol{s}^k = \boldsymbol{c}^{\mathrm{T}} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w}^k$ and stop if (1) $\boldsymbol{s}^k \ge \boldsymbol{0}$, and (2) $(\boldsymbol{x}^k)^{\mathrm{T}}\boldsymbol{s}^k < \varepsilon$. In this case $\boldsymbol{x}^k$ is primal-optimal and $\boldsymbol{w}^k$ is dual-optimal.

It is also interesting to note that the vector $\boldsymbol{s}^k$ is nothing but a vector of reduced costs and (1) and (2) represent the usual optimality conditions!

It is also possible to use other stopping criteria.

For example, if there is very little progress being made, i.e., $||\boldsymbol{x}^{k+1}-\boldsymbol{x}^{k}||<\varepsilon$, or if the relative improvement in the objective is minimal, i.e., $\{\boldsymbol{cx}^{k}-\boldsymbol{cx}^{k+1}\}\div|\boldsymbol{cx}^{k}|<\varepsilon$.

It is also worth noting that the $\{\boldsymbol{w}^{k}\}$ may not converge to the dual optimum if the problem is degenerate, and in such cases a simpler stopping criterion such as the ones above might be called for.

The method described is called the ***Primal*** Affine Scaling Method. There is also a ***Dual*** Affine Scaling Algorithm. The general principles for this are similar except that the Dual Algorithm works on the Dual problem namely Maximize $\{\boldsymbol{b}^{T}\boldsymbol{w}$, st $\boldsymbol{A}^{T}\boldsymbol{w}+\boldsymbol{s}=\boldsymbol{c}^{T}$, $\boldsymbol{s}\geq\boldsymbol{0}$, $\boldsymbol{w}$ unrestricted$\}$. The iteration procedure is analogous to the Primal Algorithm (initial interior point, rescaling, moving etc.), except that the work is done on the dual problem - like the Dual Simplex method it successively increases the dual objective while maintaining dual feasibility.

Both of these methods are very simple, but in practice, they have been shown to work very well and are competitive with the best Interior Point methods. Both methods are globally convergent; unfortunately, there is no proof of polynomial complexity for either.

Finally, there is also a ***Primal-Dual*** Affine Scaling Algorithm that has polynomial complexity of order $O(nL^{2})$, but in general, does not seem to perform as well as the Primal or Dual Affine Scaling methods.

EXAMPLE:

Maximize x_1+2x_2, st $x_1+x_2\le 2$, $-x_1+x_2\le 1$, x_1, $x_2\ge 0$,

In the required form, this is

Minimize $-x_1 -2x_2$

st $x_1+x_2+x_3 \quad =2, \quad -x_1+x_2+ \quad x_4 \ =1, \quad x_1, x_2, x_3, x_4 \ge 0.$

i.e., Min $\boldsymbol{cx} = [-1 \ \ -2 \ \ 0 \ \ 0]\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}$

st $\boldsymbol{Ax=b}$, i.e., $\begin{bmatrix} 1 & 1 & 1 & 0 \\ -1 & 1 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$, $\quad \boldsymbol{x\ge 0}$

Note that the dual is given by {Max $2w_1+w_2$, st $w_1-w_2\le -1$, $w_1+w_2\le -2$, $w_1\le 0$, $w_2\le 0$}.

Say we start at $\boldsymbol{x^1}=[0.1 \quad 0.1 \quad 1.8 \quad 1.0]^T$ with $Z=-0.3$.

STEP 1: Define $\boldsymbol{D_1}=\begin{bmatrix} 0.1 & & & \\ & 0.1 & & \\ & & 1.8 & \\ & & & 1 \end{bmatrix}$, so that

$\boldsymbol{P_1}=\boldsymbol{AD_1}=\begin{bmatrix} 0.1 & 0.1 & 1.8 & 0 \\ -0.1 & 0.1 & 0 & 1 \end{bmatrix}$ and $\bar{\boldsymbol{c}} = \boldsymbol{cD_1} = [-0.1 \ \ -0.2 \ \ 0 \ \ 0]$

Solving the system $(\boldsymbol{P_1P_1}^T)\boldsymbol{w^1} = \boldsymbol{P_1}\bar{\mathbf{c}}^T$ for the vector $\boldsymbol{w^1}$ yields

$$\begin{bmatrix} 0.1 & 0.1 & 1.8 & 0 \\ -0.1 & 0.1 & 0 & 1 \end{bmatrix}\begin{bmatrix} 0.1 & -0.1 \\ 0.1 & 0.1 \\ 1.8 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.1 & 1.8 & 0 \\ -0.1 & 0.1 & 0 & 1 \end{bmatrix}\begin{bmatrix} -0.1 \\ -0.2 \\ 0 \\ 0 \end{bmatrix},$$

i.e., $\begin{bmatrix} 3.26 & 0 \\ 0 & 1.02 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} -0.03 \\ -0.01 \end{bmatrix}$ so that $\boldsymbol{w^1}=\begin{bmatrix} -0.0092 \\ -0.0098 \end{bmatrix}$.

STEP 2 : Checking the two stopping criteria

(i) $\boldsymbol{s^1} = \boldsymbol{c}^T - \boldsymbol{A}^t\boldsymbol{w^1} = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -0.0092 \\ -0.0098 \end{bmatrix} = \begin{bmatrix} -1.0006 \\ -1.9810 \\ 0.0092 \\ 0.0098 \end{bmatrix}$ --- NOT $\geq \boldsymbol{0}$

(ii) $(\boldsymbol{x^1})^T\boldsymbol{s^1} = [0.1 \quad 0.1 \quad 1.8 \quad 1] \begin{bmatrix} -1.0006 \\ -1.9810 \\ 0.0092 \\ 0.0098 \end{bmatrix} = -0.2718$

STEP 3: The search direction is given by

$$\boldsymbol{d'} = -[\bar{\boldsymbol{c}}^T - \boldsymbol{P_1}^T\boldsymbol{w^1}] = -\begin{bmatrix} -0.1 \\ -0.2 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.1 & -0.1 \\ 0.1 & 0.1 \\ 1.8 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} -0.0092 \\ -0.0098 \end{bmatrix} = \begin{bmatrix} 0.10006 \\ 0.19810 \\ -0.01656 \\ -0.00980 \end{bmatrix}$$

From the above vector, we compute v=0.0166 so that $\boldsymbol{x^2} = \boldsymbol{x^1} + (\alpha/v)\boldsymbol{D_1 d'}$

$$= \begin{bmatrix} 0.1 \\ 0.1 \\ 1.8 \\ 1 \end{bmatrix} + (0.99/0.0166) \begin{bmatrix} 0.1 & & & \\ & 0.1 & & \\ & & 1.8 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} 0.10006 \\ 0.19810 \\ -0.01656 \\ -0.00980 \end{bmatrix} = \begin{bmatrix} 0.698 \\ 1.284 \\ 0.018 \\ 0.414 \end{bmatrix}$$

ITERATION 2

STEP 1: Define $\boldsymbol{D_2} = \begin{bmatrix} 0.698 & & & \\ & 1.284 & & \\ & & 0.018 & \\ & & & 0.414 \end{bmatrix}$, so that

$\boldsymbol{P_2} = \boldsymbol{AD_2} = \begin{bmatrix} 0.698 & 1.284 & 0.018 & 0 \\ -0.698 & 1.284 & 0 & 0.414 \end{bmatrix}$ and $\bar{\boldsymbol{c}} = \boldsymbol{cD_2} = [-0.698 \quad -2.568 \quad 0 \quad 0]$

Solving the system $(\boldsymbol{P_2 P_2}^{\mathrm{T}})\boldsymbol{w}^2 = \boldsymbol{P_2}\,\bar{\mathbf{c}}^T$ for the vector $\boldsymbol{w}^2$ yields

$$\begin{bmatrix} 0.698 & 1.284 & 0.018 & 0 \\ -0.698 & 1.284 & 0 & 0.414 \end{bmatrix}\begin{bmatrix} 0.698 & -0.698 \\ 1.284 & 1.284 \\ 0.018 & 0 \\ 0 & 0.414 \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \end{bmatrix} =$$

$$\begin{bmatrix} 0.698 & 1.284 & 0.018 & 0 \\ -0.698 & 1.284 & 0 & 0.414 \end{bmatrix}\begin{bmatrix} -0.698 \\ -2.568 \\ 0 \\ 0 \end{bmatrix},$$

which yields $\boldsymbol{w}^1 = \begin{bmatrix} -1.5275 \\ -0.4490 \end{bmatrix}$.

<u>STEP 2</u> : Checking the two stopping criteria

(iii) $\boldsymbol{s}^1 = \boldsymbol{c}^{\mathrm{T}} - \boldsymbol{A}^{\mathrm{t}}\boldsymbol{w}^1 = \begin{bmatrix} -1 \\ -2 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} 1 & -1 \\ 1 & 1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} -1.5275 \\ -0.4490 \end{bmatrix} = \begin{bmatrix} 0.0785 \\ -0.0235 \\ 1.5275 \\ 0.4490 \end{bmatrix}$ --- NOT $\geq \boldsymbol{0}$

(iv) $(\boldsymbol{x}^1)^{\mathrm{T}}\boldsymbol{s}^1 = [0.698 \quad 1.284 \quad 0.018 \quad 0.414]\begin{bmatrix} 0.0785 \\ -0.0235 \\ 1.5275 \\ 0.4490 \end{bmatrix} = 0.238$

<u>STEP 3</u>: The search direction is given by

$$\boldsymbol{d}' = -[\,\bar{\mathbf{c}}^T - \boldsymbol{P_1}^{\mathrm{T}}\boldsymbol{w}^1] = -\begin{bmatrix} -0.698 \\ -2.568 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0.698 & -0.698 \\ 1.284 & 1.284 \\ 0.018 & 0 \\ 0 & 0.414 \end{bmatrix}\begin{bmatrix} -1.5275 \\ -0.4490 \end{bmatrix} = \begin{bmatrix} 0.0548 \\ 0.0302 \\ -0.0275 \\ -0.1859 \end{bmatrix}$$

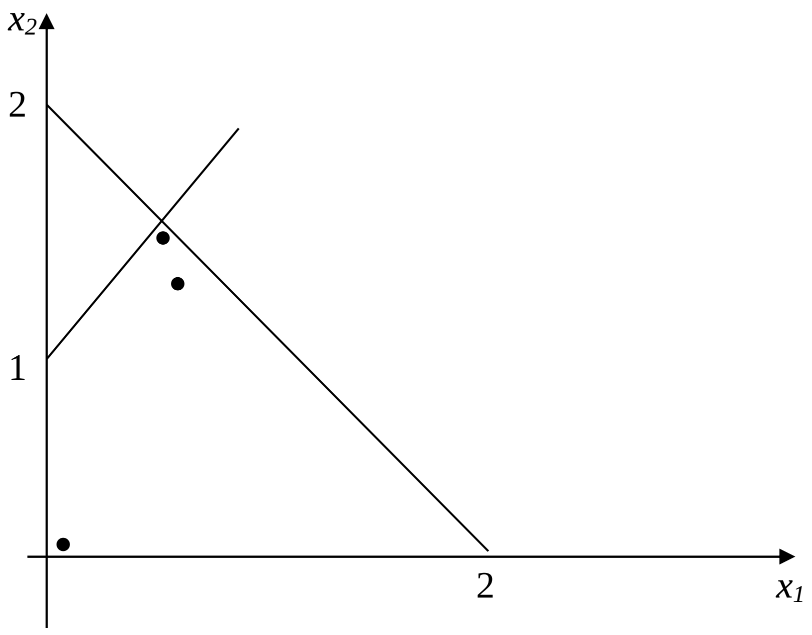
From the above vector, we compute v=0.1859 so that $\boldsymbol{x}^2 = \boldsymbol{x}^1 + (\alpha/v)\boldsymbol{D_1 d}'$

$$= \begin{bmatrix} 0.698 \\ 1.284 \\ 0.018 \\ 0.414 \end{bmatrix} + (0.99/0.1859)\begin{bmatrix} 0.698 & & & \\ & 1.284 & & \\ & & 0.018 & \\ & & & 0.414 \end{bmatrix}\begin{bmatrix} 0.0548 \\ 0.0302 \\ -0.0275 \\ -0.1859 \end{bmatrix} = \begin{bmatrix} 0.494 \\ 1.490 \\ 0.016 \\ 0.004 \end{bmatrix}$$

Note that we are approaching the optimum solution,

namely [0.5   1.5   0   0].

The graph below shows the progress made by the algorithm:



NOTE: This version is called the "long-step" affine scaling method. There are earlier "short-step" versions where the step size uses a different value for v such as $v = \|\boldsymbol{d'}\|$ or $v = \text{Max}_i |d_i'|$. In general, the long-step version is more popular.

<u>STARTING THE METHOD:</u> To find a strictly positive starting point the easiest way is to use a Big-M type approach by introducing an artificial variable x_{n+1} and solving:

$$\boxed{\text{Minimize } \{\boldsymbol{cx} + Mx_{n+1} \text{ st } \boldsymbol{Ax}+[\boldsymbol{b}-\boldsymbol{Ae}]x_{n+1}=\boldsymbol{b},\ \boldsymbol{x}\geq\boldsymbol{0},\ x_{n+1}\geq 0\}.}$$

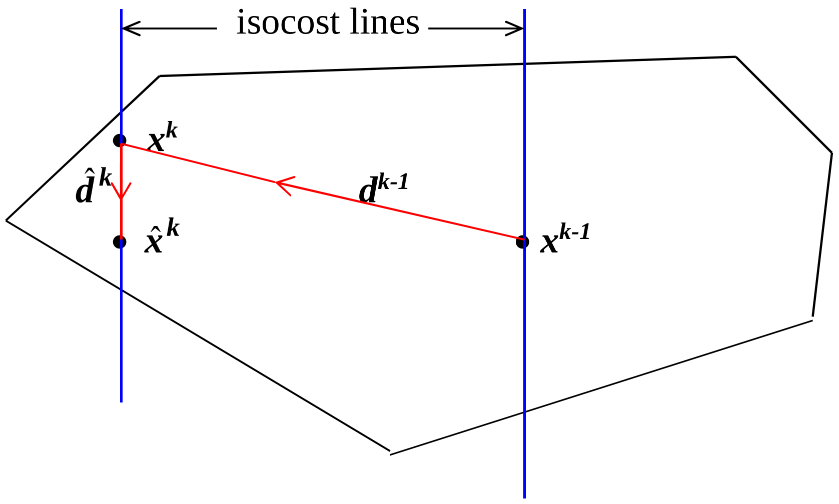
Note that the point $[\boldsymbol{x}, x_{n+1}] = [\boldsymbol{e}, 1]$ is positive and feasible for this and if M is large enough, then as long as the original problem has an optimal solution the value of x_{n+1} will go to zero at the optimum.

There is also a (more complex) Two-Phase approach to getting started...

PATH FOLLOWING ALGORITHMS

This is another major class of Interior Point methods. These methods resemble affine scaling methods, but use a different objective. Moreover, they tend to offset some of the computational problems associated with affine scaling and have better convergence properties. Motivating their development was the drawback with affine scaling that if one gets too close to the boundary (where some x_j=0), then during a move from $\boldsymbol{x}^{k-1}$ to $\boldsymbol{x}^{k}$ the method tends slow down. Of course, one could use a smaller value of the step size (α), but this just increases the no. of iterations.

Most implementations of the affine scaling method need a procedure to take point $\boldsymbol{x}^{k}$ and move it further inside the feasible region in order to avoid getting "trapped" by the boundary, while *simultaneously ensuring that the objective function does not increase*. For example, one approach is illustrated in the figure where we move from $\boldsymbol{x}^{k-1}$ to $\boldsymbol{x}^{k}$ along $\boldsymbol{d}^{k-1}$, and then in an additional step move along $\hat{\boldsymbol{d}}^{k-1}$ to the point $\hat{\boldsymbol{x}}^{k}$ in order to move further away from the boundary.



A more common way to deal with this "boundary" problem is to draw upon an approach that has been commonly used in *nonlinear* programming - namely the barrier function. The idea is to use a penalty for the objective to stay away from the boundary. The most common approach is the logarithmic barrier function.

Consider the usual primal problem, i.e., {Minimize $\boldsymbol{cx}$, st $\boldsymbol{Ax}=\boldsymbol{b}$, $\boldsymbol{x}\geq\boldsymbol{0}$}.

For this problem, at the boundary of the polyhedron, at least one $x_j=0$. To keep away from the boundary we modify this problem and state it as follows:

$$\textbf{LP}_{\mu}: \quad \text{Minimize } F_{\mu}(\boldsymbol{x}) = \boldsymbol{cx} - \mu(\textstyle\sum_j \ln x_j),$$

$$\text{st } \boldsymbol{Ax}=\boldsymbol{b},\ \boldsymbol{x}>\boldsymbol{0}$$

where $\mu>0$ is some scalar "penalty" also called the ***barrier parameter***. Note that when some $x_j\rightarrow 0$, the second term in the objective $-\mu(\sum_j \ln x_j)\rightarrow\infty$; this effectively keeps x_j from getting too close to the boundary.

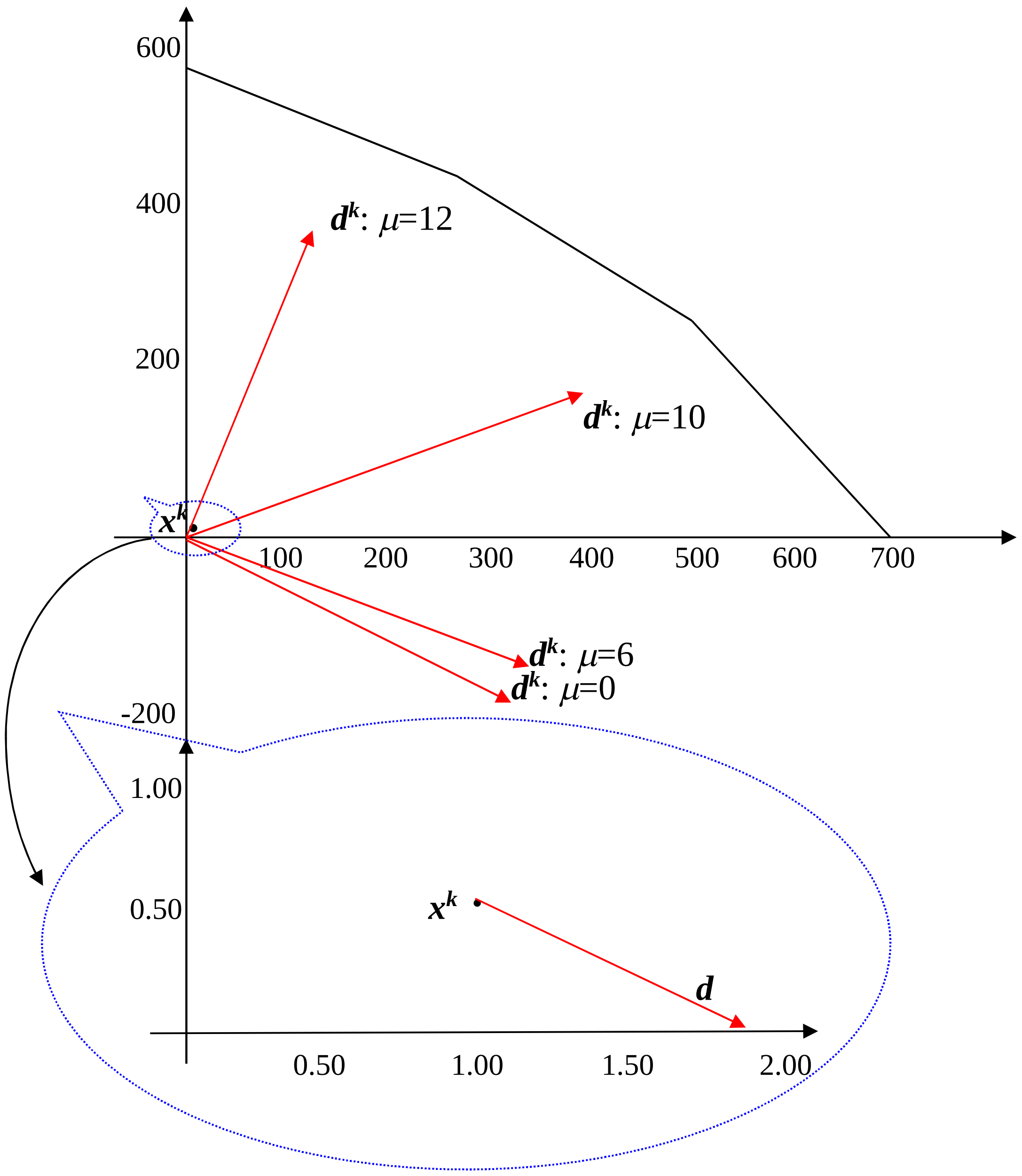
Consider the following example from Martin's 1999 LP book:

Min $-10x_1 - 9x_2$

$$\begin{array}{llr}
\text{st} & 7x_1 + 10x_2 + x_3 & \leq 6300 \\
 & 3x_1 + 5x_2 \quad + x_4 & \leq 3600 \\
 & 3x_1 + 2x_2 \quad\quad + x_5 & \leq 2124 \\
 & 2x_1 + 5x_2 \quad\quad\quad + x_6 & \leq 2700 \qquad \text{All } x_j\geq 0.
\end{array}$$

Consider the current feasible point $\boldsymbol{x}^{\boldsymbol{k}}= [1 \quad 0.5 \quad 6288 \quad 3594.5 \quad 2120 \quad 2695.5]$ with objective = -14.5.

The feasible region is shown below:



The negative gradient (direction of maximum improvement) of the objective function at the point $\boldsymbol{x^k}$ is given by $-\boldsymbol{c}$ = [-10 -9 0 0 0 0]^T and projecting this on to the null space $\{\boldsymbol{x}|\boldsymbol{Ax}=\boldsymbol{0}\}$ to maintain feasibility yields the direction

$\boldsymbol{d} = -(\boldsymbol{I} - \boldsymbol{A^T}(\boldsymbol{AA^T})^{-1}\boldsymbol{A})\boldsymbol{c^T}$ = [0.67 -0.38 -0.90 -0.12 -1.25 0.55]^T.

As shown above very little movement is possible before hitting the boundary!

In the affine-scaling method, we got around this by rescaling each time so that the point comes to the center of the transformed region. In the barrier function approach, consider the negative gradient direction of $F_\mu(\boldsymbol{x}) = \boldsymbol{cx} - \mu(\sum_j \ln x_j)$: at the point $\boldsymbol{x^k}$ this direction is given by $-(\boldsymbol{c}^T - \mu \boldsymbol{D_k}^{-1}\boldsymbol{e})$, as opposed to $-\boldsymbol{c}^T$ with the original objective. Recall that here $\boldsymbol{D_k}$= Diag($\boldsymbol{x^k}$) and $\boldsymbol{e}$=[1 1 ... 1]^T. Thus the feasible direction obtained by the usual projection on to the null space $\{\boldsymbol{x}|\boldsymbol{Ax}=\boldsymbol{0}\}$ yields

$$\boldsymbol{d} = -(\boldsymbol{I} - \boldsymbol{A^T}(\boldsymbol{AA^T})^{-1}\boldsymbol{A})(\boldsymbol{c^T} - \mu\boldsymbol{D_k^{-1}}\boldsymbol{e})$$

The Table below lists this search direction for several values of μ:

μ	$\boldsymbol{d} = -(\boldsymbol{I} - \boldsymbol{A^T}(\boldsymbol{AA^T})^{-1}\boldsymbol{A})(\boldsymbol{c^T} - \mu\boldsymbol{D_k^{-1}}\boldsymbol{e})$
0	[0.67 -0.38 -0.90 -0.12 -1.25 0.55]^T
6	[0.37 -0.11 -1.53 -0.58 -0.91 -0.21]^T
10	[0.18 0.07 -1.95 -0.89 -0.68 -0.71]^T
12	[0.08 0.16 -2.16 -1.04 -0.56 -0.96]^T

These are pictured on the previous page - note that the effect of μ is to "push" the projected negative gradient away from the boundary of $x_2 \geq 0$: as the value of μ is increased the effect is to rotate the direction of the projected negative gradient in a counter-clockwise direction.

Note that problem LP$_\mu$ is a *nonlinear* programming problem with a strictly convex objective: as long as $\mu>0$, it has a unique global optimum with all $x_j>0$.

To obtain the optimal solution we define a vector of Lagrange multipliers $\boldsymbol{w} \in \mathrm{R}^m$ (one for each constraint in $\boldsymbol{Ax=b}$) and the Lagrangian function

$$L_\mu(\boldsymbol{x,y}) = \boldsymbol{cx} - \mu(\textstyle\sum_{\mathrm{j}} \ln x_j) - \boldsymbol{w}^{\boldsymbol{T}}(\boldsymbol{Ax-b})$$

The Karush-Kuhn-Tucker necessary conditions for optimality require that the derivative of L_μ with respect to each x_j and each w_i must be equal to 0 at the optimum. This yields (m+n) equations in (m+n) unknowns which we solve for optimal $\boldsymbol{x}$ and $\boldsymbol{w}$ vectors.

$$\partial L_\mu/\partial \boldsymbol{w} = \boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0} \qquad \text{(w.r.t. } \boldsymbol{w}\text{: } m \text{ equations)}$$

$$\partial L_\mu/\partial \boldsymbol{x} = \boldsymbol{c} - \mu\boldsymbol{D}^{-1}\boldsymbol{e} - \boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} = \boldsymbol{0} \qquad \text{(w.r.t. } \boldsymbol{x}\text{: } n \text{ equations)}$$

where $\boldsymbol{D}$=Diag[$x_1, x_2, \ldots, x_n$] as usual.

In fact, if we define $\{\boldsymbol{s} \in \mathrm{R}^n | \boldsymbol{s} = \mu\boldsymbol{D}^{-1}\boldsymbol{e}\}$, i.e., ($s_j = \mu/x_j$), then these reduce to

(1) $\boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0}$

(2) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}^{\mathrm{T}}$

(3) $\boldsymbol{s} = \mu\boldsymbol{D}^{-1}\boldsymbol{e}$, i.e., $s_jx_j = \mu$ for all j

Also note that $\boldsymbol{x>0}$ (since the objective of LP$_\mu$ is convex) and thus, $\boldsymbol{s>0}$ (since μ>0). These conditions should look familiar; they are the usual LP optimality conditions! Respectively, (1) is primal feasibility, (2) is dual feasibility and (3) is an approximation to complementary slackness, which will approach exactness as $\mu \rightarrow 0$. If these equations were linear then we could just solve the system and linear programming would be trivial - unfortunately, they are ***nonlinear*** and so solving them is a nontrivial task!

Suppose we are solving LP$_\mu$ for a given $\mu>0$. Then as long as the following barrier assumptions are satisfied:

a) The set $\{\boldsymbol{x}\in\mathrm{R}^n | \boldsymbol{Ax}=\boldsymbol{b}, \boldsymbol{x}>0\}$ is non-empty

b) The set $\{(\boldsymbol{w},\boldsymbol{s})\in\mathrm{R}^m\times\mathrm{R}^n | \boldsymbol{A}^\mathrm{T}\boldsymbol{w}+\boldsymbol{s}=\boldsymbol{c}^\mathrm{T}, \boldsymbol{z}>0\}$ is non-empty

c) The constraint matrix $\boldsymbol{A}$ has rank m.

the system of equations (1), (2), (3) is guaranteed to have a unique solution - say $\boldsymbol{x}_\mu$, $\boldsymbol{w}_\mu$, $\boldsymbol{s}_\mu$. Here $\boldsymbol{x}_\mu$ is optimal in LP$_\mu$ and $(\boldsymbol{w}_\mu, \boldsymbol{s}_\mu)$ solves the corresponding dual.

For the **exact** LP optimality μ should be equal to zero, so the barrier approach will be to start with some positive value for μ and solve LP$_\mu$ for successively smaller μ.

FACT: Given the barrier assumptions above, $(\boldsymbol{x}_\mu, \boldsymbol{w}_\mu, \boldsymbol{s}_\mu)$ converges to an optimal primal-dual solution to the original LP as $\mu\to 0$.

Since $(\boldsymbol{x}_\mu, \boldsymbol{w}_\mu, \boldsymbol{s}_\mu)$ is unique for each μ this set of minimizers that are generated trace out a path called the **central path** or **central trajectory**.

In particular, the solution to LP$_\mu$ when $\mu=\infty$ is the solution to Minimize $-\sum_\mathrm{j} \ln x_j$, st $\boldsymbol{Ax}=\boldsymbol{b}$ is called the "analytic center" of the feasible region and represents the point "farthest away" from the boundaries, in the sense of maximizing $\prod_\mathrm{j} x_j$.

The phrase "path-following" method refers to the fact that these methods follow this central path. In practice, they don't <u>exactly</u> follow the central path since equation (3) is only solved approximately at each step - they just stay "close" to the central path.

There are three path following methods (primal, dual and primal-dual) and they differ only in how equations (3) $\boldsymbol{s}=\mu\boldsymbol{D}^{-1}\boldsymbol{e}$ is written and approximated.

PRIMAL PATH FOLLOWING ALGORITHM

The basic idea here is to use Newton's method which is a procedure for solving a system of *n* nonlinear equations in *n* unknowns, i.e., of the form $f_1(\boldsymbol{x})=0$, $f_2(\boldsymbol{x})=0,\ldots,$ $f_n(\boldsymbol{x})=0$, where $\boldsymbol{x}\in\mathrm{R}^n$. It proceeds by guessing an initial solution - say $\boldsymbol{x^0}$ and refining the guess $\boldsymbol{x^k}$ at each iteration *k* via a correction step $\boldsymbol{\Delta x}$; thus $\boldsymbol{x^{k+1}} = \boldsymbol{x^k} + \boldsymbol{\Delta x}$**.** The correction step $\boldsymbol{\Delta x}$ is computed by defining a first order Taylor series approximation.

Now recall that the system to be solved for finding the optimum solution to LP_μ is the following, where (1) and (2) are linear while (3) is nonlinear

(1) $\boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0}$

(2) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}^{\mathrm{T}}$

(3) $\boldsymbol{s}=\mu\boldsymbol{D}^{-1}\boldsymbol{e}$, i.e., $s_j = (\mu/x_j)$, i.e, $(\mu/x_j) - s_j=0$ for all j

Suppose at iteration *k* we are at $\boldsymbol{x^k}$, $\boldsymbol{w^k}$, $\boldsymbol{s^k}$. In this method, we will always ensure that (1) and (2) are exactly satisfied at each iteration (i.e., that $\boldsymbol{Ax^k}=\boldsymbol{b}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w^k} + \boldsymbol{s^k} = \boldsymbol{c}^{\mathrm{T}}$). However, the troublesome (nonlinear) equations (3) are solved only approximately. Define $f(x_j, s_j) = (\mu/x_j) - s_j$. A first order Taylor series approximation for $f(x_j, s_j)$ yields

$$f(x_j, s_j) \approx f(x_j^k, s_j^k) + \nabla f(x_j^k, s_j^k)^{\mathrm{T}}\begin{bmatrix} x_j - x_j^k \\ s_j - s_j^k \end{bmatrix}$$

$$= [(\mu_k/x_j^k) - s_j^k] + [-\mu_k/(x_j^k)^2 \quad -1] \begin{bmatrix} x_j - x_j^k \\ s_j - s_j^k \end{bmatrix}$$

$$= (\mu_k/x_j^k) - s_j^k - \mu_k[x_j/(x_j^k)^2] + \mu_k/(x_j^k) - s_j + s_j^k$$

$$= \mu_k/(x_j^k)^2[2x_j^k - x_j] - s_j$$

In other words, equations (3), which were given by

$$(\mu/x_j) - s_j = 0$$

may be approximated as

$$\mu_k/(x_j^k)^2[2x_j^k - x_j] - s_j = 0, \text{ i.e,} \qquad \mu_k/(x_j^k)^2[2x_j^k - x_j] = s_j$$

Note that in the above, μ_k and x_j^k are <u>known</u> so that we have a **linear** approximation to the original (nonlinear) equation.

Now, suppose we take a step of size $\Delta\boldsymbol{x}$, $\Delta\boldsymbol{w}$, $\Delta\boldsymbol{s}$ from $\boldsymbol{x^k}$, $\boldsymbol{w^k}$, $\boldsymbol{s^k}$ to get to $\boldsymbol{x^{k+1}}$, $\boldsymbol{w^{k+1}}$, $\boldsymbol{s^{k+1}}$, i.e. $\boldsymbol{x^k} + \Delta\boldsymbol{x} = \boldsymbol{x^{k+1}}$, $\boldsymbol{w^k} + \Delta\boldsymbol{w} = \boldsymbol{w^{k+1}}$ and $\boldsymbol{s^k} + \Delta\boldsymbol{s} = \boldsymbol{s^{k+1}}$

(thus, $x_j^{k+1} - x_j^k = \Delta x_j$, $w_j^{k+1} - w_j^k = \Delta w_j$, $s_j^{k+1} - s_j^k = \Delta s_j$ ...)

We want this new point to satisfy

1. $\boldsymbol{Ax^{k+1} - b = 0}$
2. $\boldsymbol{A}^T\boldsymbol{w^{k+1}} + \boldsymbol{s^{k+1}} = \boldsymbol{c}^T$

where $\boldsymbol{s^{k+1}}$ is defined via $s_j^{k+1} = \mu_k/(x_j^k)^2[2x_j^k - x_j^{k+1}] = \mu_k/(x_j^k)^2[x_j^k - \Delta x_j]$ for all j.

In vector form we may rewrite $\boldsymbol{s^{k+1}} = \mu_k(\boldsymbol{D_k^2})^{-1}[\boldsymbol{x}^k - \Delta x]$

where $\boldsymbol{D_k}$=Diag$[x_1^k \quad x_2^k \quad ... \quad x_n^k]$ as usual.

Thus we are reduced to solving

$\boldsymbol{A}[\boldsymbol{x}^k + \Delta \boldsymbol{x}] - \boldsymbol{b} = \boldsymbol{0}$, i.e.,

1) $\boldsymbol{A}\Delta\boldsymbol{x} = \boldsymbol{0}$ (since $\boldsymbol{A}\boldsymbol{x}^k - \boldsymbol{b} = \boldsymbol{0}$), and

$\boldsymbol{A}^T[\boldsymbol{w}^k + \Delta\boldsymbol{w}] + \mu_k(\boldsymbol{D}_k^2)^{-1}[\boldsymbol{x}^k - \Delta x] = \boldsymbol{c}^T$, i.e.,

2) $\mu_k(\boldsymbol{D}_k^2)^{-1}\Delta x + \boldsymbol{A}^T\Delta\boldsymbol{w} = \boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w}^k - \mu_k(\boldsymbol{D}_k)^{-1}\boldsymbol{e}$

$= \boldsymbol{s}^k - \mu_k(\boldsymbol{D}_k)^{-1}\boldsymbol{e}$ (since $\boldsymbol{A}^T\boldsymbol{w}^k + \boldsymbol{s}^k = \boldsymbol{c}^T$)

This is a linear system of $m+n$ equations in $m+n$ unknowns and we may solve for $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{w}$. If we define the usual $\boldsymbol{P}_k = \boldsymbol{A}\boldsymbol{D}_k$ then (after some messy linear algebra!) it may be shown that

$$\Delta\boldsymbol{w} = (\boldsymbol{P}_k\boldsymbol{P}_k^T)^{-1}\boldsymbol{P}_k[\boldsymbol{D}_k\boldsymbol{s}^k - \mu_k\boldsymbol{e}]$$

$$\Delta\boldsymbol{x} = -\frac{1}{\mu_\kappa}\boldsymbol{D}_k[\boldsymbol{I} - \boldsymbol{P}_k^T(\boldsymbol{P}_k\boldsymbol{P}_k^T)^{-1}\boldsymbol{P}_k](\boldsymbol{D}_k\boldsymbol{c}^T - \mu_k\boldsymbol{e})$$

and since $\boldsymbol{s}^{k+1} = \boldsymbol{c}^T - \boldsymbol{A}^T\boldsymbol{w}^{k+1}$, *i.e.,* $\boldsymbol{s}^k + \Delta\boldsymbol{s} = \boldsymbol{c}^T - \boldsymbol{A}^T(\boldsymbol{w}^k + \Delta\boldsymbol{w}) = \boldsymbol{s}^k - \boldsymbol{A}^T\Delta\boldsymbol{w}$, we have

$$\Delta\boldsymbol{s} = -\boldsymbol{A}^T\Delta\boldsymbol{w}$$

The above values for $\Delta\boldsymbol{x}$, $\Delta\boldsymbol{w}$, $\Delta\boldsymbol{s}$ constitute the ***Newton*** direction and the process of computing these is called a ***Newton Step***. These values allow us to find the next guess at the solution $\boldsymbol{x}^{k+1}$, $\boldsymbol{w}^{k+1}$ and $\boldsymbol{s}^{k+1}$. At this point we would set $k=k+1$ and find another Newton step and continue the process until we converge to an exact solution to the original (nonlinear) system.

With classical barrier function theory, we solve LP_μ exactly by finding the exact solution to the system (1), (2) and (3) through a series of Newton steps as shown above. At this point, we would then reduce μ_k to μ_{k+1} and solve a new set of equations for the next point and so on and the successive optima would converge to the optimum for the original LP. In other words, each optimum solution to LP_μ is a point on the central trajectory (by definition) and we would move along this path.

In practice, we don't take *a series* of Newton steps (to actually converge to the central trajectory) for a given μ_k before reducing the value to μ_{k+1} at iteration k+1. Rather, when we solve LP_μ with $\mu=\mu_k$ we only take **one single** Newton step and move along the Newton direction while ensuring that no x_j and no s_j becomes zero. We then immediately reduce μ_k to μ_{k+1} to get a *new* problem LP_μ! In other words we never solve (1), (2) and (3) exactly for any LP_μ - rather we find an approximate solution in one step and use this as the starting point for the next LP_μ.

This procedure continues until at some iteration, $\mu_k<\varepsilon$ (some suitably small tolerance) at which stage $\boldsymbol{x}^{\boldsymbol{k}}$ converges to the optimum solution $\boldsymbol{x}^*$ to the original LP. It may be shown that this set of single steps ensures convergence in polynomial time. One nice thing about the path following approach is that we have a feasible primal vector $\boldsymbol{x}$ and a feasible dual vector $\boldsymbol{w}$ available at every iteration - the amount of infeasibility in (3) measures the "duality gap."

15261

COMPARISON TO AFFINE SCALING ALGORITHM

It is interesting to compare the search direction we found at each stage for ***x*** with the direction we got at each iteration of the affine scaling method.

With affine-scaling the search direction- say ***d*** - was given by (refer to the NOTE after Step 3 on page 11):

$$\Delta \boldsymbol{x} = \boldsymbol{d} \quad = -[\boldsymbol{D_k}^2 - \boldsymbol{D_k}^2\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{D_k}^2\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A}\boldsymbol{D_k}^2]\,\boldsymbol{c}^{\mathrm{T}}$$

$$= -\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{D_k}\boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{D_k}^2\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A}\boldsymbol{D_k}]\,\boldsymbol{D_k}\,\boldsymbol{c}^{\mathrm{T}}$$

$$= -\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}]\boldsymbol{D_k}\,\boldsymbol{c}^{\mathrm{T}} \quad = \quad \boldsymbol{d} \qquad \text{(say)}$$

With the primal path following method, we just saw that this search direction - say $\boldsymbol{d}_\mu$ - is given by

$$\Delta \boldsymbol{x} = \boldsymbol{d}_\mu \quad = -\frac{1}{\mu_\kappa}\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}](\boldsymbol{D_k}\,\boldsymbol{c}^{\mathrm{T}} - \mu_k\boldsymbol{e})$$

$$= -\frac{1}{\mu_\kappa}\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}]\boldsymbol{D_k}\,\boldsymbol{c}^{\mathrm{T}} + \frac{1}{\mu_\kappa}\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}](\mu_k\boldsymbol{e})$$

$$= \frac{1}{\mu_\kappa}\boldsymbol{d} + \boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}](\boldsymbol{e})$$

Thus in the primal path following algorithm, the search direction is the affine scaling direction, corrected by a factor equal to $\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_k}\boldsymbol{P_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}](\boldsymbol{e})$. The latter may be viewed as a force that pushes a solution away from the boundary; thus it is also occasionally referred to as a "centering force".

Note that as $\mu_k \rightarrow 0$ (so that LP_μ becomes closer to the original LP), the affine-scaling search direction dominates the correction term and $\boldsymbol{d}_\mu \rightarrow \boldsymbol{d}$.

PRIMAL PATH FOLLOWING: ALGORITHM SPECIFICATION

STEP 0 INITIALIZATION: Start with $k=0$ and $\boldsymbol{x^0}$, $\boldsymbol{s^0} > \boldsymbol{0}$ and $\boldsymbol{w^0}$ such that $\boldsymbol{Ax^0=b}$ and $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w^0}+\boldsymbol{s^0}=\boldsymbol{c}^{\mathrm{T}}$. Define suitable ε, $\mu_0>0$ and $\alpha,\theta\in(0,1)$.

STEP 1 NEWTON SEARCH DIRECTION: Define $\boldsymbol{D_k}$ and compute $\boldsymbol{P_k}=\boldsymbol{AD_k}$; then find the movement directions via

$$\Delta\boldsymbol{w} = (\boldsymbol{P_kP_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}[\boldsymbol{D_ks^k} - \mu_k\boldsymbol{e}]$$

$$\Delta\boldsymbol{x} = -\frac{1}{\mu_\kappa}\boldsymbol{D_k}[\boldsymbol{I} - \boldsymbol{P_k}^{\mathrm{T}}(\boldsymbol{P_kP_k}^{\mathrm{T}})^{-1}\boldsymbol{P_k}](\boldsymbol{D_k}\,\boldsymbol{c}^{\mathrm{T}} - \mu_k\boldsymbol{e})$$

$$\Delta\boldsymbol{s} = -\boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{w}$$

STEP 2 CALCULATE NEW SOLUTION via

$$\boldsymbol{x^{k+1}} = \boldsymbol{x^k}+\alpha^k_P\,\Delta\boldsymbol{x},$$

$$\boldsymbol{w^{k+1}} = \boldsymbol{w^k}+\alpha^k_D\Delta\boldsymbol{w}$$

$$\boldsymbol{s^{k+1}}=\boldsymbol{s^k}+\alpha^k_D\Delta\boldsymbol{s}$$

where the step sizes are determined by performing the ratio tests:

$$\alpha^k_P = \alpha[\mathrm{Min}_j\{x_j^k/|\Delta x_j|: \Delta x_j<0\}]$$

$$\alpha^k_D = \alpha[\mathrm{Min}_j\{s_j^k/|\Delta s_j|: \Delta s_j<0\}]$$

STEP 3 TERMINATION CHECK: If $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x^k} - \boldsymbol{b}^{\mathrm{T}}\boldsymbol{w^k} \geq \varepsilon$, update $\mu_{k+1}=\theta\mu_k$. Update k to k+1 and return to Step 1; otherwise, stop.

DUAL PATH FOLLOWING ALGORITHM

This method is based on the same principles that we just saw for the primal path following method - the difference is that instead of the barrier function being formed for primal nonnegativity constraints ($\boldsymbol{x}\geq\boldsymbol{0}$), it is formed for the dual nonnegativity constraints. The dual barrier optimization problem is thus given by

DLP$_\mu$: Maximize $G_\mu(\boldsymbol{x}) = \boldsymbol{b}^T\boldsymbol{w} + \mu(\sum_j \ln s_j)$,

st $\boldsymbol{A}^T\boldsymbol{w}+\boldsymbol{s}=\boldsymbol{c}^T$, $\boldsymbol{s}>\boldsymbol{0}$

Defining the appropriate Lagrangian and deriving the Karush-Kuhn Tucker leads to the identical conditions we had earlier, namely

1) $\boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0}$
2) $\boldsymbol{A}^T\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}$
3) $\boldsymbol{s}=\mu\boldsymbol{D}^{-1}\boldsymbol{e}$, i.e., $s_jx_j=\mu$ for all j

Once again we will solve the same system, however, in this approach (3) is stated a little differently. Rather than stating it as $(\mu/x_j) - s_j=0$ for all j as we did earlier, we state this as $(\mu/s_j) - x_j=0$ for all j!

Define $f(x_j, s_j) = (\mu/s_j)-x_j$. A first order Taylor series approximation for $f(x_j, s_j)$ yields

$$f(x_j, s_j) \approx [(\mu_k/s_j^k)-x_j^k] + [-\mu_k/(s_j^k)^2 \quad -1]\begin{bmatrix} x_j - x_j^k \\ s_j - s_j^k \end{bmatrix}$$

$$= (\mu_k/s_j^k)- x_j^k - \mu_k[x_j/(s_j^k)^2] + \mu_k/(s_j^k) - x_j + x_j^k$$

$$= \mu_k/(s_j^k)^2[2s_j^k - s_j] - x_j$$

In other words, equations (3) which were given by

$$(\mu/s_j) - x_j = 0$$

may be approximated as

$$\mu_k/(s_j^k)^2[2s_j^k - s_j] = x_j$$

Compare this with what we did with the primal approach!

In vector form we may rewrite $\boldsymbol{x}^{k+1} = \mu_k(\boldsymbol{S_k}^{2})^{-1}[\boldsymbol{s}^k - \Delta s]$

where $\boldsymbol{S_k}$= Diag[$s_1^k \;\; s_2^k \;\; \ldots \; s_m^k$].

Thus we are reduced to solving

$\boldsymbol{Ax}^{k+1} - \boldsymbol{b} = \boldsymbol{0}$, i.e.,

1) $\mu_k\boldsymbol{A}(\boldsymbol{S_k}^{2})^{-1}[\boldsymbol{s}^k - \Delta s] - \boldsymbol{b} = \boldsymbol{0}$ (since $\boldsymbol{Ax}^k - \boldsymbol{b} = \boldsymbol{0}$), and

$\boldsymbol{A}^T[\boldsymbol{w}^k + \Delta\boldsymbol{w}] + \boldsymbol{s}^k + \Delta s = \boldsymbol{c}$, i.e.,

2) $\boldsymbol{A}^T\Delta\boldsymbol{w} + \Delta\boldsymbol{s} = \boldsymbol{0}$ (since $\boldsymbol{A}^T\boldsymbol{w}^k + \boldsymbol{s}^k = \boldsymbol{c}$)

Substituting $\Delta\boldsymbol{s} = -\boldsymbol{A}^T\Delta\boldsymbol{w}$ from (2) into (1) and simplifying yields

$$\Delta\boldsymbol{w} = (\boldsymbol{AS_k}^{-2}\boldsymbol{A}^T)^{-1}[\frac{1}{\mu_\kappa}\boldsymbol{b} - \boldsymbol{AS_k}^{-1}\boldsymbol{e}]$$

$$\Delta\boldsymbol{x} = [-\boldsymbol{S_k}^{-1} + \boldsymbol{S_k}^{-2}\boldsymbol{A}^T(\boldsymbol{AS_k}^{-2}\boldsymbol{A}^T)^{-1}\boldsymbol{AS_k}^{-1}](\boldsymbol{S_k x}^k - \mu_k\boldsymbol{e})$$

$$\Delta\boldsymbol{s} = -\boldsymbol{A}^T\Delta\boldsymbol{w}$$

The method is identical to the primal path following except that it uses the above Newton search directions in Step 1 at each iteration.

PRIMAL-DUAL PATH FOLLOWING ALGORITHM

This method, first proposed by Megiddo and later extended by Kojima et al. is generally considered the best interior-point method. It is similar to the previous two methods, BUT this form does not correspond to any clearly defined barrier function! Consider the Karush-Kuhn Tucker conditions we had in both earlier cases, namely

1) $\boldsymbol{Ax} - \boldsymbol{b} = \boldsymbol{0}$

2) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w} + \boldsymbol{s} = \boldsymbol{c}^{\mathrm{T}}$

3) $\boldsymbol{s}=\mu\boldsymbol{D}^{-1}\boldsymbol{e}$, i.e., $s_jx_j=\mu$ for all j

We will solve this system as usual, but now we restate (3) in yet another way.

Recall that for the primal approach we stated this as $(\mu/x_j) - s_j=0$ for all j, while for the dual approach we stated this as $(\mu/s_j) - x_j=0$ for all j.

This time we will state it as $\mu - x_js_j=0$

So if as usual, we define $f(x_j, s_j) = \mu - x_js_j$ the Taylor series approximation for yields

$$f(x_j, s_j) \approx (\mu_k - x_j^ks_j^k) + [-s_j^k \quad -x_j^k] \begin{bmatrix} x_j - x_j^k \\ s_j - s_j^k \end{bmatrix}$$

$$= \mu_k - x_j^ks_j^k - s_j^kx_j + s_j^kx_j^k - x_j^ks_j + x_j^ks_j^k \quad = \mu_k + x_j^ks_j^k - s_j^kx_j - x_j^ks_j$$

In other words, equations (3) which were given by

$$\mu - s_jx_j = 0$$

may be approximated as

$$\mu_k + x_j^ks_j^k = s_j^kx_j + x_j^ks_j = s_j^k(x_j^k + \Delta x_j) + x_j^k(s_j^k + \Delta s_j)$$

i.e., $\mu_k - x_j^k s_j^k = s_j^k \Delta x_j + x_j^k \Delta s_j$

$$\mu_k \boldsymbol{e} + \boldsymbol{D_k S_k e}^{\,k} = \boldsymbol{S}_k \Delta \boldsymbol{x} + \boldsymbol{D}_k \Delta \boldsymbol{s}$$

where $\boldsymbol{S_k}$ and $\boldsymbol{D_k}$ are the usual diagonal matrices.

Thus we are reduced to solving

$\boldsymbol{A}[\boldsymbol{x^k} + \Delta \boldsymbol{x}] - \boldsymbol{b} = \boldsymbol{0}$, i.e.,

1) $\boldsymbol{A}\Delta\boldsymbol{x} = \boldsymbol{0}$ (since $\boldsymbol{Ax^k} - \boldsymbol{b} = \boldsymbol{0}$), and

$\boldsymbol{A}^{\mathrm{T}}[\boldsymbol{w^k} + \Delta \boldsymbol{w}] + \boldsymbol{s}^k + \Delta s = \boldsymbol{c}^{\mathrm{T}}$, i.e.,

2) $\boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{w} + \Delta\boldsymbol{s} = \boldsymbol{0}$ (since $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{w^k} + \boldsymbol{s^k} = \boldsymbol{c}^{\mathrm{T}}$)

and $\mu_k \boldsymbol{e} + \boldsymbol{D_k S_k e}^{\,k} = \boldsymbol{S}_k \Delta \boldsymbol{x} + \boldsymbol{D}_k \Delta \boldsymbol{s}$ which we just derived above for (3)

Using (1) and (2) to eliminate $\Delta\boldsymbol{x}$ and $\Delta\boldsymbol{x}$ from this, and simplifying yields

$$\Delta\boldsymbol{w} = -(\boldsymbol{A D_k S_k}^{-1}\boldsymbol{A}^{\mathrm{T}})^{-1}\boldsymbol{A S_k}^{-1}[\mu \boldsymbol{e} - \boldsymbol{D_k S_k e}]$$

$$\Delta\boldsymbol{s} = -\boldsymbol{A}^{\mathrm{T}}\Delta\boldsymbol{w}$$

$$\Delta\boldsymbol{x} = \boldsymbol{S_k}^{-1}[(\mu \boldsymbol{e} - \boldsymbol{D_k S_k e}) - \boldsymbol{D_k}\Delta\boldsymbol{s}]$$

Once again, after this the method is identical to the earlier ones except that that it uses the above Newton search directions in Step 1 at each iteration.

KARMARKAR'S ALGORITHM

We will only look at the essential elements of this approach since it has been replaced with the better methods described earlier:

BASIC STRATEGY

STEP 0: Start with an interior point.

STEP 1: Transform the solution space so that the point is at the "center" of the (transformed) feasible region.

STEP 2: Move it in the steepest descent direction, stopping "a little" before hitting the boundary (so that the new point is also in the interior).

STEP 3: Transform the space once again so that this new point is at the center of the transformed polytope.

STEP 4: Keep repeating Steps 2 and 3 until an optimum is obtained with the desired accuracy.

BASIC QUESTION 1: How do we transform the space so that a point goes to the center?

ANSWER: By means of a ***Projective Transform***.

BASIC QUESTION 2: How do we find the direction that maximizes the improvement?

ANSWER: By a simple projection of a vector on to an affine space.

Karmarkar's Projective Transform

Consider the projective transform

$$\boxed{\begin{array}{l} \mathrm{T_A} : \mathbf{R}_+^n \to \Omega^n \\ \Omega^n = \{\boldsymbol{x} \in \mathbf{R}_+^{n+1} : \sum_{k=1}^{n+1} x_k = 1\} \\ A = (a_1, a_2, ..., a_n) > 0 \end{array}}$$

The transform T_A is defined as

$$\mathrm{T_A} = \frac{1}{1 + \sum_{k=1}^{n+1} \left(\frac{x_k}{a_k} \right)} \left[\frac{x_1}{a_1}, \frac{x_2}{a_2}, ..., \frac{x_k}{a_k}, 1 \right]$$

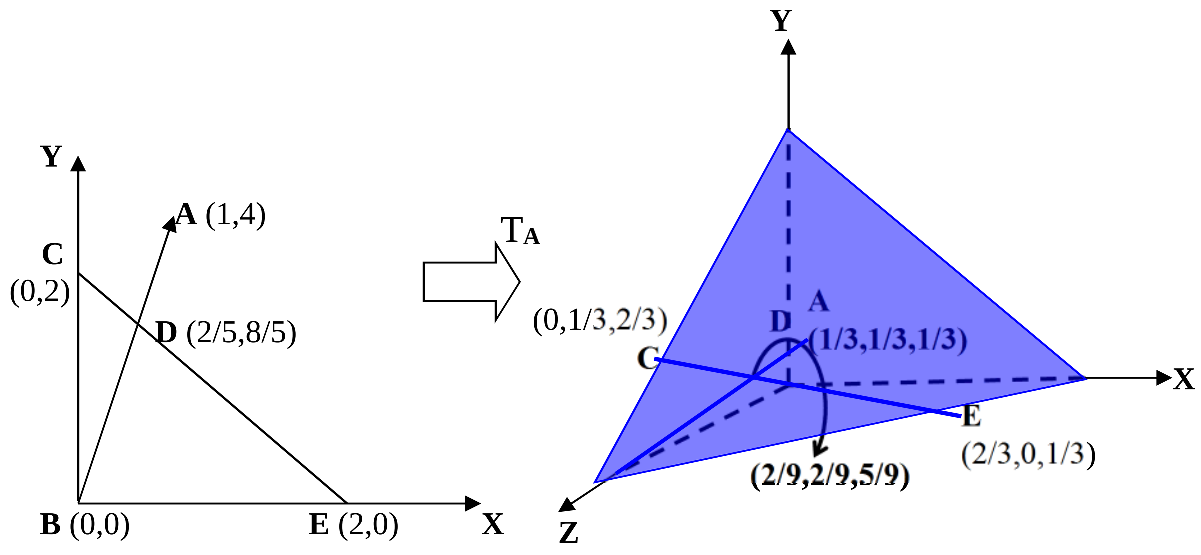
Note that

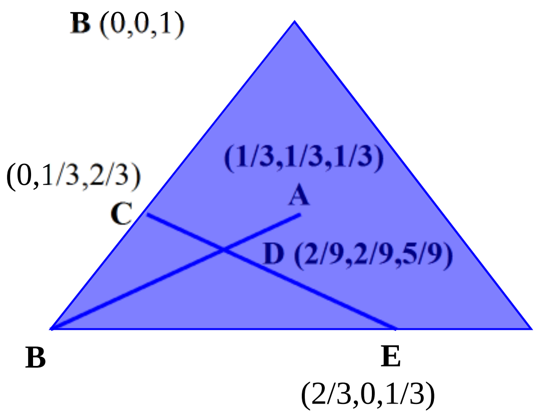
1. $\mathrm{T_A}(\mathrm{A}) = \left[\frac{1}{n+1}, \frac{1}{n+1}, ..., \frac{1}{n+1} \right]$ **("Centering")**

2. $\mathrm{T_A^{-1}}(x_1, x_2, ..., x_n, x_{n+1}) = \frac{1}{x_{n+1}} (a_1 x_1, a_2 x_2, ..., a_n x_n)$

3. If $\boldsymbol{y} = \mathrm{T_A}(\boldsymbol{x})$, then $\sum_{k=1}^{n+1} y_k = 1$

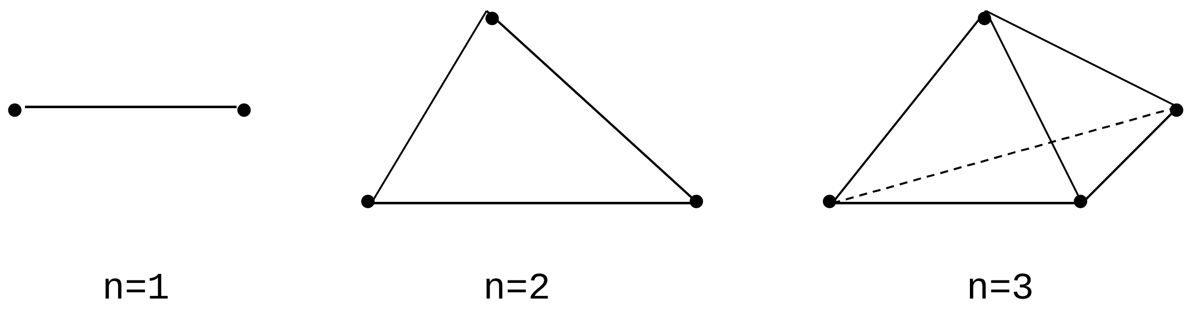
Also, note that points are mapped to points, line segments are mapped to line segments, and convex sets are mapped to convex sets.





DEFINITION 1: A simplex in n-dimensions is the convex hull of a set of (n+1) noncoplanar points (i.e., points not all on the same hyperplane in $\mathbf{R^n}$).

Examples are shown below for n=1,2,3:



DEFINITION 2: The *n*-dimensional *unit* simplex $\mathbf{\Omega^n}$ is the set of $\boldsymbol{x} \in \mathbf{R^n}$ which satisfy $\sum_{k=1}^{n+1} x_k = 1$, and $x_k \geq 0$ for all k.

$\mathbf{\Omega^n}$ is the convex hull of the (n+1) points $\begin{bmatrix}1\\0\\ \vdots\\0\end{bmatrix}, \begin{bmatrix}0\\1\\ \vdots\\0\end{bmatrix}, \ldots, \begin{bmatrix}0\\0\\ \vdots\\1\end{bmatrix}$.

Thus, $\mathbf{T_A}$ transforms the positive orthant $\mathbf{R^n_+}$ into a simplex in *n* dimensions (e.g., $\mathbf{R}^2$ is transformed into a 2-dimensional simplex).

However, notice that with this transform (in $\mathbf{R^2}$) the "points" $(0,\infty)$ and $(\infty,0)$ are transformed to the points (0,1,0) and (1,0,0). Furthermore, we are moving from 2 to 3 dimensions. In general, even though the simplex is in *n* dimensions, we have *n*+1 variables.

Instead, ...

...suppose our original feasible region was itself restricted to lie within a unit simplex of dimension (n-1), as opposed to lying within $\mathbf{R}_+^{\mathbf{n}}$. In other words we have the additional restriction that $\sum_{k=1}^{n+1} x_k = 1$. Then we can apply a similar centering transform which is simpler and has the same properties as **TA** above; as given by

$$\boxed{\mathrm{T_A} : \Omega^{n-1} \to \Omega^{n-1}, \quad \mathrm{T_A} = \frac{1}{\sum_{k=1}^{n} \left(\frac{x_k}{a_k} \right)} \left[\frac{x_1}{a_1}, \frac{x_2}{a_2}, \ldots, \frac{x_k}{a_k} \right]}$$

We now no longer mess around with infinity and we remain in the same dimension as the original problem.

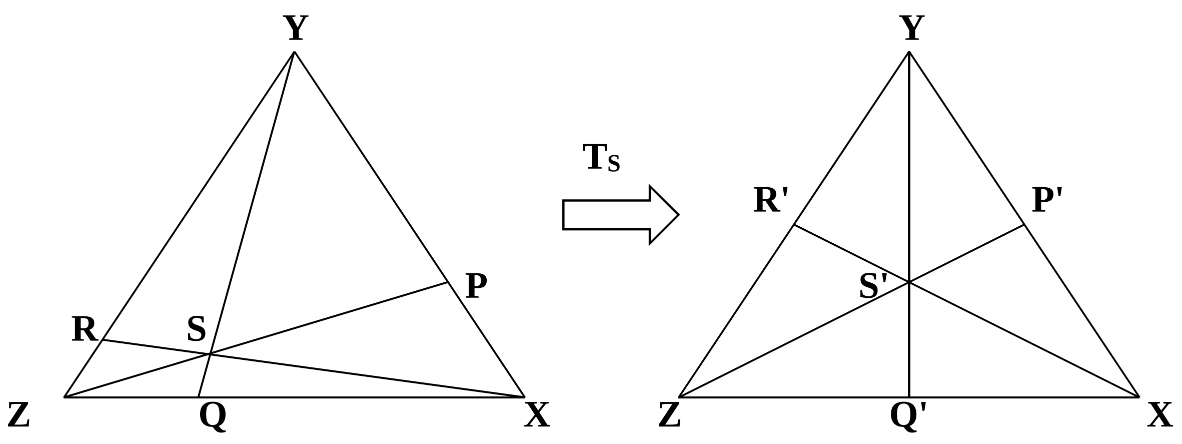
If we define

1. $\boldsymbol{D}$=Diag(a_1, a_2, ..., a_n) $\in \mathbf{R}^{\mathbf{n \times n}}$, and

2. $\boldsymbol{e}^{T}$ **=** [1 1 1 ... 1] $\in \mathbf{R}^{\mathbf{n}}$,

then the above transform may be rewritten as

$$\boxed{\mathrm{T_A}(\boldsymbol{x}) = \frac{\boldsymbol{D}^{-1}\boldsymbol{x}}{\boldsymbol{e}^{T}\boldsymbol{D}^{-1}\boldsymbol{x}}}$$



P=(3/4, 1/4, 0)	P'=$\mathbf{T}_S$(A)=(1/2, 1/2, 0)
Q=(1/3, 0, 2/3)	Q'=$\mathbf{T}_S$(B)=(1/2, 0, 1/2)
R=(0, 1/7, 6/7)	R'=$\mathbf{T}_S$(C)=(0, 1/2, 1/2)
S=(3/10, 1/10, 3/5)	S'=$\mathbf{T}_S$(S)=(1/3, 1/3, 1/3)

Note that

1. $\mathbf{T}_S(S) = (1/n, 1/n, \ldots 1/n)$ **(centering)**

2. $\mathbf{T}_S(\boldsymbol{y}) = \dfrac{\boldsymbol{Dy}}{\boldsymbol{e}^T \boldsymbol{Dy}}$ (**inversion)**

In summary...

- Given a unit simplex $\boldsymbol{\Omega}^{\mathbf{n-1}}$ and a point A lying in it, $\mathbf{T}_A$ transforms this into another unit simplex with the point A being mapped to the point A' at the "center" of the transformed simplex.
- Furthermore, there is a unique image of every point $\boldsymbol{x}$ in the original simplex given by a corresponding point $\boldsymbol{y}=\mathbf{T}_A(x)$ in the transformed simplex, and a unique image of every point $\boldsymbol{y}$ in the transformed simplex given by a

corresponding point $\boldsymbol{x}=\mathbf{T_A}^{-1}(\boldsymbol{y})$ in the original simplex. This is the essence of the projective transform...

STANDARD FORM REQUIRED FOR KARMARKAR'S METHOD

Before moving on to Basic Question 2, we state the following standard format in which every problem should be stated before being solved by Karmarkar's method:

PROGRAM $\mathbf{P}$

Minimize $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$

st $\quad \boldsymbol{Ax} = \boldsymbol{0}$

$\boldsymbol{e}^{T}\boldsymbol{x} = \boldsymbol{1}$

$x^{\mathrm{T}} \geq \boldsymbol{0}$

where $\boldsymbol{A}$ is $m\times n$ and has rank m, $n\geq 2$, $\boldsymbol{c}$ and $\boldsymbol{x}$ are vectors of $\boldsymbol{n}$ elements each, and $\boldsymbol{0}$ is a column vector of m zeroes.

Additionally, the following should also hold:

A1) The point $\boldsymbol{x_0} = [1/n, 1/n, ..., 1/n]$ is feasible in $\mathbf{P}$

A2) The optimal objective value of $\mathbf{P}$ is zero.

QUESTION: How do we convert a general LP of the form $\{\text{Min } \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{Ax}=\boldsymbol{b},\ \boldsymbol{x}\geq\boldsymbol{0}\}$ into this restrictive format ???

ANSWER: Actually, quite easily!

Refer to the Appendix for an example on this...

ON TO BASIC QUESTION 2...

Having accomplished the transformation, how do we then move at each iteration?

Program **P** was $\{\text{Min } \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x} \mid \boldsymbol{Ax}=\boldsymbol{0},\ \boldsymbol{e}^T\boldsymbol{x}=1,\ \boldsymbol{x}\geq\boldsymbol{0}\}$.

Recall that $\boldsymbol{y} = \boldsymbol{T}_{\boldsymbol{x}^k}(\boldsymbol{x}) = \dfrac{\boldsymbol{D}_k^{-1}\boldsymbol{x}}{\boldsymbol{e}^T\boldsymbol{D}_k^{-1}\boldsymbol{x}}$ & $\boldsymbol{x} = \boldsymbol{T}_{\boldsymbol{x}^k}^{-1}(\boldsymbol{y}) = \dfrac{\boldsymbol{D}_k\boldsymbol{y}}{\boldsymbol{e}^T\boldsymbol{D}_k\boldsymbol{y}}$

Thus **P** may be rewritten in terms of the transformed variables as

$$\{\text{Min } \frac{\mathbf{c}^T\mathbf{D}_\mathbf{k}\mathbf{y}}{\mathbf{e}^T\mathbf{D}_\mathbf{k}\mathbf{y}} \mid \mathbf{AD}_\mathbf{k}\mathbf{y} = \mathbf{0}, \mathbf{e}^T\mathbf{y} = 1, \mathbf{y} \geq 0\}.$$

However, we know from (A2) that the optimal value of this problem is 0. Then since $\boldsymbol{e}^T\boldsymbol{D}_k\boldsymbol{y}$ is always positive, we may minimize $\boldsymbol{cD}_k\boldsymbol{y}$ as opposed to the nonlinear objective above. Thus, if we define

$$\bar{\mathbf{c}}^T = \mathbf{c}^T\mathbf{D}_\mathbf{k}, \qquad \boldsymbol{P}_k = \begin{bmatrix} \boldsymbol{AD}_k \\ \boldsymbol{e}^T \end{bmatrix}, \qquad \boldsymbol{b} = \begin{bmatrix} \boldsymbol{0} \\ 1 \end{bmatrix}$$

the problem reduces to

$$\{\text{Min } \bar{\boldsymbol{c}}^{\mathrm{T}}\boldsymbol{y} \mid \boldsymbol{P}_k\boldsymbol{y}=\boldsymbol{b},\ \boldsymbol{y}\geq\boldsymbol{0}\}.$$

Suppose we are currently at the (feasible) point $\boldsymbol{y}^k$. Just as in the affine scaling method we compute the projection matrix $\mathcal{P} = [\boldsymbol{I}-\boldsymbol{P}_k^{\mathrm{T}}(\boldsymbol{P}_k\boldsymbol{P}_k^{\mathrm{T}})^{-1}\boldsymbol{P}_k]$, and use the direction $\boldsymbol{d}^* = -\mathcal{P}\,\bar{\boldsymbol{c}}$ which uses $\mathcal{P}$ to project the direction of steepest descent on to the feasible region. Again, in practice, we don't explicitly compute $[\boldsymbol{I}-\boldsymbol{P}_k^{\mathrm{T}}(\boldsymbol{P}_k\boldsymbol{P}_k^{\mathrm{T}})^{-1}\boldsymbol{P}_k]$. Instead, we first solve the system of linear equations $(\boldsymbol{P}_k\boldsymbol{P}_k^{\mathrm{T}})\boldsymbol{w} = \boldsymbol{P}_k\bar{\boldsymbol{c}}$ for the vector $\boldsymbol{w}$ $(= (\boldsymbol{P}_k\boldsymbol{P}_k^{\mathrm{T}})^{-1}\boldsymbol{P}_k\bar{\boldsymbol{c}})$and then obtain $\boldsymbol{d}^*$ as $\boldsymbol{d}^* = -[\bar{\boldsymbol{c}} - \boldsymbol{P}_k^{\mathrm{T}}\boldsymbol{w}]$.

BACK TO BASIC QUESTION 2...

Knowing the direction of movement, how far to move ???

Suppose that (by using $\boldsymbol{T}_{\boldsymbol{x}^k}$) the current point $\boldsymbol{y}^{\boldsymbol{k}}$ is at the center of the unit simplex $\boldsymbol{e}^{\mathrm{T}}\boldsymbol{y} = 1$. Let us construct an *n*-dimensional "ball" $\mathfrak{B}$ (a circle within a triangle in 2 dimensions, a sphere within a triangular polyhedron in 3 dimensions, etc.) with its center at $\boldsymbol{y}^{\boldsymbol{k}}$, with a radius *r* such that $\mathfrak{B}$ is *exactly inscribed* within the unit simplex. It may be shown that the radius of this ball is given by $r = 1/\sqrt{n(n-1)}$.

Note now that as long as we move a distance **less than** *r* from the center, we are <u>guaranteed</u> to stay within the unit simplex. Now, recall that the region determined by $\boldsymbol{P_k y}=\boldsymbol{b}$ is the intersection of the region $\boldsymbol{AD_k y}=\boldsymbol{0}$ with the unit simplex $\boldsymbol{e}^{\mathrm{T}}\boldsymbol{y} = 1$. In other words, $\boldsymbol{P_k y}=\boldsymbol{b}$ is entirely within the simplex. Thus moving a distance less than *r* from the center $\boldsymbol{y}^{\boldsymbol{k}}$ ensures that we always stay within the feasible region $\boldsymbol{P_k y}=\boldsymbol{b}$.

We thus choose our new point $\boldsymbol{y}^{\boldsymbol{k+1}} = \boldsymbol{y}^{\boldsymbol{k}} + (\alpha r)\dfrac{\boldsymbol{d}^*}{\|\boldsymbol{d}^*\|}$ where $0<\alpha<1$ and this ensures

that $\boldsymbol{x}^{\boldsymbol{k+1}}=\boldsymbol{T}^{-1}_{\boldsymbol{y}^{k+1}}(\boldsymbol{y}^{\boldsymbol{k+1}})$ is in the interior of the **<u>original</u>** problem.

Putting this all together and summarizing the algorithm we have:

KARMARKAR'S INTERIOR POINT METHOD (one version...)

STEP 0: Using the appropriate change of variables, restate the original problem so that it conforms to the format of Program **P**: {Min $\boldsymbol{cx}$ | $\boldsymbol{Ax}=\boldsymbol{0}$, $\boldsymbol{e}^T\boldsymbol{x}=1$, $\boldsymbol{x}\geq\boldsymbol{0}$} where $\boldsymbol{x}$ = $(1/n\ \ 1/n\ \dots\ 1/n)$ is feasible and the optimum value is zero.

Define (1) a value for α (a value of $[(n\text{-}1)/3n]$ is suggested), and (2) a small tolerance $\varepsilon=2^{-L}$, where L is a large positive integer such as the input length (which is the number of bits required to store all the problem data). In practice, we may choose some lower bound on this number; or even more practically, some "sufficiently small" value for ε.

Set k=0, and $\boldsymbol{x}^{\boldsymbol{k}} = (1/n\ \ \ 1/n\ \dots\ 1/n)$ -- the initial solution.

STEP 1: STOP if $\boldsymbol{cx}^{\boldsymbol{k}} \leq \varepsilon$. Else go to Step 2.

STEP 2: Define the diagonal matrix $\boldsymbol{D_k}$=Diag($\boldsymbol{x}^{\boldsymbol{k}}$) and use $\mathbf{T_A}$ to transform the problem so that $\boldsymbol{y}^{\boldsymbol{k}} = \boldsymbol{T}_{\boldsymbol{x}^k}(\boldsymbol{x}^{\boldsymbol{k}})$ is at the center of transformed region, i.e., $\boldsymbol{y}^{\boldsymbol{k}} = (1/n\ \ 1/n\ \dots\ 1/n)$. Compute $\bar{\mathbf{c}}^T = \mathbf{c}^T\mathbf{D_k}$, $\boldsymbol{P_k}=\begin{bmatrix}\boldsymbol{AD_k}\\ \boldsymbol{e}^T\end{bmatrix}$, $\boldsymbol{b}=\begin{bmatrix}\boldsymbol{0}\\ 1\end{bmatrix}$ and $r = 1/\sqrt{n(n-1)}$.

STEP 3: Solve the system $(\boldsymbol{P_k}\,\boldsymbol{P_k}^{\mathrm{T}})\boldsymbol{w} = \boldsymbol{P_k}\bar{\boldsymbol{c}}$ for the vector $\boldsymbol{w}$ and compute $\boldsymbol{d} = -[\bar{\boldsymbol{c}} - \boldsymbol{P_k}^{\mathrm{T}}\boldsymbol{w}]$. Compute $\boldsymbol{y}^{\boldsymbol{k+1}} = \boldsymbol{y}^{\boldsymbol{k}} + (\alpha r)\dfrac{\boldsymbol{d}}{\|\boldsymbol{d}\|}$.

STEP 4: Obtain $\boldsymbol{x}^{\boldsymbol{k+1}}=\boldsymbol{T}^{-1}_{y^{k+1}}(\boldsymbol{y}^{\boldsymbol{k+1}}) = \dfrac{\boldsymbol{D}_k\boldsymbol{y}^{k+1}}{\boldsymbol{e}^{\mathrm{T}}\boldsymbol{D}_k\boldsymbol{y}^{k+1}}$; set k=k+1 and return to Step 1.

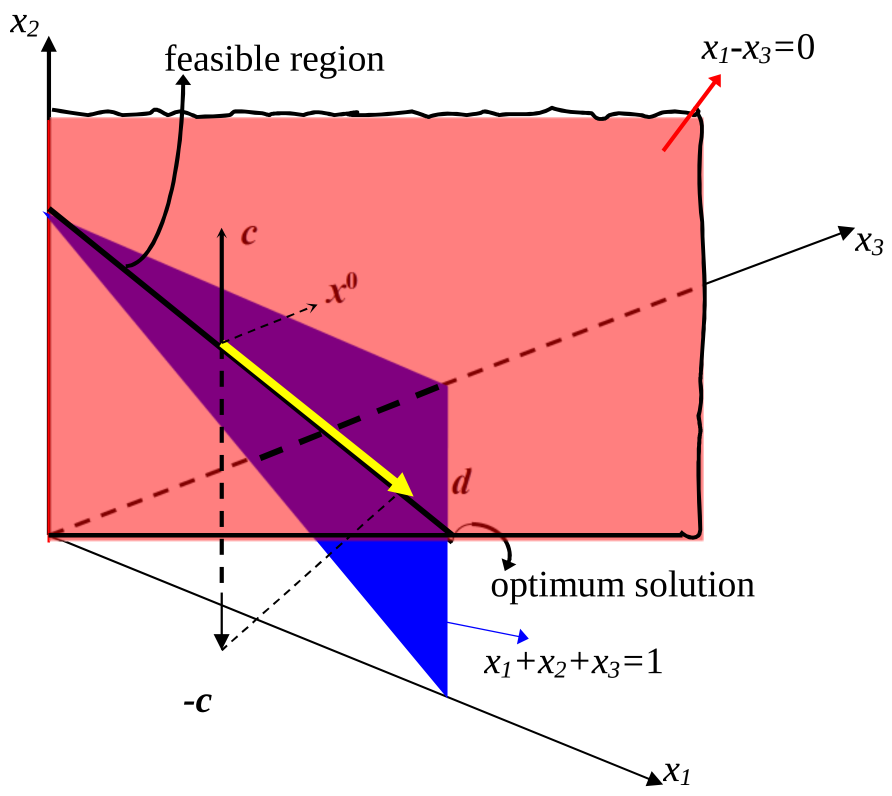
EXAMPLE: Min $Z = x_1 + 2x_2 - x_3$

st $x_1 + x_2 + x_3 = 1$

$x_1 \qquad - x_3 = 0, \qquad x_1, x_2, x_3 \geq 0.$

The feasible region for this problem is shown below.

(The optimum solution is $\boldsymbol{x}^* = [0.5\ \ 0\ \ 0.5]$ with $Z^*=0$)



Step 0 is not required since this problem is already in the proper format.

Start with $\boldsymbol{x}^0=[1/3\ \ 1/3\ \ 1/3]$.

ITERATION 0

STEP 1: $|\boldsymbol{c}^T\boldsymbol{x}^0| = 0.667 > \varepsilon$, so continue

STEP 2: $$\boldsymbol{D_0} = \begin{bmatrix} 1/3 & 0 & 0 \\ 0 & 1/3 & 0 \\ 0 & 0 & 1/3 \end{bmatrix} \qquad \bar{\mathbf{c}}^T = \mathbf{c}^T\mathbf{D}_0 = [1/3 \quad 2/3 \quad -1/3]$$

$$\boldsymbol{P_0} = \begin{bmatrix} \boldsymbol{AD}_0 \\ \boldsymbol{e}^T \end{bmatrix} = \begin{bmatrix} 1/3 & 0 & -1/3 \\ 1 & 1 & 1 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

STEP 3: $(\boldsymbol{P_0P_0}^T)\boldsymbol{w}=\boldsymbol{P_0}\bar{\boldsymbol{c}} \;\Rightarrow\; \begin{bmatrix} 2/9 & 0 \\ 0 & 3 \end{bmatrix}\boldsymbol{w} = \begin{bmatrix} 2/9 \\ 2/3 \end{bmatrix} \;\Rightarrow\; \boldsymbol{w}=\begin{bmatrix} 1 \\ 2/9 \end{bmatrix}$.

Then $\boldsymbol{d} = -(\bar{\boldsymbol{c}} - \boldsymbol{P_0}^T\boldsymbol{w}) = -\left\{\begin{bmatrix} 1/3 \\ 2/3 \\ -1/3 \end{bmatrix} - \begin{bmatrix} 5/9 \\ 2/9 \\ -1/9 \end{bmatrix}\right\} = \frac{1}{9}\begin{bmatrix} 2 \\ -4 \\ 2 \end{bmatrix}$.

and $\dfrac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \dfrac{\boldsymbol{d}}{\left(\sqrt{24}\right)/9} = \begin{bmatrix} 1/\sqrt{6} \\ -2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix}$. Therefore

$$\boldsymbol{y^1} = \boldsymbol{y^0} + \alpha r\frac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} + (2/9)*(1/\sqrt{6}) \begin{bmatrix} 1/\sqrt{6} \\ -2/\sqrt{6} \\ 1/\sqrt{6} \end{bmatrix} = \begin{bmatrix} 0.37037 \\ 0.25926 \\ 0.37037 \end{bmatrix}.$$

STEP 4: $$\boldsymbol{x^1} = \boldsymbol{T}_{\boldsymbol{y^1}}^{-1}(\boldsymbol{y^1}) = \frac{\boldsymbol{D}_0\boldsymbol{y}^1}{\boldsymbol{e}^T\boldsymbol{D}_0\boldsymbol{y}^1} = \left(\tfrac{1}{3}\right)\begin{bmatrix} 0.37037 \\ 0.25926 \\ 0.37037 \end{bmatrix}\left(\frac{1}{1/3}\right) = \begin{bmatrix} 0.37037 \\ 0.25926 \\ 0.37037 \end{bmatrix}$$

ITERATION 1

STEP 1: $|\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}^{1}| = 0.52 > \varepsilon$, so continue

STEP 2: $$\boldsymbol{D_1} = \begin{bmatrix} 0.37037 & 0 & 0 \\ 0 & 0.25926 & 0 \\ 0 & 0 & 0.37037 \end{bmatrix}$$

$$\bar{\mathbf{c}}^T = \mathbf{c}^T \mathbf{D_1} = [0.37037 \quad 0.51852 \quad 0.37037]$$

$$\boldsymbol{P_1} = \begin{bmatrix} \boldsymbol{AD}_1 \\ \boldsymbol{e}^T \end{bmatrix} = \begin{bmatrix} 0.37037 & 0 & -0.37037 \\ 1 & 1 & 1 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

STEP 3: $(\boldsymbol{P_1P_1}^{\mathrm{T}})\boldsymbol{w}=\boldsymbol{P_1}\bar{\boldsymbol{c}} \Rightarrow \begin{bmatrix} 0.27435 & 0 \\ 0 & 3 \end{bmatrix}\boldsymbol{w} = \begin{bmatrix} 0.27435 \\ 0.51852 \end{bmatrix} \Rightarrow \boldsymbol{w}=\begin{bmatrix} 1 \\ 0.17284 \end{bmatrix}.$

Then $\boldsymbol{d} = -(\bar{\boldsymbol{c}} - \boldsymbol{P_1}^{\mathrm{T}}\boldsymbol{w}) = -\left\{\begin{bmatrix} 0.37037 \\ 0.51852 \\ -0.37037 \end{bmatrix} - \begin{bmatrix} 0.54321 \\ 0.17284 \\ -0.19753 \end{bmatrix}\right\} = \begin{bmatrix} 0.17284 \\ -0.34568 \\ 0.17284 \end{bmatrix}.$

and $\dfrac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \dfrac{\boldsymbol{d}}{0.42337} = \begin{bmatrix} 0.40825 \\ -0.81650 \\ 0.40825 \end{bmatrix}.$ Therefore

$$\boldsymbol{y}^2 = \boldsymbol{y}^1 + \alpha r \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} + (2/9)*(1/\sqrt{6}) \begin{bmatrix} 0.40825 \\ -0.81650 \\ 0.40825 \end{bmatrix} = \begin{bmatrix} 0.37037 \\ 0.25926 \\ 0.37037 \end{bmatrix}.$$

STEP 4: $\boldsymbol{x}^2 = \boldsymbol{T}_{y^2}^{-1}(\boldsymbol{y}^2) = \dfrac{\boldsymbol{D}_0\boldsymbol{y}^1}{\boldsymbol{e}^{\mathrm{T}}\boldsymbol{D}_0\boldsymbol{y}^1} = \begin{bmatrix} 0.13717 \\ 0.06722 \\ 0.13717 \end{bmatrix}\left(\dfrac{1}{0.34156}\right) = \begin{bmatrix} 0.4016 \\ 0.1968 \\ 0.4016 \end{bmatrix}.$

ITERATION 2

STEP 1: $|\boldsymbol{c}^{T}\boldsymbol{x}^{2}| = 0.3936 > \varepsilon$, so continue

STEP 2: $$\boldsymbol{D}_2 = \begin{bmatrix} 0.4016 & 0 & 0 \\ 0 & 0.1968 & 0 \\ 0 & 0 & 0.4016 \end{bmatrix}$$

$$\bar{\mathbf{c}}^T = \mathbf{c}^T \mathbf{D}_2 = [0.4016 \quad 0.1968 \quad 0.4016]$$

$$\boldsymbol{P}_2 = \begin{bmatrix} \boldsymbol{AD}_2 \\ \boldsymbol{e}^T \end{bmatrix} = \begin{bmatrix} 0.4016 & 0 & -0.4016 \\ 1 & 1 & 1 \end{bmatrix} \qquad \boldsymbol{b} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

STEP 3: $(\boldsymbol{P}_2\boldsymbol{P}_2^{T})\boldsymbol{w}=\boldsymbol{P}_2\bar{\boldsymbol{c}} \Rightarrow \begin{bmatrix} 0.3226 & 0 \\ 0 & 3 \end{bmatrix}\boldsymbol{w} = \begin{bmatrix} 0.3226 \\ 0.3936 \end{bmatrix} \Rightarrow \boldsymbol{w}=\begin{bmatrix} 1 \\ 0.1312 \end{bmatrix}.$

Then $\boldsymbol{d} = -(\bar{\boldsymbol{c}} - \boldsymbol{P}_2^{T}\boldsymbol{w}) = -\left\{\begin{bmatrix} 0.4016 \\ 0.3936 \\ -0.4016 \end{bmatrix} - \begin{bmatrix} 0.5328 \\ 0.1312 \\ -0.2704 \end{bmatrix}\right\} = \begin{bmatrix} 0.1312 \\ -0.2624 \\ 0.1312 \end{bmatrix}.$

and $\dfrac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \dfrac{\boldsymbol{d}}{0.32137} = \begin{bmatrix} 0.40825 \\ -0.81650 \\ 0.40825 \end{bmatrix}.$ Therefore

$$\boldsymbol{y}^3 = \boldsymbol{y}^2 + \alpha r \frac{\boldsymbol{d}}{\|\boldsymbol{d}\|} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} + (2/9)*(1/\sqrt{6}) \begin{bmatrix} 0.40825 \\ -0.81650 \\ 0.40825 \end{bmatrix} = \begin{bmatrix} 0.37037 \\ 0.25926 \\ 0.37037 \end{bmatrix}.$$

STEP 4: $\boldsymbol{x}^3 = \boldsymbol{T}_{\boldsymbol{y}^3}^{-1}(\boldsymbol{y}^3) = \dfrac{\boldsymbol{D}_2\boldsymbol{y}^3}{\boldsymbol{e}^{\mathrm{T}}\boldsymbol{D}_2\boldsymbol{y}^3} = \left(1/3\right)\begin{bmatrix} 0.14874 \\ 0.05102 \\ 0.14874 \end{bmatrix}\left(\dfrac{1}{0.3485}\right) = \begin{bmatrix} 0.4268 \\ 0.1464 \\ 0.4268 \end{bmatrix}$

Note that we are (very slowly !) converging to the optimum solution

$\boldsymbol{x}^*= [0.5 \quad 0 \quad 0.5]$...

IMPLEMENTATION

There are a number of issues that arise with respect to the implementation of interior point methods. These include the following:

1) How do we find a strictly positive primal-dual feasible point for path following methods?
2) How do we handle simple bounds on variables?
3) How about sensitivity analysis?
4) Since we never *actually* hit the boundary with these methods, how do we move from the point at which the method terminates to the actual optimum (which of course, is an extreme point ***on the*** boundary...)?
5) What are computational efficiencies that can be derived?

Although all of these questions have been explored at length, we will examine only the last two questions here

PURIFICATION

Question 4 relates to the fact that we never actually **reach** the optimum (although we can get arbitrarily close...) since we always remain in the interior with all the methods we have seen. In a "purification" or "crossover" scheme we start with the final iterate from the method and find an exact extreme point solution with an objective value that is at least as good as the one at the last iterate.

In fact, since convergence becomes slower as we get close to the optimum and one iteration of an interior point method typically takes much more effort than one of the simplex algorithm, one approach suggested is to use purification to find an extreme when we are getting close to the optimum, and then switch over to the simplex method to chug through the last few extreme points and get to the optimal one!

Recall that at each extreme point there are at least as many linearly independent, binding constraints as the number of decision variables. Thus, we need to find a point where there are at least n linearly independent binding constraints (out of the $m+n$ original constraints in the problem: m functional plus n nonnegativity) and the objective is at least as good as that at the current point. Suppose we have $p<n$ binding constraints at $\boldsymbol{x^k}$. Then by the definition of linear independence, there exists a vector $\boldsymbol{d}\neq\boldsymbol{0}$ in the null space of the binding constraints. That is, suppose $g_1, g_2,..., g_p$ are binding where $p<n$, then we find a solution $\boldsymbol{z}$ to the system of equations $g_1=0, g_2=0,..., g_p=0$. If $\mathbf{c}^{\mathrm{T}}\boldsymbol{z} <0$ then we move from $\boldsymbol{x^k}$ along the direction $\boldsymbol{d=z}$; else we move along $\boldsymbol{d=-z}$. The distance α moved to the new point $\boldsymbol{x^{k+1}}$ is such that further movement is blocked by some other constraint becoming binding - this is bound to happen as long as the problem is bounded. At the new point $\boldsymbol{x^{k+1}}$ we have $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x^{k+1}} =\boldsymbol{c}^{\mathrm{T}}(\boldsymbol{x} +\alpha\boldsymbol{d})\leq\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}$ (since $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{d}$ is always nonpositive by the way we have chosen $\boldsymbol{d}$). Repeating this process, we eventually get to a basic feasible solution $\boldsymbol{x}^*$ with $\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x^*} \leq\boldsymbol{c}^{\mathrm{T}}\boldsymbol{x^k}$.

COMPUTATIONAL ISSUES

An interesting feature of interior point methods is that the number of iteration required to solve an LP is relatively insensitive to the size of a problem and for larger problems, is usually much smaller than with the simplex method. However, the time required for one iteration is dependent on the problem size, and with larger problems is usually much more than the time for a simplex iteration. Thus the efficiency of these methods will depend heavily on how efficiently we can implement each iteration.

In all interior point methods, the bulk of the work at each iteration is in solving a system of linear equations. Typically these equations take the form

$$(\boldsymbol{AD})(\boldsymbol{AD})^{\mathrm{T}}\boldsymbol{w}=\boldsymbol{ADc}, \text{ i.e., } (\boldsymbol{AD}^{2}\boldsymbol{A}^{T})\boldsymbol{w} = \boldsymbol{r} \qquad (\dagger)$$

where $\boldsymbol{w}$ is a direction vector that we are solving for, $\boldsymbol{r}$ is some modified right-hand-side vector, and $\boldsymbol{D}^2$ is a diagonal matrix that changes from one iteration to the next. By some accounts, this step of forming the matrix $\boldsymbol{AD}^2\boldsymbol{A}^{\mathrm{T}}$ and solving the resulting system takes almost 90% of the computational time.

This system is common to all interior point methods; specific definitions of $\boldsymbol{D}$ and $\boldsymbol{r}$ may vary from one to the other. For example, in Karmarkar's original method this is embodied in Step 3 (refer to page 45); in the primal affine scaling algorithm (refer to page 11) it is in Step 1, and in the path-following approach (refer to page 30) it is in Step 1.

The reason why barrier type methods (derived from nonlinear programming) could not compete with the simplex algorithm is that the numerical linear algebra for symmetric, positive-definite matrices (such as $\boldsymbol{AD^2A^T}$) was fully developed only in the last fifteen years or so - mainly for applications involving large finite-element models. Karmarkar's method spurred a renewed interest in non-traditional algorithms and the developments in numerical computing have been incorporated into powerful new interior point algorithms. Even now, the choice of correct parameter values and pre-processing the problem data (including such simple things as a reordering of the rows of $\boldsymbol{A}$) can have a tremendous influence on problem solution times.

There are three competitive approaches to solving the system (†):

(1) $\boldsymbol{Q}$-$\boldsymbol{R}$ factorization of $\boldsymbol{DA}^{\mathrm{T}}$,

(2) Cholesky factorization of $\boldsymbol{AD^2A}^{\mathrm{T}}$, and

(3) the use of a conjugate gradient algorithm with preconditioning.

We will briefly examine each, although the Cholesky factorization seems to be the most popular approach.

In **$\boldsymbol{Q}$-$\boldsymbol{R}$ factorization**, the matrix $\boldsymbol{DA}^{\mathrm{T}}$ is expressed as $\boldsymbol{QR}$ where $\boldsymbol{Q}$ is an $n \times n$ orthonormal matrix (i.e., $\boldsymbol{QQ}^{\mathrm{T}} = \boldsymbol{I}$) and $\boldsymbol{R}$ is an upper triangular matrix. Thus (†) becomes $[(\boldsymbol{QR})^{\mathrm{T}}\boldsymbol{QR}]\boldsymbol{w} = (\boldsymbol{R}^{\mathrm{T}}\boldsymbol{R})\boldsymbol{w} = \boldsymbol{r}$. This is solved by solving two triangular systems; first $\boldsymbol{R}^{\mathrm{T}}\boldsymbol{y}=\boldsymbol{r}$ for $\boldsymbol{y}$ via forward substitution, and then $\boldsymbol{Rw}=\boldsymbol{y}$ for $\boldsymbol{w}$ via back substitution.

The **conjugate gradient method** is a nonlinear programming algorithm, with the number of iterations depending on the number of distinct eigenvalues that $\boldsymbol{AD}^{2}\boldsymbol{A}^{\mathrm{T}}$ possesses - this number can be reduced by a so-called "preconditioning" procedure.

Cholesky factorization is a numerically stable procedure which has been very well developed and is probably the most popular approach to solving (†). Suppose we denote the matrix $\boldsymbol{AD}^{2}\boldsymbol{A}^{\mathrm{T}}$ by $\boldsymbol{M}$. Then it can be shown that $\boldsymbol{M}$ is symmetric and positive definite ($\boldsymbol{z}^{\mathrm{T}}\boldsymbol{Mz}>0$ for $\boldsymbol{z}\neq\boldsymbol{0}$) square matrix of order $m\times m$. Such matrices may be factorized in the form

$$\boldsymbol{M} = \boldsymbol{LL}^{\mathrm{T}}$$

where $\boldsymbol{L}$ is an $m \times m$ lower triangular matrix known as the Cholesky factor of matrix $\boldsymbol{M}$. If we could compute $\boldsymbol{L}$ efficiently, then our system (†) reduces to solving $(\boldsymbol{LL}^{\mathrm{T}})\boldsymbol{w}=\boldsymbol{r}$. Thus we could first solve $\boldsymbol{Ly}=\boldsymbol{r}$ via forward substitution for $\boldsymbol{y}$, and then solve $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{w}=\boldsymbol{y}$ for $\boldsymbol{w}$ via back substitution.

<u>QUESTION</u>: How do we find the Cholesky factor $\boldsymbol{L}$?

Consider the (i,j)th element of $\boldsymbol{M}$ - say m_{ij}. We know that this is the inner product of the ith row of $\boldsymbol{L}$ and the jth column of $\boldsymbol{L}^{\mathrm{T}}$. But the latter is the jth row of $\mathbf{L}$. Thus

$$m_{ij} = \sum_{k=1}^{m} l_{ik} l_{jk} \ .$$

However, ***L*** is lower triangular so that l_{ik} (or l_{jk}) = 0 if $i<k$ (or $j<k$).

Thus $m_{ij} = \sum_{k=1}^{p} l_{ik} l_{jk}$ where p=Min(i,j) (1)

$m_{11} = l_{11}l_{11}$, $m_{12} = l_{11}l_{21}$, ... , $m_{1m} = l_{11}l_{m1}$,

$m_{21} = l_{21}l_{11}$, $m_{22} = l_{21}l_{21}+l_{22}l_{22}$, $m_{23} = l_{21}l_{31}+l_{22}l_{32}$, etc.

$m_{31} = l_{31}l_{11}$, $m_{32} = l_{31}l_{21}+l_{32}l_{22}$, $m_{3} = l_{31}l_{31}+l_{32}l_{32}+l_{33}l_{33}$, etc.

First, note that

$$m_{ii} = (l_{i1})^2 + (l_{i2})^2 + \ldots + (l_{ii})^2 \quad (2)$$

Therefore, for Column 1 of ***L***, we see that

$$l_{11} = (m_{11})^{0.5}$$

Next, note that $m_{i1} = l_{i1}l_{ii}$. Thus, knowing l_{ii}, we may then find

$$l_{i1} = m_{i1}/l_{11} \quad \text{for } i=2,\ldots,m.$$

Thus Column 1 of ***L*** is easily computed.

Next, since Column 1 is fully determined, we now know l_{i1} for i=1,...,n

Thus for Columns j=2,3,...,m we may first use (2) to compute

$$l_{jj} = \sqrt{m_{jj} - \sum_{k=1}^{j-1} l_{jk}^2}$$

and then use (1) to compute

$$l_{ij} = \left(m_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk} \right) \Big/ l_{jj}$$

Thus this is a strategy where the columns of ***L*** are computed one at a time. However, a different strategy of row-wise computation can also be used. Another option is the use of vector-processing or parallel computing. Yet another approach is the use of recursive functions. The important point here is that for a serious implementation, one should study the capabilities of the hardware and software to be used and try and select the most effective computational procedure for finding the Cholesky factor.

Another area of research is the development of **block** Cholesky factorization schemes where the matrix ***M*** is partitioned into a total of p^2 blocks, each of order r (where $m=pr$) as follows:

$$\boldsymbol{M} = \begin{bmatrix} M_{11} & M_{12} & \dots & M_{1p} \\ M_{21} & . & & : \\ & & . & : \\ : & & & \\ M_{p1} & \dots & \dots & M_{pp} \end{bmatrix}$$

The Cholesky factor can accordingly be partitioned as

$$\boldsymbol{L} = \begin{bmatrix} L_{11} & 0 & \dots & 0 \\ L_{21} & . & & : \\ & & . & : \\ : & & & \\ L_{p1} & \dots & \dots & L_{pp} \end{bmatrix}$$

Since $\boldsymbol{M}=\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}}$, relationships very similar to the ones for the regular factorization can be obtained with sub-matrices taking the place of individual entries. Thus Cholesky factors for the individual submatrices can be evaluated using the method described earlier, and then these can be put together to form the overall Cholesky factor for ***M***. Again, this is an active research area.

SPARSE CHOLESKY FACTORIZATION: One of the most important points to note with Cholesky factorization is that it is of tremendous computational advantage to obtain relatively sparse factors, i.e., ***L*** should have as few non-zeros as possible in the lower triangular area. This reduces the time required to compute ***L***. Furthermore, recall that we solve $(\boldsymbol{L}\boldsymbol{L}^{\mathrm{T}})\boldsymbol{w}=\boldsymbol{r}$ by solving $\boldsymbol{L}\boldsymbol{y}=\boldsymbol{r}$ via forward substitution for ***y***, and $\boldsymbol{L}^{\mathrm{T}}\boldsymbol{w}=\boldsymbol{y}$ via back substitution for ***w***; fewer non-zeros in ***L*** will speed this up. ***L*** can be kept from becoming too dense by a suitable re-ordering of the rows and columns of ***M***. This of course doesn't change anything fundamental, since it is just equivalent to reordering the equations and the variables.

As an example consider the following example from the book by George and Liu:

$$\boldsymbol{M}=\begin{bmatrix} 4 & 1 & 2 & 0.5 & 2 \\ 1 & 0.5 & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 0.5 & 0 & 0 & 0.625 & 0 \\ 2 & 0 & 0 & 0 & 16 \end{bmatrix} \text{ has } \boldsymbol{L}=\begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0.5 & 0.5 & 0 & 0 & 0 \\ 1 & -1 & 1 & 0 & 0 \\ 0.25 & -0.25 & -0.5 & 0.5 & 0 \\ 1 & -1 & -2 & -3 & 1 \end{bmatrix}$$

Thus a relatively sparse matrix ***M*** has a very dense factor ***L*** with all 15 lower triangular entries being $\neq 0$. On the other hand, suppose we first reorder the rows and then the columns of ***M*** to get

$$\begin{bmatrix} 2 & 0 & 0 & 0 & 16 \\ 0.5 & 0 & 0 & 0.625 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ 1 & 0.5 & 0 & 0 & 0 \\ 4 & 1 & 2 & 0.5 & 2 \end{bmatrix} \text{ and then } \begin{bmatrix} 16 & 0 & 0 & 0 & 2 \\ 0 & 0.625 & 0 & 0 & 0.5 \\ 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & 0.5 & 1 \\ 2 & 0.5 & 2 & 1 & 4 \end{bmatrix}.$$

All we have done is changed the numbering of the equations and variables. However this ***M*** has a Cholesky factor ***L*** given by

$$\boldsymbol{L} = \begin{bmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 0.791 & 0 & 0 & 0 \\ 0 & 0 & 1.73 & 0 & 0 \\ 0 & 0 & 0 & 0.707 & 0 \\ 0.5 & 0.632 & 1.15 & 1.41 & 0.129 \end{bmatrix}$$ which is clearly much more sparse than the

factor on the previous page (with 9 as opposed to 15 non-zero entries)!

Thus reordering schemes applied to ***M*** maintain sparse Cholesky factors by minimizing the "fill-in" in ***L***. This too is a well-researched area with several efficient heuristics (the problem of minimizing the non-zeros in ***L*** is NP-complete). Marsten et al. (Interfaces 20:4, 1990) give an example of a large problem, where the number of nonzeros in ***L*** is reduced from 39,674 (61% dense) to 3,252 (5% dense) by the "minimum-degree row reordering" heuristic.

SYMBOLIC CHOLESKY FACTORIZATION

Recall that the matrix $\boldsymbol{M}=\boldsymbol{ADA}^{\mathrm{T}}$ is different each time the system is solved, since ***D*** changes each time. Thus if were reordering the rows of ***M*** to reduce fill-in, it would seem like a tedious task to do this each time! Fortunately, it turns out that even though ***D*** (and hence ***M***) changes at each iteration, the pattern of non-zeros in the Cholesky factor (i.e., their positions) remains unchanged. Thus we first choose ***D***=*I* and take $\boldsymbol{AA}^{\mathrm{T}}$ and factorize it in the most efficient way possible by a re-ordering scheme. If we now memorize the positions of the non-zeros in this factor, then at all subsequent iterations, the factors computed will have the non-zeros appearing at the same positions. Thus we need only find their numerical values. This initial factorization is thus referred to as a *symbolic* Cholesky factorization.