AMSC 607 / CMSC 764 Advanced Numerical Optimization

Fall 2008

UNIT 3: Constrained Optimization

PART 4: Introduction to Interior Point Methods

Dianne P. O'Leary

©2008

Interior Point Methods

We'll discuss linear programming first, followed by three nonlinear problems.

Algorithms for Linear Programming Problems

$$\min_{\mathbf{x}} \mathbf{c}^T \mathbf{x}$$

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

$$\mathbf{x} \geq \mathbf{0}$$

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

See the feasible direction notes.

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 $\mathbf{P} = \mathbf{I} - \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}$ is a projector into the nullspace of $\mathbf{A}$:

  If we have a vector $\mathbf{y}$, then $\mathbf{z} = \mathbf{P}\mathbf{y} = \mathbf{y} - \mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{y}$, so

  $$\mathbf{A}\mathbf{z} = \mathbf{A}\mathbf{P}\mathbf{y} = \mathbf{A}\mathbf{y} - \mathbf{A}\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{y} = \mathbf{A}\mathbf{y} - \mathbf{A}\mathbf{y} = \mathbf{0}\,.$$

  Therefore, for any vector $\mathbf{y}$, $\mathbf{P}\mathbf{y}$ is a feasible direction.

- If we want the steepest descent direction at a particular point $\mathbf{x}$, we need to solve

  $$\min_{\|\mathbf{P}\mathbf{y}\|=1} \mathbf{c}^T(\mathbf{x} + \mathbf{P}\mathbf{y})\,,$$

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

  $$\min_{\|\mathbf{P}\mathbf{y}\|=1} \mathbf{c}^T\mathbf{P}\mathbf{y}\,.$$

  Therefore, the steepest descent direction is a vector of length 1 in the direction $-\mathbf{P}\mathbf{c}$.

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

- 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 = diag(x)
  ```

Affine scaling to a central point

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

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

so our new variables are

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

How does this change the rest of the problem?

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

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

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

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

The resulting problem

$$\min_{\bar{\mathbf{x}}} \bar{\mathbf{c}}^T\bar{\mathbf{x}}$$

$$\begin{aligned} \bar{\mathbf{A}}\bar{\mathbf{x}} &= \mathbf{b} \\ \bar{\mathbf{x}} &\geq \mathbf{0} \end{aligned}$$

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

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

and we can take a step

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

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

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

$$\Delta\mathbf{x} = \mathbf{X}^{(k)}\Delta\bar{\mathbf{x}} = -\mathbf{X}^{(k)}(\mathbf{I} - \mathbf{X}^{(k)T}\mathbf{A}^T(\mathbf{A}\mathbf{X}^{(k)2}\mathbf{A}^T)^{-1}\mathbf{A}\mathbf{X}^{(k)})\mathbf{X}^{(k)}\mathbf{c}\,,$$

and

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha\Delta\mathbf{x}\,.$$

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:

$$\begin{aligned} \alpha_{max} &= \min_{\Delta x_i < 0} \frac{-x_i}{\Delta x_i} \\ \alpha &= .9\alpha_{max}. \end{aligned}$$

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.
- The affine scaling method is not the best. We'll build toward better 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

$$\min_{\mathbf{x}} \mathbf{c}^T \mathbf{x}$$

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \end{aligned}$$

Problem 2: The dual

$$\max_{\mathbf{y}} \mathbf{b}^T \mathbf{y}$$

$$\mathbf{A}^T \mathbf{y} \leq \mathbf{c}$$

(Notice that I used the variable $\mathbf{y}$ instead of $\boldsymbol{\lambda}$, to match the notation in the book.)

This can be written with a slack variable as

$$\max_{\mathbf{y}} \mathbf{b}^T \mathbf{y}$$

$$\begin{aligned} \mathbf{A}^T\mathbf{y} + \mathbf{z} &= \mathbf{c} \\ \mathbf{z} &\geq \mathbf{0} \end{aligned}$$

Problem 3: Log-Barrier formulation

$$\min_{\mathbf{x}} B(\mathbf{x}, \mu)$$

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

where

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

Problem 4: Optimality conditions for the Log-Barrier formulation

The Lagrangian for the Log-Barrier problem is

$$L_B = \mathbf{c}^T\mathbf{x} - \mu \sum_{i=1}^{n} \ln x_i - \mathbf{y}^T(\mathbf{A}\mathbf{x} - \mathbf{b})$$

so we need

$$\begin{aligned} \mathbf{c} - \mu\mathbf{X}^{-1}\mathbf{e} - \mathbf{A}^T\mathbf{y} &= \mathbf{0} \\ \mathbf{A}\mathbf{x} - \mathbf{b} &= \mathbf{0} \end{aligned}$$

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

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{b} \\ \mathbf{x} &> \mathbf{0} \\ \mathbf{A}^T\mathbf{y} + \mathbf{z} &= \mathbf{c} \\ \mathbf{z} &> \mathbf{0} \\ \mathbf{X}\mathbf{z} &= \mu\mathbf{e}\,. \end{aligned}$$

The last condition defines what we mean by centering: we keep both $\mathbf{x}$ and $\mathbf{z}$ bounded away from zero this way.

Problem 5: Optimality conditions for linear programming

$$\begin{aligned} \mathbf{Ax} &= \mathbf{b} \\ \mathbf{x} &\geq \mathbf{0} \\ \mathbf{A}^T\mathbf{y}+\mathbf{z} &= \mathbf{c} \\ \mathbf{z} &\geq \mathbf{0} \\ \mathbf{x}^T\mathbf{z} &= \mathbf{0} \end{aligned}$$

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

$$\mathbf{x}^T\mathbf{z} = \mathbf{e}^T\mathbf{Xz} = \mu\mathbf{e}^T\mathbf{e} > 0\,.$$

(And except for nonnegativity rather than positivity.)

But it is clear that we achieve optimality by driving μ 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 $\mathbf{Ax}=\mathbf{b}$, $\mathbf{x}\geq\mathbf{0}$, $\mathbf{A}^T\mathbf{y}+\mathbf{z}=\mathbf{c}$, and $\mathbf{z}\geq\mathbf{0}$, then

$$\mathbf{x}^T\mathbf{c}-\mathbf{y}^T\mathbf{b} = \mathbf{x}^T(\mathbf{A}^T\mathbf{y}+\mathbf{z}) - \mathbf{y}^T\mathbf{Ax} = \mathbf{x}^T\mathbf{z} \geq \mathbf{0}\,.$$

Recall that the duality gap for linear programming is zero, so if $(\mathbf{x}^*,\mathbf{y}^*)$ is optimal,

$$\mathbf{c}^T\mathbf{x}^* = \mathbf{b}^T\mathbf{y}^*,$$

so, since $\mathbf{c}^T\mathbf{x} = \mathbf{b}^T\mathbf{y}+\mathbf{x}^T\mathbf{z}$,

$$0 \leq \mathbf{c}^T\mathbf{x}-\mathbf{c}^T\mathbf{x}^* = \mathbf{x}^T\mathbf{z}+\mathbf{b}^T\mathbf{y}-\mathbf{b}^T\mathbf{y}^*\,.$$

By optimality, $\mathbf{b}^T\mathbf{y}-\mathbf{b}^T\mathbf{y}^*\leq 0$, so

$$0 \leq \mathbf{c}^T\mathbf{x}-\mathbf{c}^T\mathbf{x}^* \leq \mathbf{x}^T\mathbf{z}\,.$$

So when $\mathbf{x}^T\mathbf{z}$ is small enough, we can stop!

The computational formulation of IPMs

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

$$\begin{aligned} \mathbf{Xz}-\mu\mathbf{e} &= \mathbf{0}, \\ \mathbf{Ax}-\mathbf{b} &= \mathbf{0}, \\ \mathbf{A}^T\mathbf{y}+\mathbf{z}-\mathbf{c} &= \mathbf{0} \end{aligned}$$

with $\mathbf{x} \geq \mathbf{0}$, $\mathbf{z} \geq \mathbf{0}$, and we want to also drive μ to zero.

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

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

so the Newton step is

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

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

If we do factor the matrix $\mathbf{J}$, 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

$$\begin{aligned} \mathbf{x} &+ \Delta\mathbf{x}, \\ \mathbf{y} &+ \Delta\mathbf{y}, \\ \mathbf{z} &+ \Delta\mathbf{z}. \end{aligned}$$

The reduced system

So far we have the system

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

where we have multiplied the first block equation by $\mathbf{X}^{-1}$.

We can also work with the reduced system obtaining by using the 3rd block equation to solve for $\Delta\mathbf{z}$:

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

Then our system becomes

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

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

If we take $-\mathbf{A}\mathbf{D}^{-2}$ times the first equation and add the second equation, we obtain

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

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 $\mathbf{X}^{-1}$ 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 Interior Point Methods for Linear Programming

- Primal-Dual methods perform better than Primal methods or Dual methods.
- 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 $\mathbf{A}$, $\mathbf{b}$, and $\mathbf{c}$ 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 $\mathbf{x}^*$. Then $\mathbf{x}^*$ 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 $\mathbf{x}^*$, 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 $\epsilon \equiv 2^{-L}$

- 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 $(\mathbf{x}^{(0)}, \mathbf{y}^{(0)}, \mathbf{z}^{(0)})$ satisfying

$$\begin{aligned} \mathbf{A}\mathbf{x}^{(0)} &= \mathbf{b} \\ \mathbf{A}^T\mathbf{y}^{(0)} + \mathbf{z}^{(0)} &= \mathbf{c} \\ \mathbf{x}^{(0)} &> \mathbf{0} \\ \mathbf{z}^{(0)} &> \mathbf{0} \end{aligned}$$

For $k = 0, 1, \ldots,$ until $\mathbf{x}^{(k)T}\mathbf{z}^{(k)}$ small enough,

- Solve

$$\begin{bmatrix} \mathbf{Z}^{(k)} & 0 & \mathbf{X}^{(k)} \\ \mathbf{A} & 0 & 0 \\ 0 & \mathbf{A}^T & \mathbf{I} \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x}^{(k)} \\ \Delta\mathbf{y}^{(k)} \\ \Delta\mathbf{z}^{(k)} \end{bmatrix} = \begin{bmatrix} -\mathbf{X}^{(k)}\mathbf{z}^{(k)} + \sigma_k\mu_k\mathbf{e} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix}$$

  where $\sigma_k \in [0,1]$ and $\mu_k = \mathbf{x}^{(k)T}\mathbf{z}^{(k)}/n$.

- Set

$$\begin{bmatrix} \mathbf{x}^{(k+1)} \\ \mathbf{y}^{(k+1)} \\ \mathbf{z}^{(k+1)} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{(k)} \\ \mathbf{y}^{(k)} \\ \mathbf{z}^{(k)} \end{bmatrix} + \alpha_k \begin{bmatrix} \Delta\mathbf{x}^{(k)} \\ \Delta\mathbf{y}^{(k)} \\ \Delta\mathbf{z}^{(k)} \end{bmatrix}$$

  choosing α_k so that $\mathbf{x}^{(k+1)} > \mathbf{0}$ and $\mathbf{z}^{(k+1)} > \mathbf{0}$.

Note: For nonzero σ, if we set $\alpha_k = 1$, then we will have

$$x_i^{(k+1)} z_i^{(k+1)} \approx \sigma \mu_k,$$

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 ϕ should decrease sufficiently fast, where ϕ is a potential function. One very useful one (Tanabe-Todd-Ye):

$$\phi_\rho(\mathbf{x}, \mathbf{z}) = \rho \log \mathbf{x}^T \mathbf{z} - \sum_{i=1}^{n} \log(x_i z_i)$$

  where $\rho > n$.
- Implementation:
  - Choose $\sigma_k = n/\rho$.
  - Choose α_k using a line search for the function ϕ with an upper bound on α equal to α_{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 $.99\alpha_{max}$ (or similar values) yield a prescribed constant decrease in ϕ.

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 μ 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 σ 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 σ and thus must choose an α_k so that $x_i^{(k+1)} z_i^{(k+1)} \geq \gamma \mu_k$, where γ is chosen between 0 and 1. (A typical γ is 10^{-3}.)

  - Predictor-corrector methods take
    * a predictor step with $\sigma = 0$ and α_k chosen to keep $||\mathbf{Xz} - \mu \mathbf{e}||_2 \leq \theta \mu$ (typical θ 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 μ 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 μ_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 $\mathbf{x} > \mathbf{0}$ and a $\mathbf{z} > \mathbf{0}$.
- It is hard to satisfy the equality constraints $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{A}^T\mathbf{y} + \mathbf{z} = \mathbf{c}$.

So infeasible IPMs replace the step equation by

$$\begin{bmatrix} \mathbf{Z}^{(k)} & \mathbf{0} & \mathbf{X}^{(k)} \\ \mathbf{A} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{A}^T & \mathbf{I} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}^{(k)} \\ \Delta \mathbf{y}^{(k)} \\ \Delta \mathbf{z}^{(k)} \end{bmatrix} = \begin{bmatrix} -\mathbf{X}^{(k)}\mathbf{z}^{(k)} + \sigma_k \mu_k \mathbf{e} \\ \mathbf{c} - \mathbf{A}^T \mathbf{y}^{(k)} - \mathbf{z}^{(k)} \\ \mathbf{b} - \mathbf{A}\mathbf{x}^{(k)} \end{bmatrix}$$

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

Stephen J. Wright, *Primal-Dual Interior-Point Methods* (for linear programming), SIAM, 1997.

A Note on Fiacco and McCormick

Back in the 1960s, Anthony Fiacco and Garth McCormick proposed solving linear programming problems (or any constrained optimization problem) using a barrier method. They called it "Sequential Unconstrained Minimization Technique" or SUMT.

Everyone laughed.

No one is laughing now.

IPM for Nonlinear Programming

- general NLP
- 2nd-order cone programming
- semi-definite programming

Nonlinear Programming

IPM for NLP

Primal problem:

$$\min_{\mathbf{x}} f(\mathbf{x})$$

$$\mathbf{c}(\mathbf{x}) \geq \mathbf{0}$$

Note: Linear equality constraints can be handled as in IPMs for LP, but we'll omit these from the discussion.

Barrier function:

$$B(\mathbf{x}, \mu) = f(\mathbf{x}) - \mu \sum \log c_i(\mathbf{x})$$

Optimality conditions for barrier function:

$$\begin{aligned} \mathbf{g}(\mathbf{x}) - \mu \mathbf{A}(\mathbf{x})^T \mathbf{C}(\mathbf{x})^{-1}\mathbf{e} &= \mathbf{0} \\ \mathbf{c}(\mathbf{x}) &\geq \mathbf{0} \end{aligned}$$

Optimality conditions for NLP primal problem:

$$\begin{aligned} \mathbf{g}(\mathbf{x}) - \mathbf{A}(\mathbf{x})^T \boldsymbol{\lambda} &= \mathbf{0} \\ \mathbf{c}(\mathbf{x}) &\geq \mathbf{0} \\ \boldsymbol{\lambda} &\geq \mathbf{0} \\ \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x}) &= \mathbf{0} \end{aligned}$$

Consequently, we have these multiplier estimates:

$$\boldsymbol{\lambda} = \mu \mathbf{C}(\mathbf{x})^{-1}\mathbf{e}\,,$$

or, equivalently, $\boldsymbol{\lambda}$ is defined by

$$\mathbf{C}(\mathbf{x})\boldsymbol{\lambda} = \mu\mathbf{e}\,.$$

The central path

Therefore, we define the central path as

$$\begin{aligned} \mathbf{g}(\mathbf{x}) - \mathbf{A}(\mathbf{x})^T\boldsymbol{\lambda} &= \mathbf{0} \\ \lambda_i c_i(\mathbf{x}) &= \mu \end{aligned}$$

Follow that path as $\mu \to 0$.

What we need to accomplish:

- Propose a good algorithm for following that path.
- Establish convergence and complexity for some interesting class of problems.

Following that path

$$\begin{aligned} \mathbf{g}(\mathbf{x}) - \mathbf{A}(\mathbf{x})^T\boldsymbol{\lambda} &= \mathbf{0} \\ \lambda_i c_i(\mathbf{x}) &= \mu \end{aligned}$$

What does Newton's method look like for this problem?

$$\begin{bmatrix} \mathbf{H}(\mathbf{x}) - \sum \lambda_i \nabla^2 c_i(\mathbf{x}) & -\mathbf{A}(\mathbf{x})^T \\ \boldsymbol{\Lambda}\mathbf{A}(\mathbf{x}) & \mathbf{C}(\mathbf{x}) \end{bmatrix} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\boldsymbol{\lambda} \end{bmatrix} = - \begin{bmatrix} \mathbf{g}(\mathbf{x}) - \mathbf{A}(\mathbf{x})^T\boldsymbol{\lambda} \\ \mathbf{C}(\mathbf{x})\boldsymbol{\lambda} - \mu\mathbf{e} \end{bmatrix} \equiv \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \end{bmatrix}.$$

Some notes:

- Later, we'll use the notation $\mathbf{p}$ to denote the solution vector for this system of equations.
- Note that the $(1,1)$ block is $\mathbf{H}_L$, the Hessian matrix for the Lagrangian, an old friend.
- Recall that we got a similar system for linear constraints.
- Since $\mathbf{x}$ is temporarily fixed, we drop it from the notation.

We can take $\mathbf{A}^T\mathbf{C}^{-1}$ times the second equation and add it to the first, to eliminate the $\Delta\boldsymbol{\lambda}$ variables:

$$(\mathbf{H}_L + \mathbf{A}^T\mathbf{C}^{-1}\boldsymbol{\Lambda}\mathbf{A})\Delta\mathbf{x} = \mathbf{r}_1 + \mathbf{A}^T\mathbf{C}^{-1}\mathbf{r}_2 \,.$$

Note that $\mathbf{C}^{-1}\boldsymbol{\Lambda} = \mu^{-1}\boldsymbol{\Lambda}^2$, so the matrix is

$$\mathbf{H}_L + \mu^{-1}\mathbf{A}^T\boldsymbol{\Lambda}^2\mathbf{A} \,.$$

Again, just as in our barrier function discussion, we are taking a well-behaved matrix $\mathbf{H}_L$ and adding a piece that is growing without bound in k directions (k = number of active constraints at the solution) as $\mu \to 0$.

We overcome it the same way, using a QR factorization and then changing to that coordinate system. Review the old notes (Constrained Optimization, Feasible Direction Methods, p. 2).

Convergence and complexity

We have now developed the basic algorithm: apply Newton's method to the equations defining the central path.

What we haven't specified is

- how far to step along the Newton direction, and
- when and how fast we can reduce μ.

The answers to these questions are given in N&S 17.7.

An important assumption:

Convergence results for general nonlinear programming are not very strong, since having multiple local solutions can be hindrances.

So we will assume that the function $f(\mathbf{x})$ is convex, and the feasible set defined by $\mathbf{c}(\mathbf{x}) \geq \mathbf{0}$ is convex.

For convex programming problems such as these, the convergence results are almost as nice as for linear programming.

Some ingredients to an algorithm

1. Choose a self-concordant barrier function. This overcomes the problems of the 1960s algorithms and enables proof that the complexity is low.

   - A convex function F of a single variable x is self-concordant on some open convex set if, for all x in the set,

     $$|F'''(x)| \leq 2F''(x)^{3/2}\,.$$

     Example:

     $$\begin{aligned} F(x) &= -\log x \\ F''(x) &= \frac{1}{x^2} \\ F'''(x) &= -\frac{2}{x^3} \end{aligned}$$

     so $-\log x$ is self-concordant for $x > 0$. []

   - A convex function F of several variables is self-concordant on some domain if $F(\mathbf{x}^{(k)}) \to \infty$ as $\mathbf{x}^{(k)} \to$ boundary.

     $$| \bigtriangledown^3 F(\mathbf{x})[\mathbf{h},\mathbf{h},\mathbf{h}]| \leq 2[\mathbf{h}^T \bigtriangledown^2 F(\mathbf{x})\mathbf{h}]^{3/2}$$

     for all $\mathbf{x}$ in the interior of the domain and all $\mathbf{h} \in \mathcal{R}^n$, where $\bigtriangledown^3 F(\mathbf{x})[\mathbf{h},\mathbf{h},\mathbf{h}]$ is a third-order directional derivative:

     $$\bigtriangledown^3 F(\mathbf{x})[\mathbf{h}_1,\mathbf{h}_2,\mathbf{h}_3] \equiv \frac{\partial^3}{\partial t_1 \partial t_2 \partial t_3} F(\mathbf{x}+t_1\mathbf{h}_1+t_2\mathbf{h}_2+t_3\mathbf{h}_3)|_{t_1=t_2=t_3=0}\,.$$

     Example: -$\log(\mathbf{a}^T\mathbf{x}-b)$ is self-concordant on the set defined by $\mathbf{a}^T\mathbf{x}-b > 0$, so the log barrier can be used for linear constraints. []

   What do self-concordant functions do for us?

   - They ensure that the second derivative is not changing too rapidly, so a quadratic model works well and Newton's method converges quickly.
   - They ensure that as the barrier parameter changes, the solution does not move too much, so the solution to the previous problem is a good guess for the next problem.

2. If the Newton step is zero, then we know that we have reached the solution. Thus we can measure the progress of the algorithm using the norm of the Newton step. This number is called the Newton decrement and is measured as

   $$\delta \equiv \|\mathbf{p}\|_{\mathbf{x}}$$

where

$$\|\mathbf{h}\|_{\mathbf{x}}^2 = \mathbf{h}^T \bigtriangledown^2 F(\mathbf{x})\mathbf{h},$$

F is the barrier function, and $\mathbf{x}$ is the current iterate.

This is truly a norm if $\bigtriangledown^2 F(\mathbf{x})$ is positive definite for all $\mathbf{x}$ of interest, which we assume from now on.

3. How far should we step in the Newton direction $\mathbf{p}$? We take a damped Newton step:

$$\alpha = \frac{1}{1+\delta}.$$

This means that the step length is always bounded above by 1, and converges to 1 as we approach the solution.

4. We'll put the primal problem in a standard form

$$\min_{\mathbf{x},\mathbf{y}} \mathbf{y}$$

subject to

$$\begin{array}{rcl} \mathbf{c}(\mathbf{x}) & \geq & \mathbf{0} \\ \mathbf{y} - \mathbf{f}(\mathbf{x}) & \geq & \mathbf{0} \end{array}$$

where $\mathbf{y}$ is a new scalar variable, making $n+1$ variables in all.

5. We stop the iteration for a subproblem (a fixed value of μ) when we have a feasible $(\mathbf{x}, \mathbf{y})$ and

$$\delta \leq \kappa$$

for some fixed positive number κ.

6. We update the barrier parameter μ by the rule

$$\mu^{(k+1)} = \left(1 + \frac{\gamma}{\sqrt{\nu}}\right)^{-1} \mu^{(k)}$$

where γ is a fixed positive number and ν is defined on p.604.

This completely specifies the algorithm.

Now, what can we say about its speed?

The convergence result

This IPM can determine the solution to a convex programming problem, within a specified tolerance, in polynomial time.

(This is Theorem 17.4 in N&S.)

The result also holds for linear programming as a special case.

Two special cases of convex programming

We'll conclude our consideration of IPMs by discussing two special cases of convex programs:

- second-order cone programs.
- semi-definite programs.

By using these formulations, we can solve some surprisingly complicated problems.

Second-order cone programs

A second-order cone program (SOCP) is defined as

$$\min_{\mathbf{x}} \mathbf{f}^T \mathbf{x}$$

subject to

$$\|\mathbf{A}_i \mathbf{x} + \mathbf{b}_i\| \le \mathbf{c}_i^T \mathbf{x} + d_i\,, \;\; i = 1, \ldots, m$$

Unquiz:

1. Show that linear programming is a special case of SOCP.
2. Show that this problem is an SOCP:
$$\min_{\mathbf{x}} \sum_{i=1}^{\ell} \|\mathbf{F}_i \mathbf{x} + \mathbf{g}_i\|_2$$
3. Show that this problem is an SOCP:
$$\min_{\mathbf{x}} \max_i \|\mathbf{F}_i \mathbf{x} + \mathbf{g}_i\|_2$$

We'll come back to these last two problems when we discuss nonlinear least squares.

[]

Many other problems can be written as SOCPs: for example,

$$\min_{\mathbf{x}} \sum \frac{1}{\mathbf{a}_i^T \mathbf{x} + b_i}$$

subject to linear inequality constraints, including the positivity of each denominator.

For more information: See the article by Lobo, Vandenberghe, Boyd, and Lebret in *Linear Algebra and Its Applications* 284 (1998) p.193 for this and other examples from portfolio management, truss design, etc.

Semi-definite Programming

SOCPs are a special case of semi-definite programming problems. These problems are of the form

$$\min_{\mathbf{X}} \mathbf{C} \bullet \mathbf{X}$$

subject to

$$\begin{array}{rcl} \mathbf{A}(\mathbf{X}) & = & \mathbf{b} \\ \mathbf{X} & \geq & \mathbf{0} \end{array}$$

They look a lot like linear programming problems, but

- We define
$$\mathbf{C} \bullet \mathbf{X} \equiv trace(\mathbf{C}^T \mathbf{X}) = \sum_{i,j} c_{ij} x_{ij}$$
where $\mathbf{C}$ and $\mathbf{X}$ are symmetric matrices.
- We define
$$\mathbf{A}(\mathbf{X}) \equiv \begin{bmatrix} \mathbf{A}_1 \bullet \mathbf{X} \\ \dots \\ \mathbf{A}_m \bullet \mathbf{X} \end{bmatrix}$$
- We define
$$\mathbf{X} \geq \mathbf{0}$$
to mean that $\mathbf{X}$ is symmetric positive semidefinite.

The dual problem:

$$\max_{\mathbf{y},\mathbf{S}} \mathbf{b}^T \mathbf{y}$$

subject to

$$\begin{array}{rcl} \sum_{i=1}^{m} \mathbf{y}_i \mathbf{A}_i + \mathbf{S} & = & \mathbf{C} \\ \mathbf{S} & \geq & \mathbf{0} \end{array}$$

where $\mathbf{A}_i$ and $\mathbf{S}$ are symmetric matrices.

Examples:

- In control theory, structural optimization, and matrix theory, we frequently need to minimize the maximum eigenvalue of a matrix

$$\mathbf{B}(\mathbf{y}) = \mathbf{T}_0 + \sum_i y_i \mathbf{T}_i$$

  where the matrices $\mathbf{T}_i$ are given.

  We can write this as

$$\max_{\mathbf{y},t} -t$$

  subject to

$$\begin{aligned} t\mathbf{I} - \mathbf{B}(\mathbf{y}) &= \mathbf{S} \\ \mathbf{S} &\geq \mathbf{0} \end{aligned}$$

- Some data fitting problems can also be cast as semidefinite programs. For example,

$$\min_{\mathbf{x}} \max_i |\log(\mathbf{a}_i^T \mathbf{x}) - \log(b_i)|$$

For more information on semidefinite programming: see

- Vandenberghe and Boyd, *SIAM Review* 38 (1996) p.49.
- Todd, *Acta Numerica* 10 (2001) p.515.

Final words

- When using IPMs, go for the most specific algorithm you can:
  - Don't use a semidefinite programming code if the problem is a SOCP.
  - Don't use a SOCP code on a linear programming problem.
- Although you are writing your own code as an exercise, writing a production-quality code is quite difficult. Use available software rather than relying on your own.
- The most important reference for IPMs:
  Interior-Point Polynomial Algorithms in Convex Programming,
  Y. Nesterov and A. Nemirovskii, SIAM Press, 1994.