
Interior Point Methods

The Plan

To study interior point methods (IPMs) for

- linear programming
- **You are here:** nonlinear programming
- 2-nd order cone programming
- (Semi-definite programming)

Nonlinear Programming

IPM for NLP

Primal problem:

$$\min_x f(x)$$

$$c(x) \geq 0$$

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

Barrier function:

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

Optimality conditions for barrier function:

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

Optimality conditions for NLP primal problem:

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

Consequently, we have these multiplier estimates:

$$\lambda = \mu \mathbf{C}^{-1} \mathbf{e},$$

or, equivalently, λ is defined by

$$C\lambda = \mu e.$$

The central path

Therefore, we define the **central path** as

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

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

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

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

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

$$\begin{bmatrix} H(x) - \sum \lambda_i \nabla^2 c_i(x) & -A(x)^T \\ \Lambda A(x) & C(x) \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \end{bmatrix} = - \begin{bmatrix} g(x) - A(x)^T \lambda \\ C\lambda - \mu \end{bmatrix} \equiv \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}.$$

Some notes:

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

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

$$(H_L + A^T C^{-1} \Lambda A) \Delta x = r_1 + A^T C^{-1} r_2.$$

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

$$H_L + \mu^{-1} A^T \Lambda^2 A.$$

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

We overcome it the same way, using a QR factorization and then changing to that coordinate system. Review the old notes (Unit 3, Part 3, p.6).

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 $f(x)$ is convex, and the feasible set defined by $c(x) \geq 0$ 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 $F(x^{(k)}) \rightarrow \infty$ as $x^{(k)} \rightarrow$ boundary.

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

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

$$\nabla^3 F(x)[h_1, h_2, h_3] \equiv \frac{\partial^3}{\partial t_1 \partial t_2 \partial t_3} F(x + t_1 h_1 + t_2 h_2 + t_3 h_3) \Big|_{t_1=t_2=t_3=0}.$$

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

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

$$\delta \equiv \|p\|_x$$

where

$$\|h\|_x^2 = h^T \nabla^2 F(x)h,$$

F is the barrier function, and x is the current iterate.

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

3. **How far should we step in the Newton direction 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**

$$\min_{x,y}$$

subject to

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

where 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 (x, 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

$$\min_x f^T x$$

subject to

$$\|A_i x + b_i\| \leq c_i^T x + d_i, \quad i = 1, \dots, m$$

Unquiz:

1. Show that linear programming is a special case of SOCP.
2. Show that this problem is an SOCP:

$$\min_x \sum_{i=1}^{\ell} \|F_i x + g_i\|_2$$

3. Show that this problem is an SOCP:

$$\min_x \max_i \|F_i x + g_i\|_2$$

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,

$$\min_x \sum \frac{1}{a_i^T x + b_i}$$

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

$$\min_X C \bullet X$$

subject to

$$\begin{aligned} A(X) &= b \\ X &\geq 0 \end{aligned}$$

They look a lot like linear programming problems, but

- We define

$$C \bullet X \equiv \text{trace}(C^T X) = \sum_{i,j} c_{ij} x_{ij}$$

where C and X are **symmetric matrices**.

- We define

$$A(X) \equiv \begin{bmatrix} A_1 \bullet X \\ \dots \\ A_m \bullet X \end{bmatrix}$$

- We define

$$X \geq 0$$

to mean that X is **symmetric positive semidefinite**.

The dual problem:

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

subject to

$$\begin{aligned} \sum_{i=1}^m y_i A_i + S &= C \\ S &\geq 0 \end{aligned}$$

where A_i and S are symmetric matrices.

Examples:

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

$$B(y) = T_0 + \sum_i y_i T_i$$

where the matrices T_i are given.

We can write this as

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

subject to

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

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

$$\min_x \max_i |\log(a_i^T x) - \log(b_i)|$$

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.