CSE 599: Interplay between Convex Optimization and Geometry **Winter 2018**

Lecture 14: Primal-Dual Interior-Point Method

Lecturer: Yin Tat Lee

Disclaimer: *Please tell me any mistake you noticed.*

In this and next week, we study interior point methods. In practice, this framework reduces the problem of optimizing a convex function to <50 many linear systems. In theory, it reduces the problem to $\tilde{O}(\sqrt{n})$ many. Although it has many numerical stability problem in practice and it rarely offers nearly linear time algorithm, this is the current best framework for optimizing general convex functions in both theory and practice for the high accuracy regime.

For simplicity, we start by describing interior point method for linear programs first. The first half is about the polynomial bound and the second half is about the implementation details.

14.1 Short-step interior point method

14.1.1 Basic Properties of Central Path

We first consider the primal problem

$$(\mathrm{P}): \qquad \min_x c^\top x \text{ subjects to } Ax = b, x \geq 0$$

where $A \in \mathbb{R}^{m \times n}$. The difficulty of linear programs is the constraint $x \geq 0$. Without this constraint, we can simply solve it by a linear system. One natural idea to solve linear programs is to replace the hard constraint $x \geq 0$ by some other function. So, let consider the regularized version of the linear program

$$(\mathrm{P}_t): \qquad \min_x c^\top x - t\sum_{i=1}^n \ln x_i \text{ subjects to } Ax = b.$$

We will explain the reason of choosing $\ln x$ in more detail next lecture. For now, you can think it as a nice function that blows up at $x = 0$.

One can think that $-\ln x$ gives a force from every constraint $x \geq 0$ to make sure $x \geq 0$ is true. Since the gradient of $-\ln x$ blows up when $x = 0$, when x is close enough, the force is large enough to counter the cost c. When $t \to 0$, then the problem (P_t) is closer to the original problem (P) and hence the minimizer of (P_t) is closer to a minimizer of (P).

First, we give a formula for the minimizer of (P_t).

Lemma 14.1.1 (Existence and Uniqueness of central path)**.** *If the polytope $\{Ax = b, x \geq 0\}$ has an interior, then the optimum of (P_t) is uniquely given by*

$$\begin{aligned} xs &= t, \\ Ax &= b, \\ A^\top y + s &= c, \\ (x, s) &\geq 0 \end{aligned}$$

where $xs = t$ is a shorthand of $x_i s_i = t$ for all i.

Proof. The optimality condition is given by

$$c - \frac{t}{x} = A^\top y$$

where y is the dual variable. Write $s_i = \frac{t}{x_i}$, we have the formula. The solution is unique because the function $-\ln x$ is strictly convex. □

Definition 14.1.2. We define the central path (x_t, y_t, s_t) of the linear program is given by

$$\begin{aligned} x_t s_t &= t, \\ Ax_t &= b, \\ A^\top y_t + s_t &= c, \\ (x_t, s_t) &\geq 0. \end{aligned}$$

To give another interpretation of the central path, we note that the dual problem is

$$\text{(D)}: \qquad \max_{y,s} b^\top y \text{ subjects to } A^\top y + s = c, s \geq 0.$$

Note that for any feasible x and y, we have that

$$0 \leq c^\top x - b^\top y = c^\top x - x^\top A^\top y = x^\top s.$$

Hence, (x, y, s) solves the linear program if it satisfies the central path equation with $t = 0$. Therefore, central path is a balanced way to decrease $x_i s_i$ uniformly to 0.

Now, we formalize the intuition that for small t, x_t is a good approximation of the primal solution.

Lemma 14.1.3 (Duality Gap)**.** *We have that*

$$\textit{Duality Gap} = c^\top x_t - b^\top y_t = c^\top x_t - x_t^\top A^\top y_t = x_t^\top s_t = t \cdot n.$$

The interior point method follows the following framework:

1. Find $\mathcal{C}_1$
2. Until $t < \frac{\varepsilon}{n}$,
   (a) Use $\mathcal{C}_t$ to find $\mathcal{C}_{(1-h)t}$ for $h = \frac{1}{10\sqrt{n}}$.

Note that this algorithm only finds a solution with ε error. If the linear program is integral, we can simply stop at small enough ε and round it off to the closest integral point.

14.1.2 Finding the initial point

The first question to find $\mathcal{C}_1$. This can be handled by extending the problem into higher dimension. By rescaling the problem and adding a new variable, we can assume the solution satisfies $\sum_i x_i = n$ and that $\|c\|_\infty \leq \frac{1}{2}$.

Now, we consider the following linear program

$$\min_{x \geq 0, x' \geq 0} c^\top x + M \cdot x' \text{ subjects to } \begin{bmatrix} A & (b - A1) \\ -1^\top & 0 \end{bmatrix} \begin{bmatrix} x \\ x' \end{bmatrix} = \begin{bmatrix} b \\ -n \end{bmatrix}$$

for some very large M. This linear program has the following properties:

- All 1 vector is a feasible interior point of the linear program.
- Any vector x such that $Ax = b$ and $x \geq 0$ gives a feasible vector $\begin{bmatrix} x \\ 0 \end{bmatrix}$ in this linear program. Therefore, if M is large enough, this linear program has the same solution as the old linear program.
- The dual polytope
$$\begin{bmatrix} A^\top & -1 \\ (b - A1)^\top & 0 \end{bmatrix} y \leq \begin{bmatrix} c \\ M \end{bmatrix}$$
has a trivial interior point $y = \begin{bmatrix} 0_m \\ 1 \end{bmatrix}$. (used $\|c\|_\infty \leq \frac{1}{2}$ and M large enough)

Therefore, we have an initial point (x, y, s) with $(x, s) > 0$. To make $xs = 1$, we can rescale the constraint matrix A.

14.1.3 Following the central path

During the algorithm, we maintain a point (x, y, s) such that $Ax = b$, $A^\top y + s = c$ and xs is close to t. We show how to find a feasible $(x + \Delta x, y + \Delta y, s + \Delta y)$ such that it is even closer to t. We can write the equation as follows:

$$\begin{aligned} (x + \Delta x)(s + \Delta s) &\approx t, \\ A(x + \Delta x) &= b, \\ A^\top (y + \Delta y) + (s + \Delta s) &= c. \end{aligned}$$

(Omitted the non-negative conditions.) Using our assumption on (x, y, s) and noting that $\Delta x \cdot \Delta s$ is small, the equation can simplified as follows

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S & 0 & X \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta s \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ t - xs \end{bmatrix}.$$

This is a linear equation and hence we can solve it exactly as follows:

Exercise 14.1.4. Let $r = t - xs$. Prove that $S\Delta x = (I - P)r$ and $X\Delta s = Pr$ where $P = XA^\top (AS^{-1}XA^\top)^{-1}AS^{-1}$.

Lemma 14.1.5. *Let $x' = x + \Delta x$ and $s' = s + \Delta s$. If $\sum_i (x_i s_i - t)^2 \leq \varepsilon^2 t^2$, we have that*

$$\sum_i (x'_i s'_i - t)^2 \leq \left(\varepsilon^4 + O(\varepsilon^5)\right) t^2.$$

Proof. We have that

$$\text{LHS} = \sum_i (x'_i s'_i - t)^2 = \sum_i (\Delta x)_i^2 (\Delta s)_i^2 \leq \frac{1}{(1 - \varepsilon)^2 t^2} \cdot \sum_i x_i^2 s_i^2 (\Delta x)_i^2 (\Delta s)_i^2.$$

Using the formula in Exercise 14.1.4, this can be bounded by

$$(1 - \varepsilon)^2 t^2 \text{LHS} \leq \sum_i ((I - P)r)_i^2 (Pr)_i^2 \leq \|(I - P)r\|_4^2 \|Pr\|_4^2 \leq \|(I - P)r\|_2^2 \|Pr\|_2^2.$$

Note that P is a projection matrix and that

$$\|P\|_{\text{op}} = \left\|S^{\frac{1}{2}} X^{\frac{1}{2}}\right\|_{\text{op}} \left\|S^{-\frac{1}{2}} X^{\frac{1}{2}} A^\top (AS^{-1}XA^\top)^{-1} AS^{-\frac{1}{2}} X^{\frac{1}{2}}\right\|_{\text{op}} \left\|S^{-\frac{1}{2}} X^{-\frac{1}{2}}\right\|_{\text{op}} \leq \frac{1}{1 - \varepsilon}$$

where we used that the middle matrix is an orthogonal projection matrix. Similarly, we have $\|I - P\|_{\text{op}} \leq \alpha$. Hence, we have

$$\text{LHS} \leq \frac{1}{(1 - \varepsilon)^6 t^2} \|r\|_2^4 = \frac{\varepsilon^4 t^4}{(1 - \varepsilon)^6 t^2}.$$

□

Using this, we have the following theorem:

Theorem 14.1.6. *We can solve linear program with ε "error" in $O(\sqrt{n}\log(\frac{n}{\varepsilon}))$ many iterations and each iteration only need to solve 1 linear system.*

Proof. Let $E = \sum_i (x_i s_i - t)^2$ be the error of the current iteration. We always maintain $E \le \frac{t^2}{50}$ for current (x, y, s) and current t. Whenever $E \ge \frac{t^2}{200}$, we decrease E by Lemma 14.1.5. If not, we decrease t by $t(1 - \frac{1}{10\sqrt{n}})$. Note that

$$\sum_i (x_i s_i - t(1-h))^2 \le 2\sum_i (x_i s_i - t)^2 + 2t^2 h^2 n \le \frac{2t^2}{200} + \frac{2t^2}{100} \le \frac{(t(1-h))^2}{50}.$$

Therefore, the invariant is preserved after this step of t. Hence, each iteration, we simply solve 1 linear system and decrease t by $(1 - \frac{1}{10\sqrt{n}})$ factor. So, it takes $O(\sqrt{n}\log(\frac{n}{\varepsilon}))$ to decrease t from 1 to $\frac{\varepsilon}{n}$. □

Remark. Due the our reduction to get the initial point, we are paying huge factors in the logarithmic terms.

For many problems, this factor is polynomial and hence the total running time is still $O(\sqrt{n}\log(\frac{n}{\varepsilon}))$. However, for some problem such as circuit evaluation, the factor can be exponentially large and hence we only get a $n^{1.5}$ time algorithm. This is natural because otherwise, we would prove that any program can be run in $\sqrt{n}$ depth!

14.1.4 Where does $\sqrt{n}$ come from?

Central path satisfies the following ODE

$$\begin{aligned} S_t \frac{d}{dt} x_t + X_t \frac{d}{dt} S_t &= 1, \\ A \frac{d}{dt} x_t &= 0, \\ A^\top \frac{d}{dt} y_t + \frac{d}{dt} s_t &= 0. \end{aligned}$$

Solving this linear system, we have that $S_t \frac{dx_t}{dt} = (I - P_t)1$ and $X_t \frac{ds_t}{dt} = P_t 1$ where $P_t = X_t A^\top (A S_t^{-1} X_t A^\top)^{-1} A S_t^{-1}$. Using that $x_t s_t = t$, we have that

$$P_t = X_t A^\top (A X_t^2 A)^{-1} A X_t = S_t^{-1} A^\top (A S_t^{-2} A)^{-1} A S_t$$

and that $X_t^{-1} \frac{dx_t}{dt} = \frac{1}{t}(I - P_t)1$ and $S_t^{-1} \frac{ds_t}{dt} = \frac{1}{t} P_t 1$. Equivalently, we have

$$\frac{d \ln x_t}{d \ln t} = (I - P_t)1 \text{ and } \frac{d \ln s_t}{d \ln t} = P_t 1.$$

Note that

$$\|P_t 1\|_\infty \le \|P_t 1\|_2 = \sqrt{n}.$$

Hence, x_t and s_t can changes at most constant factor when we change t by a $1 \pm \frac{1}{\sqrt{n}}$ factor.

Exercise 14.1.7. If we are given x such that $\|\ln x - \ln x_t\|_\infty \le O(1)$, then we can find x_t by solving $\tilde{O}(1)$ linear systems.

If v is a random vector with length $\sqrt{n}$, then we have $\|P_t v\|_\infty = O(\sqrt{\log n})$. Maybe this is the reason we see the running time of interior point closer to nearly constant iterations instead of polynomial.

14.2 Mehrotra predictor–corrector method

In practice, we implement the interior point slightly differently. First, we do not need to find an initial point that is feasible. Hence, we do not need to do that reduction. Second, our step is not aiming at $xs = t$. Instead, every step, we first aim at $xs = 0$ (called affine direction). Then, we check if we can take a large step on this affine direction. This is used to evaluation how long step size h we can take. Finally, we compute a second order update step.

Formally, the algorithm as follows:

- Compute initial point:
  - Set x, y and s are the minimizer of the problem $\min_{Ax=b} \|x\|^2$ and the problem $\min_{A^\top y+s=c} \|s\|^2$.
  - Set $\delta_x = \max(-1.5 \min_i x_i, 0)$ and $\delta_s = \max(-1.5 \min_i s_i, 0)$.
  - Set $y^{(0)} = y$, $x^{(0)} = x + \delta_x + 0.5\frac{(x+\delta_x e)^\top (s+\delta_s e)}{\sum_i (s_i+\delta_s)}$ and $s^{(0)} = s + \delta_s + 0.5\frac{(x+\delta_x e)^\top (s+\delta_s e)}{\sum_i (x_i+\delta_x)}$.
- For $k = 0, 1, 2, \cdots$
  - Solve the equation

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S^{(k)} & 0 & X^{(k)} \end{bmatrix} \begin{bmatrix} \Delta x^{\text{aff}} \\ \Delta y^{\text{aff}} \\ \Delta s^{\text{aff}} \end{bmatrix} = \begin{bmatrix} -r_c \\ -r_b \\ -X^{(k)} S^{(k)} e \end{bmatrix}$$

    where $r_b = Ax^{(k)} - b$ and $r_c = A^\top y^{(k)} + s^{(k)} - c$.
  - Calculate

$$\begin{aligned} \alpha_{\text{aff}}^{\text{pri}} &= \arg\max\{\alpha \in [0,1] | x^{(k)} + \alpha \Delta x^{\text{aff}} \geq 0\}; \\ \alpha_{\text{aff}}^{\text{dual}} &= \arg\max\{\alpha \in [0,1] | s^{(k)} + \alpha \Delta s^{\text{aff}} \geq 0\}; \\ \mu &= x^{(k)\top} s^{(k)}/n; \\ \mu_{\text{aff}} &= (x^{(k)} + \alpha_{\text{aff}}^{\text{pri}} \Delta x^{\text{aff}})^\top (s^{(k)} + \alpha_{\text{aff}}^{\text{dual}} \Delta s^{\text{aff}})/n; \end{aligned}$$

  - Set centering parameter to $\sigma = \min\left(\left(\frac{\mu_{\text{aff}}}{\mu}\right)^3, 1\right)$;
  - Solve the equation

$$\begin{bmatrix} 0 & A^\top & I \\ A & 0 & 0 \\ S^{(k)} & 0 & X^{(k)} \end{bmatrix} \begin{bmatrix} \Delta x^{\text{cc}} \\ \Delta y^{\text{cc}} \\ \Delta s^{\text{cc}} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \sigma\mu e - \Delta X^{\text{aff}} \Delta S^{\text{aff}} e \end{bmatrix};$$

  - Calculate

$$\begin{aligned} (\Delta x^{(k)}, \Delta y^{(k)}, \Delta s^{(k)}) &= (\Delta x^{\text{aff}} + \Delta x^{\text{cc}}, \Delta y^{\text{aff}} + \Delta y^{\text{cc}}, \Delta s^{\text{aff}} + \Delta s^{\text{cc}}); \\ \alpha_{\max}^{\text{pri}} &= \arg\max\{\alpha \in [0,1] | x^{(k)} + \alpha \Delta x^{(k)} \geq 0\}; \\ \alpha_{\max}^{\text{dual}} &= \arg\max\{\alpha \in [0,1] | s^{(k)} + \alpha \Delta s^{(k)} \geq 0\}; \end{aligned}$$

  - Set $\alpha_k^{\text{pri}} = \min(0.99 \cdot \alpha_{\max}^{\text{pri}}, 1)$ and $\alpha_k^{\text{dual}} = \min(0.99 \cdot \alpha_{\max}^{\text{dual}}, 1)$;
  - Set

$$\begin{aligned} x^{(k+1)} &= x^{(k)} + \alpha_k^{\text{pri}} \Delta x^{(k)}; \\ (y^{(k+1)}, s^{(k+1)}) &= (y^{(k)}, s^{(k)}) + \alpha_k^{\text{dual}} (\Delta y^{(k)}, \Delta s^{(k)}); \end{aligned}$$

Listing 2: Mehrotra predictor corrector method

```
% This is an implementation of Mehrotra's predictor-coorector algorithm.
% We solve the linear program min <c,x> subject to Ax=b, x>=0
function [x, iter, mu] = solveLP(A, b, c)
m = size(A,1);  n = size(A,2); maxiter = 100;
P = colamd(A');
A = A(P,:);      b = b(P);

% Find an initial point
R = builtin('_cholinf', A * A');
y = R\(R'\(A*c));    s_ = c - A' * y;     x_ = A' * (R\(R'\b));
dx = max(-1.5 * min(x_), 0);      ds = max(-1.5 * min(s_), 0);
x = x_ + dx + 0.5 * ((x_ + dx)' * (s_+ds))/sum(s_+ds);
s = s_ + ds + 0.5 * ((x_ + dx)' * (s_+ds))/sum(x_+dx);
nb = mean(abs(b));
mu = [];
for iter = 1:maxiter
    mu(end+1) = sum(x.*s)/n;
    rb = A * x - b;      rc = A' * y + s - c;     rxs = x .* s;
    if mu(end) < 1e-6 && mean(abs(rb)) < 1e-6 * nb, break; end

    R = builtin('_cholinf', A * diag(sparse(x./s)) * A');
    [dx_a, dy_a, ds_a] = step_dir(A, R, x, s, rc, rb, rxs);

    xa_step = dist(x, dx_a);     sa_step = dist(s, ds_a);
    mu_a = sum((x+xa_step*dx_a).*(s+sa_step*ds_a))/n;
    sigma = min((mu_a/mu(end))^3,1);

    rxs = -(sigma * mu(end) - dx_a .* ds_a);
    [dx_c, dy_c, ds_c] = step_dir(A, R, x, s, zeros(n,1), zeros(m,1), rxs);

    dx = dx_a + dx_c;    dy = dy_a + dy_c;    ds = ds_a + ds_c;
    x_step = min(0.99*dist(x, dx),1);
    s_step = min(0.99*dist(s, ds),1);

    x = x + x_step * dx; y = y + s_step * dy; s = s + s_step * ds;
end
if iter == maxiter, x = ones(n,1) * NaN; end % x = NaN if doesn't converge.
end
% This outputs the solution of the linear system
% [0 A' I; A 0 0 ;S 0 X][dx;dy;ds] = [-rc, -rb, -rxs]
% The chol decomposition of A S^-1 X A' = R' R is given
function [dx, dy, ds] = step_dir(A, R, x, s, rc, rb, rxs)
    r = - rb + A * ((-x.*rc+rxs)./s);
    dy = R\(R'\r);  ds = -rc - A' * dy; dx = -(rxs + x .* ds)./s;
end
function [step] = dist(x, dx)
    step = -x./dx;
    step(dx>=0) = Inf;
    step = min(step);
end
```

We note the following:

1. Usually linear programs are given by $\min_{Ax=b, l \le x \le u} c^\top x$. The best way is to handle them directly. But for the simplicity of the code, we convert the linear program to $\min_{Ax=b, x \ge 0} c^\top x$ as follows:
   (a) For any unconstrained x, replaced it by $x = x^+ - x^-$ with new constraints $x^+ \ge 0$ and $x^- \ge 0$.
   (b) For any two sided constraints $l \le x \le u$, we replace it to $x' = u - x$, $x \ge l$ and $x' \ge 0$.
   (c) For any one-sided constraints $x \ge l$, shift it to $x \ge 0$.
2. This code is barely stable enough to pass all test cases in netlib problems if we perturb c by $c + \varepsilon 1$ where ε is some tiny positive number. This avoided the issue that the set of optimum point can be unbounded. All examples are solved in 70 iterations, most of them solved in 40 iterations. (for 10^{-6} error).
3. I have not implement the test for infeasibility (See [85]) and the procedure of rounding (See [83]).
4. I have not implement methods to simplify the linear program or to rescale the program. However, to make the algorithm faster for problems with dense col in A, we split the corresponding columns into many copies and duplicate the corresponding variables (not shown in the matlab code above, but in the zip.). Usually this is not done in this way, but modify the linear system algorithm to handle those columns separately.
5. One may able to slightly improve the running time by doing a higher order approximation of the central path.

For more implementation details, see [84].

References

[83] Erling D Andersen. On exploiting problem structure in a basis identification procedure for linear programming. *INFORMS Journal on Computing*, 11(1):95–103, 1999.

[84] Stephen J Wright. *Primal-dual interior-point methods*. Siam, 1997.

[85] Xiaojie Xu, Pi-Fang Hung, and Yinyu Ye. A simplified homogeneous and self-dual linear programming algorithm and its implementation. *Annals of Operations Research*, 62(1):151–171, 1996.