CSE530 Algorithms & Complexity
Lecture 9: Interior Point Methods

Antoine Vigneron

Ulsan National Institute of Science and Technology

April 9, 2018

1 Introduction

2 Introduction

3 Newton's method

4 Logarithmic barrier method

5 Conclusion

Course Organization

- Midterm: Wednesday 04.18 14:30–17:00, room TBA. Closed book. Covers all lectures.
- Last exercise set posted on Tuesday (04.10), solutions by Saturday.

Introduction

- Previous lecture: the *simplex algorithm* is a practical algorithm for linear programming, but it does not run in (worst-case) polynomial time.
- Walks along edges of the feasible region, moving from vertex to vertex.

- In this lecture, I present a polynomial-time algorithm.
- It is an *interior-point* method.
- As opposed to the simplex algorithm, it walks in the *interior* of the feasible region.
- I will not give proofs. The goal is to show you how this algorithm works.
- Reference: Convex Optimization by Boyd and Vandenberghe. (Freely available online.)

Newton's Method

- Newton's method is a very efficient *unconstrained* minimization method for smooth convex functions.
- (Different from Newton's method for finding roots.)
- It finds the minimum over $\mathbb{R}^n$ of a twice-differentiable convex function $f : \mathbb{R}^n \to \mathbb{R}$. We denote

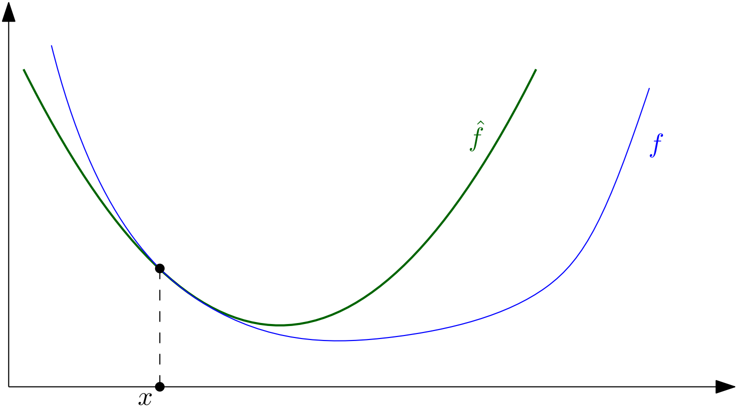
$$p^* = \min_{x \in \mathbb{R}^n} (f(x))$$

and

$$x^* = \arg\min_{x \in \mathbb{R}^n} f(x),$$

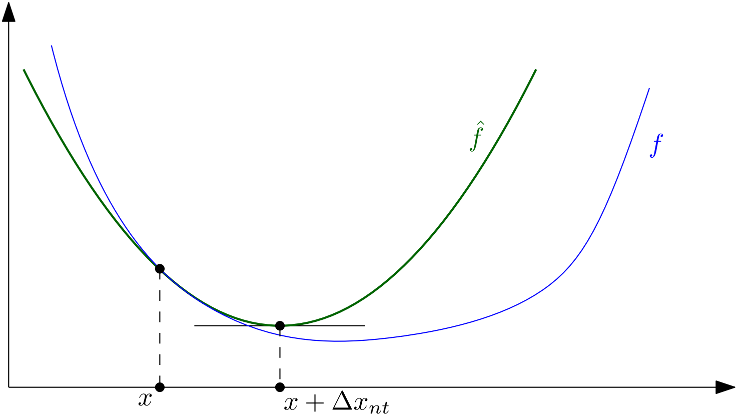
so $p^* = f(x^*)$.

Newton's Method: 1D Case



$$\hat{f}(x+\Delta x) = f(x) + f'(x)\Delta x + f''(x)\frac{\Delta x^2}{2}$$

Newton's Method: 1D Case



$$\hat{f}(x + \Delta x) = f(x) + f'(x)\Delta x + f''(x)\frac{\Delta x^2}{2}$$

Newton's Method: 1D Case

- $\hat{f}$ is the order-2 Taylor approximation of f at x.
- The *Newton step* is the value Δx_{nt} such that $\hat{f}(x + \Delta x_{nt})$ is minimum, so

$$\Delta x_{nt} = -\frac{f'(x)}{f''(x)}.$$

- The *Newton decrement* λ is given by

$$\lambda^2 = \frac{f'(x)^2}{f''(x)}.$$

  Then $\frac{1}{2}\lambda^2 = f(x) - \hat{f}(x + \Delta x_{nt})$ is the square of the difference between $f(x)$ and the minimum of $\hat{f}$.
- We regard $\frac{1}{2}\lambda^2$ as an approximation of the error made at the current step.

Newton's Method: 1D Case

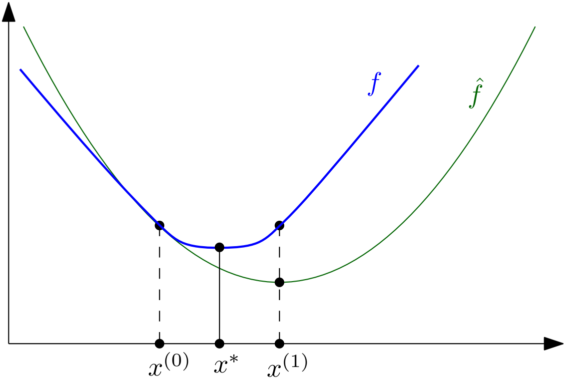
- We start from a point x in the domain of f, and we repeatedly add the Newton step to the current solution x, until the (approximate) error is smaller than ε.

Pseudocode

```
procedure PURE 1D NEWTON(f, x, ε)
    while true do
        Δx_nt ← −f′(x)/f″(x)              ▷ Newton step
        λ² ← f′(x)²/f″(x)                 ▷ Newton decrement
        if λ²/2 ⩽ ε then                  ▷ stopping criterion
            return x
        x ← x + Δx_nt                     ▷ update
```

Newton's Method: 1D Case

- The pseudocode on previous slide gives the *pure* Newton's method.
- This method works well if we start close enough to the optimal.
- If not, it may diverge. For instance, it could traverse an infinite sequence $x^{(0)}$, $x^{(1)}$, $x^{(0)}$, $x^{(1)}$ ...

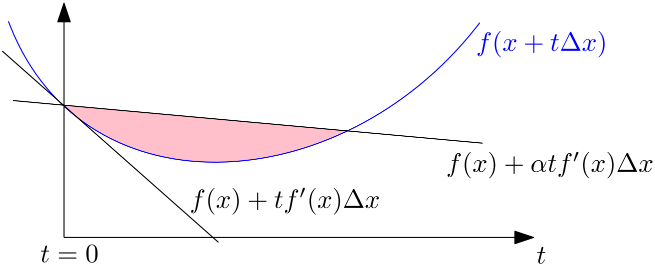


Newton's Method: 1D Case

- In order to ensure convergence, an initial stage using *backtracking line search* is needed. It uses two parameters $0 < \alpha < 0.5$ and $0 < \beta < 1$.

Pseudocode

```
procedure BACKTRACKING LINE SEARCH(f, x, Δx)
    t ← 1
    while f(x + tΔx) > f(x) + αtf'(x)Δx do
        t ← βt
    return t
```



Newton's Method: 1D Case

Pseudocode

procedure 1D NEWTON(f, x, ε)
    while true **do**
        $\Delta x_{nt} \leftarrow -f'(x)/f''(x)$ ▷ Newton step
        $\lambda^2 \leftarrow f'(x)^2/f''(x)$ ▷ Newton decrement
        if $\lambda^2/2 \leqslant \varepsilon$ **then** ▷ stopping criterion
            return x
        $t \leftarrow$ BACKTRACKING LINE SEARCH($f, x, \Delta x_{nt}$)
        $x \leftarrow x + t\Delta x_{nt}$ ▷ update

Newton's Method: 1D Case

- Backtracking line search avoids overshooting.
- It can be proved that two stages occur:
- In the initial stage, backtracking yields $t < 1$.
- After this initial stage, it remains at $t = 1$ indefinitely.
- Newton's method converges very quickly: After the initial stage, the number of correct digits roughly *doubles* at each step.

Newton's Method: n-Dimensional Case

- The *gradient* ∇f and the *Hessian* $\nabla^2 f$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ are

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix} \qquad \nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

- The Hessian is often denoted H.
- The order-2 Taylor approximation of f at x is

$$\hat{f}(x+v) = f(x) + \nabla f(x)^T v + \frac{1}{2} v^T \nabla^2 f(x) v \tag{1}$$

Newton's Method: *n*-Dimensional Case

Pseudocode

procedure NEWTON(f, x, ε)
  while true **do**
    $\Delta x_{nt} \leftarrow -\nabla^2 f(x)^{-1} \nabla f(x)$ ▷ Newton step
    $\lambda^2 \leftarrow \nabla f(x)^T \nabla^2 f(x)^{-1} \nabla f(x)$ ▷ Newton decrement
    if $\lambda^2/2 \leqslant \varepsilon$ **then** ▷ stopping criterion
      return x
    $t \leftarrow$ BACKTRACKING LINE SEARCH$(f, x, \Delta x_{nt})$
    $x \leftarrow x + t\Delta x_{nt}$ ▷ update

Pseudocode

procedure BACKTRACKING LINE SEARCH(f, x, Δx)
  $t \leftarrow 1$
  while $f(x + t\Delta x) > f(x) + \alpha t \nabla f(x)^T \Delta x$ **do**
    $t \leftarrow \beta t$
  return t

Newton's Method: *n*-Dimensional Case

- Same interpretation as in 1D: We move towards the minimum of the order-2 approximation.
- Backtracking line search is done a long the direction of the Newton step.
- Analysis also shows very fast convergence in terms of number of steps.
- However each step can be expensive: Inversion of the Hessian.
  - In practice, the Newton step can be approximated faster.
- It cannot be used for linear programming, because LPs are *constrained* optimization problems.

Logarithmic Barrier Method

- We want to solve the following LP:

$$\begin{array}{lll} \text{minimize} & \displaystyle\sum_{j=1}^{n} c_j x_j & \\ \text{subject to} & \displaystyle\sum_{j=1}^{n} a_{ij} x_j \leqslant b_i, & i = 1, \dots, m. \end{array}$$

- Using the matrix $A = (a_{ij})$ and the vectors $b = (b_i)$, $c = (c_j)$, and $x = (x_j)$, it can be written:

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leqslant b \end{array}$$

Logarithmic Barrier Method

- We denote by a_i^T the ith row of A. Then the LP can also be written

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & a_i^T x \leqslant b_i, & \qquad i = 1, \dots, m. \end{array}$$

- The *indicator function* for the nonpositive reals $I_- : \mathbb{R} \to \bar{\mathbb{R}}$ is defined by:

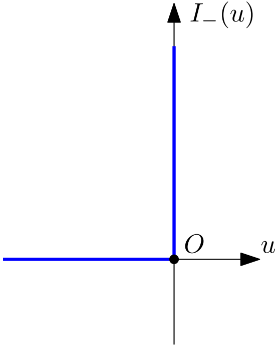
$$I_-(x) = \begin{cases} 0 & \text{if } x \leqslant 0 \\ \infty & \text{if } x > 0 \end{cases}$$

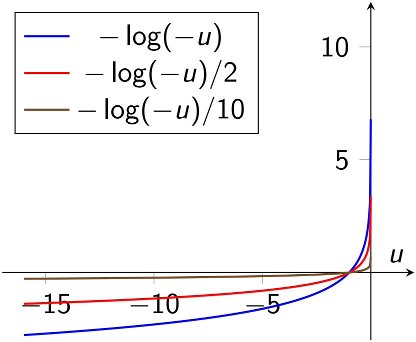
- Then the LP is equivalent to the *unconstrained* minimization problem

$$\text{minimize} \quad c^T x + \sum_{i=1}^{m} I_-(a_i^T x - b_i)$$

Logarithmic Barrier Method

- Problem: We cannot apply Newton's optimization method because I_- is not differentiable.
- So we approximate I_- by a function $\hat{I}_-(u) = -(1/t)\log(-u)$ where t is a parameter that sets the accuracy.





Logarithmic Barrier Method

- So the LP is approximated by the following optimization problem.

$$\text{minimize} \quad c^T x + \sum_{i=1}^{m} -\frac{1}{t} \log(b_i - a_i^T x)$$

where the accuracy of the approximation improves when t increases.

- It is equivalent to

$$\text{minimize} \quad tc^T x - \sum_{i=1}^{m} \log(b_i - a_i^T x).$$

- It becomes an *unconstrained* minimization problem for a twice-differentiable convex function.
- So we can solve it with Newton's method.

Logarithmic Barrier Method

- We write $\Phi(x) = -\sum_{i=1}^{m} \log(b_i - a_i^T x)$, so the problem becomes

$$\text{minimize} \quad tc^T x + \Phi(x).$$

Let $x^*(t)$ be the optimal solution to this problem.

Interpretation

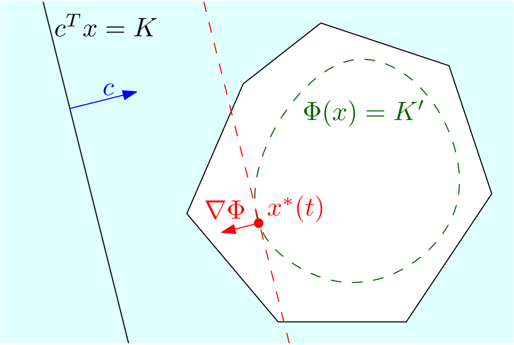
- The term $tc^T x$ pushes towards lower values of the original objective function (because we are minimizing).
- The term $\Phi(x)$ goes to ∞ when a constraint $a_i^T x \leqslant b_i$ becomes tight, which means that we approach the boundary of the feasible region.
- So the term $\Phi(x)$ pushes x inside the feasible region.
- When t increases, the term $tc^T x$ becomes more important, and $x^*(t)$ move towards an optimal solution of the original LP.

Logarithmic Barrier Method

- The gradient of the new objective function is

$$\nabla\left(tc^T x + \Phi(x)\right) = tc + \nabla\Phi(x)$$

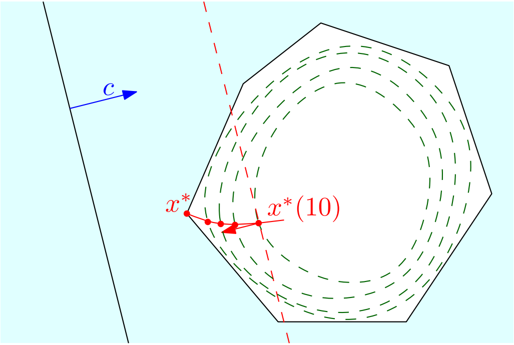
- So we have $tc + \nabla\Phi(x^*(t)) = 0$, and thus $\nabla\Phi$ points towards the opposite direction of c.



Logarithmic Barrier Method

- So the tangent at $x^*(t)$ to the level set containing $x^*(t)$ is parallel to any hyperplane $c^T x =$constant corresponding to the objective function of the LP.

Logarithmic Barrier Method



- When $t \to \infty$, the current solution $x^*(t)$ goes to the solution x^* of the original LP.
- The trajectory of $x^*(t)$ is called the *central path.*

Logarithmic Barrier Method

Pseudocode

procedure $\textsc{Barrier Method}(c, \Phi, \varepsilon, x, t)$
    while true **do**
        $x^*(t) \leftarrow \textsc{Newton}\ (f_t, x, \varepsilon)$ ▷ centering step
        $x \leftarrow x^*(t)$
        if $m/t < \varepsilon$ **then** ▷ stopping criterion
            return x
        $t \leftarrow \mu t$ ▷ increase t

- The input x should be feasible. Similar to the simplex method, we can find it with an auxiliary LP.
- f_t is the objective function for the current unconstrained minimization problem, so $f_t(u) = tc^T u + \Phi(u)$ for all u.
- $\mu > 1$ is a parameter. In practice $\mu = 10$ works well.

Logarithmic Barrier Method

- This algorithm only returns an *approximate solution*: The last point $\hat{x}$ visited by the barrier method satisfies $f(\hat{x}) - f(x^*) \leqslant \varepsilon$.
- So a final step called *termination* is needed, which finds x^* from $\hat{x}$.
- Idea:
  - Pick the n constraints that have the smallest slack.
  - x^* is at the intersection of their bounding hyperplanes.
  - It requires ε to be chosen small enough.
- Finally, we get the main result: (not proved in CSE530)

Theorem

The logarithm barrier method solves linear programming with m constraints in $O(m^3 L)$ time, where L is the total number of bits in the input.

Logarithmic Barrier Method

- The theorem above shows that there is a *weakly polynomial*-time algorithm for LP.
- It means polynomial in the input size, counting the number of bits of all input coefficients a_{ij}, b_i, c_j.

Open Problem

Does there exist a *strongly* polynomial algorithm for Linear Programming, that is, an algorithm whose running time is polynomial in n and m?

- This is one of the most important open problem in computer science.
- Possible approach: Find a pivoting rule for the simplex algorithm that makes a polynomial number of pivots.

Conclusion of this Lecture

- The logarithmic barrier method constructs a sequence of points in the *interior* of the feasible region, which converges to an optimal solution.
- So it is an *interior point* method.
- The simplex algorithm is very different: It walks along the *boundary* of the feasible region.
- Linear Programming can be solved (weakly) in polynomial time.
- So one way of finding a polynomial-time algorithm for a problem is to reduce it to LP. We saw several examples in the introductory lecture to LP.

Conclusion of this Lecture

- In practice, there are very efficient LP solvers, so it is a very useful tool.
- It can also be used to solve hard problems, by approximating the original problem using an LP. It sometimes provides a provable approximation, or it can be used as a heuristic. This is called *LP-Rounding*. (Not covered in CSE530.)
- Interior point methods can also be used for solving *convex optimization* problems: Minimizing a convex function over a convex set. (See reference book.)

Conclusion on first half of CSE530

- Focus was on three techniques to obtain *polynomial-time* algorithms:
  - Dynamic programming
  - Maximum flow
  - Linear programming
- When facing an algorithmic problem, if there is no obvious polynomial-time algorithm, try to reduce it to linear programming, convex programming or maximum flow, or try a direct approach through dynamic programming.
- If it does not work, the problem is likely to be hard (for instance **NP**-hard). Try to prove it through a reduction. (See second half of this course.)

Other Graduate Courses

- I plan to teach *Algorithms Design* in fall 2018
  - It will be another theoretical course on algorithms.
- I taught *Computational Geometry* in fall 2017.
  - Focus on algorithms and data structures for geometric objects such as points, lines, polygons, polytopes, circles lines ...
  - I plan to teach it every 4 years.