Algorithms for Linear Programming Problems

\[ \min_{x} c^T x \]
\[ Ax = b \]
\[ x \geq 0 \]

The Plan

There are two main approaches for solving linear programming problems:

- The Simplex method (Dantzig and others, 1940’s) (Predecessors such as Motzkin)
- Interior Point methods (Karmarkar, 1980’s) (Predecessors such as Fiacco and McCormick)

The geometry of the methods

The algebra of the Simplex Method

The algebra of Interior Point Methods

Our starting point: Affine scaling algorithm.

This is not the best method, but it will help us fix ideas.

The basic intuition
• Suppose we are at an interior point of the feasible set

• Picture.

• Consider the steepest descent step.

• This step doesn’t make much progress unless our starting point is central.

• So we’ll change the coordinate system so that the current point is always central.

Some facts we need

• The matrix $P = I - A^T (AA^T)^{-1} A$ is a projector into the nullspace of $A$:

\[
Az = APy = Ay - AA^T (AA^T)^{-1} Ay = Ay - Ay = 0.
\]

Therefore, for any vector $y$, $Py$ is a feasible direction.

• If we want the steepest descent direction, we need to solve

\[
\min_{\|Py\| = 1} c^T (x + Py),
\]

and the solution to this is the same as the solution to

\[
\min_{\|Py\| = 1} c^T Py.
\]

Therefore, the steepest descent direction is a vector of length 1 in the direction $-Pc$.

• Central means being approximately equidistant from all of the bounds $x \geq 0$. Therefore, $x$ is central if $x = \beta e$ for some positive scalar $\beta$.

• A convenient notation: If we make a diagonal matrix out of a vector, we will denote the matrix by using the same letter of the alphabet, but its upper case form. For example, in Matlab notation,

\[
X = \text{diag}(x)
\]
The object of the game: Rewrite the linear program so that the current guess $x^{(k)}$ is transformed to $e$. This is an affine (or linear) scaling:

$$e = (X^{(k)})^{-1}x^{(k)}$$

so our new variables are

$$\bar{x} = (X^{(k)})^{-1}x.$$

How does this change the rest of the problem?

$$c^T x = c^T X^{(k)} \bar{x} \equiv e^T \bar{x},$$

$$b = A x = AX^{(k)} \bar{x} \equiv \bar{A} \bar{x},$$

$$x \geq 0 \leftrightarrow \bar{x} \geq 0,$$

current iterate $x^{(k)} \leftrightarrow$ current iterate $\bar{x}^{(k)} = e$

The resulting problem

$$\min_{\bar{x}} e^T \bar{x}$$

$$\bar{A} \bar{x} = b$$

$$\bar{x} \geq 0$$

Now we can find the steepest descent direction in this transformed space:

$$\Delta \bar{x} \equiv \bar{p} = -\bar{P} \bar{e} = -(I - \bar{A}^T \bar{A})^{-1} \bar{A} \bar{e}$$

and we can take a step

$$\bar{x} = e + \alpha \Delta \bar{x}$$

where $\alpha$ is chosen so that $\bar{x} \geq 0$.

The point $\bar{x}$ is no longer central, so we return to the original coordinate system. In terms of $x$, our step is

$$\Delta x = X^{(k)} \Delta \bar{x} = -X^{(k)}(I - X^{(k)T}A^TAX^{(k)T}A^T)^{-1}AX^{(k)}c,$$
and

\[ x^{(k+1)} = x^{(k)} + \alpha \Delta x. \]

**A heuristic:** We don’t want to hit a boundary, since then we can’t make an affine transformation to a central point, so we step 90 (or 99)% of the way to the boundary:

\[
\alpha_{\text{max}} = \min_{\Delta x_i < 0} \frac{-x_i}{\Delta x_i}, \\
\alpha = 9\alpha_{\text{max}}.
\]

Then we can repeat the process of making the transformation and taking a step.

**Issues**

- The primary computational task is the projection. We have seen how to do this with QR factors.
- Slightly more complicated methods work better in practice. We’ll build toward these algorithms.

**5 equivalent problems**

We know **lots** of different ways to write our linear programming problem, and these different variants will yield insights and algorithms.

**Problem 1:** The primal

\[
\min_x c^T x \\
Ax = b \\
x \geq 0
\]

**Problem 2:** The dual

\[
\max_y b^T y \\
A^T y \leq c
\]
(Notice that I used the variable \( y \) instead of \( \lambda \), to match the notation in the book.)

This can be written with a slack variable as

\[
\max_y b^T y \\
A^T y + z = c \\
z \geq 0
\]

**Problem 3:** Log-Barrier formulation

\[
\min_x B(x, \mu) \\
Ax = b
\]

where

\[
B(x, \mu) = c^T x - \mu \sum_{i=1}^n \ln x_i
\]

**Problem 4:** Optimality conditions for the Log-Barrier formulation

The Lagrangian for the Log-Barrier problem is

\[
L_B = c^T x - \mu \sum_{i=1}^n \ln x_i - y^T (Ax - b)
\]

so we need

\[
c - \mu X^{-1} e - A^T y = 0 \\
Ax - b = 0
\]

**We take these conditions to define the central path:** The central path is defined by \( x(\mu), y(\mu), z(\mu) \), where

\[
Ax = b \\
x > 0 \\
A^T y + z = c \\
z > 0 \\
Xz = \mu e.
\]
The last condition defines what we mean by centering: we keep both $x$ and $z$ bounded away from zero this way.

**Problem 5:** Optimality conditions for linear programming

$$Ax = b$$
$$x \geq 0$$
$$A^T y + z = c$$
$$z \geq 0$$
$$x^T z = 0$$

**Important Observation 1:** These match the central path definition except that for the central path,

$$x^T z = c^T X z = \mu e^T e > 0.$$  
(And except for nonnegativity rather than positivity.)

But it is clear that we achieve optimality by driving $\mu$ to zero!

**Important Observation 2:** This relation between the central path and the optimality conditions gives us a hint about how to set a stopping criterion:

If $Ax = b$, $x \geq 0$, $A^T y + z = c$, and $z \geq 0$, then

$$x^T c - y^T b = x^T (A^T y + z) - y^T A x = x^T z \geq 0.$$  
Recall that the duality gap for linear programming is zero, so if $(x^*, y^*)$ is optimal,

$$c^T x^* = b^T y^*,$$

so, since $c^T x = b^T y + x^T z$,

$$0 \leq c^T x - c^T x^* = x^T z + b^T y - b^T y^*.$$  
By optimality, $b^T y - b^T y^* \leq 0$, so

$$0 \leq c^T x - c^T x^* \leq x^T z.$$  
So when $x^T z$ is small enough, we can stop!
The computational formulation

From Problem 5, we see that we need to solve a system of nonlinear equations

\[
\begin{align*}
Xz - \mu e &= 0, \\
Ax - b &= 0, \\
A^T y + z - c &= 0
\end{align*}
\]

with \( x \geq 0, \ z \geq 0 \), and we want to also drive \( \mu \) to zero.

Suppose we use Newton's method to solve this system. The Jacobian matrix is

\[
J = \begin{bmatrix}
Z & 0 & X \\
A & 0 & 0 \\
0 & A^T & I
\end{bmatrix}
\]

so the Newton step is

\[
\begin{bmatrix}
Z & 0 & X \\
A & 0 & 0 \\
0 & A^T & I
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta z
\end{bmatrix} = \begin{bmatrix}
\mu e - Xz \\
b - Ax \\
c - A^T y - z
\end{bmatrix}.
\]

We need to solve this linear system using our favorite method: LU factorization, an iterative method, etc.

If we do factor the matrix \( J \), which is expensive, then we can get multiple uses from it by using an algorithm called predictor-corrector (Mehrotra). Solve the linear system as written, and then re-solve it, updating the right-hand side by evaluating the first component at

\[
\begin{align*}
x + \Delta x, \\
y + \Delta y, \\
z + \Delta z.
\end{align*}
\]

The reduced system

So far we have the system

\[
\begin{bmatrix}
X^{-1}Z & 0 & I \\
A & 0 & 0 \\
0 & A^T & I
\end{bmatrix}
\begin{bmatrix}
\Delta x \\
\Delta y \\
\Delta z
\end{bmatrix} = \begin{bmatrix}
\mu X^{-1} e - X^{-1} Xz \\
b - Ax \\
c - A^T y - z
\end{bmatrix},
\]

where we have multiplied the first block equation by \( X^{-1} \).
We can also work with the reduced system obtaining by using the 3rd block equation to solve for $\Delta z$:

$$\Delta z = c - A^T y - z - A^T \Delta y.$$ 

Then our system becomes

$$\begin{bmatrix} X^{-1} Z & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = \begin{bmatrix} \mu X^{-1} e - z - c + A^T y + z \\ b - Ax \end{bmatrix}.$$ 

This is called the KKT system after Karush, Kuhn, and Tucker, who made a lot of contributions toward deriving the optimality conditions for nonlinear programming.

Reducing further

Let $D^2 = X^{-1} Z$ and note that this is a positive definite diagonal matrix.

If we take $-AD^{-2}$ times the first equation and add the second equation, we obtain

$$AD^{-2} A^T \Delta y = b - Ax - AD^{-2}(\mu X^{-1} e - c + A^T y).$$

This is the system of normal equations.

So, to determine the Newton direction, we have the choice of solving the big system, the medium system, or the small system.

Implementation issues

- The KKT matrix is symmetric but indefinite.
- The normal equations matrix is symmetric positive definite.
- Whatever system we choose, the sparsity pattern remains the same for every iteration. (This is in contrast to the simplex algorithm for linear programming, in which the basis changes each iteration.)
- We don’t need to solve the system to high accuracy; we only need a descent direction. But for superlinear convergence, a reasonably good direction is needed.
- The KKT system becomes very ill-conditioned as we approach the solution, because some components of $X^{-1}$ get very large. Even so, the ill-conditioning does not prevent us from computing a good search direction. This mystery was unraveled by Margaret Wright.
• Primal-Dual methods perform better than Primal methods or Dual methods. This will probably remain true.
• The current most popular algorithm is Predictor-Corrector. This may change.

The basis of the Complexity Theory

• The problem size is the number of bits needed to store the problem on a machine. This is finite if all of the data is rational, so we will make this assumption, specifying each entry in $A$, $b$, and $c$ as the ratio of two integers. We’ll suppose that it takes $L$ bits to store these entries along with $m$ and $n$.
• We note that $m \leq n$, so $m$ never appears in the complexity bounds.
• Suppose we know the active constraints at the optimal solution $x^*$. Then $x^*$ can be expressed as the solution to the linear system of equations defined by these constraints. Since the coefficient matrix and right-hand side are rational, so is $x^*$, and it has an exact representation in a number of bits bounded by a polynomial in the number of bits of data. In fact, the nonzero components are bounded below by $\varepsilon \cdot 2^{-L}$.
• Thus we have motivation for allowing an algorithm to “round-off” to the closest rational number that is representable within our bit bound, and complexity proofs need to show that this does not hurt convergence.
• And we know that at some stage we can terminate the iteration and set all of the very small components of the solution vector to zero, using a linear system to solve for the exact values of the others.
• Each iteration will take time polynomial in $L$, so we just need to make sure that the number of iterations is bounded by a polynomial in $L$.

The basic algorithm

Given $(x^{(0)}, y^{(0)}, z^{(0)})$ satisfying

\[
\begin{align*}
Ax^{(0)} &= b \\
A^T y^{(0)} + z^{(0)} &= c \\
x^{(0)} &> 0 \\
z^{(0)} &> 0
\end{align*}
\]

For $k = 0, 1, \ldots$, until $x^{(k)}Tz^{(k)}$ small enough,
• Solve
\[
\begin{bmatrix}
Z^{(k)} & 0 & X^{(k)} \\
A & 0 & 0 \\
0 & A^T & I
\end{bmatrix}
\begin{bmatrix}
\Delta x^{(k)} \\
\Delta y^{(k)} \\
\Delta z^{(k)}
\end{bmatrix}
= \begin{bmatrix}
-X^{(k)} z^{(k)} + \sigma_k \mu_k \epsilon \\
0 \\
0
\end{bmatrix}
\]
where \( \sigma_k \in [0, 1] \) and \( \mu_k = x^{(k)T} z^{(k)}/n \).

• Set
\[
\begin{bmatrix}
x^{(k+1)} \\
y^{(k+1)} \\
z^{(k+1)}
\end{bmatrix}
= \begin{bmatrix}
x^{(k)} \\
y^{(k)} \\
z^{(k)}
\end{bmatrix}
+ \alpha_k \begin{bmatrix}
\Delta x^{(k)} \\
\Delta y^{(k)} \\
\Delta z^{(k)}
\end{bmatrix}
\]
choosing \( \alpha_k \) so that \( x^{(k+1)} > 0 \) and \( z^{(k+1)} > 0 \).

Note: For nonzero \( \sigma \), if we set \( \alpha_k = 1 \), then we will have
\[
x^{(k+1)} z^{(k+1)} \approx \sigma \mu_k,
\]
so we are targeting the particular point on the central path corresponding to the parameter \( \sigma \mu_k \). If we set \( \sigma = 0 \), we are targeting the point corresponding to 0, i.e., the solution to the LP.

Some Variations

Potential Reduction Methods

• Goal: reduce a potential function, rather than follow the central path.

• Measuring progress: The value \( \phi \) should decrease sufficiently fast, where \( \phi \) is a potential function. One very useful one (Tanabe-Todd-Ye):
\[
\phi_{\rho}(x, z) = \rho \log x^T z - \sum_{i=1}^{n} \log(x_i z_i)
\]
where \( \rho > n \).

• Implementation:
  – Choose \( \sigma_k = n/\rho \).
  – Choose \( \alpha_k \) using a line search for the function \( \phi \) with an upper bound on \( \alpha \) equal to \( \alpha_{max} = \) the maximal step that hits the boundary.

• Convergence result: If \( \rho = n + \sqrt{n} \), the bound on the number of iterations is \( O(\sqrt{n} \log(1/\epsilon)) \).

• Practicalities:
Choose a larger value like $\rho = 10n$.

Line search need not be exact: see if $0.99 \alpha_{\text{max}}$ (or similar values) yield a prescribed constant decrease in $\phi$.

---

**Path Following Methods**

- **Goal:** try to stay in a neighborhood of the central path, and thus avoid points that are too close to the boundary where $x_i = 0$ or $z_i = 0$.

- **Measuring progress:** The value $\mu$ should decrease, so that we move closer to a KKT point, one that satisfies the optimality conditions for the LP.

- **Classes of methods:**
  - **Short-step methods** choose $\sigma$ close to 1 and are able to set $\alpha_k = 1$ without straying far from the central path.
  - **Long-step methods** choose smaller values of $\sigma$ and thus must choose an $\alpha_k$ so that $x_i^{(k+1)} z_i^{(k+1)} \geq \gamma \mu_k$, where $\gamma$ is chosen between 0 and 1. (A typical $\gamma$ is $10^{-3}$.)

- **Predictor-corrector methods** take
  
  * a predictor step with $\sigma = 0$ and $\alpha_k$ chosen to keep $|Xz - \mu e|_2 \leq \theta \mu$ (typical $\theta$ is 0.5),
  
  * followed by a corrector step with $\sigma = 1$ and $\alpha = 1$.
  
  The predictor step is a “long step”, and the “short” corrector step pulls the iterate back toward the central path without significantly changing the $\mu$ value achieved by the predictor.

- **Convergence analysis:** The short-step and predictor-corrector algorithms can be shown to terminate in $O(\sqrt{n} \log 1/\epsilon)$ iterations, and $\mu_k$ converges to zero superlinearly for the predictor-corrector algorithm. The bound on the long-step algorithm is $O(n \log 1/\epsilon)$, but it behaves well in practice.

---

**Dealing with infeasible initial points**

- It is easy to choose an $x > 0$ and a $z > 0$.

- It is hard to satisfy the equality constraints $Ax = b$ and $A^Ty + z = c$.

So infeasible IPMs replace the step equation by

$$
\begin{bmatrix}
Z^{(k)} & 0 & X^{(k)} \\
A & 0 & 0 \\
0 & A^T & I
\end{bmatrix}
\begin{bmatrix}
\Delta x^{(k)} \\
\Delta y^{(k)} \\
\Delta z^{(k)}
\end{bmatrix}
= 
\begin{bmatrix}
-X^{(k)}z^{(k)} + \sigma_k \mu_k e \\
c - A^T y^{(k)} - z^{(k)} \\
b - Ax^{(k)}
\end{bmatrix}
$$
The convergence analysis is not as pretty: $O(n^2 \log 1/\epsilon)$ for a "medium-step" version, but the algorithms are much more convenient to use!

Reference