2강

Solving optimization Problems

Most of the time we solve it numerically.

→ 대부분 컴퓨터의 힘을 빌려서 푼다.

min x^2 -2x+3

s.t. 0≤x≤2

→ closed-form

이건 딱 떨어지는 답이다.

그리고 대부분의 문제는 고차원 x이다.

x : vector

In general, optimization problems are difficult to solve

What do we mean by ‘difficult’?

For some problems it may take prohibitively long time to solve

Consider how complexity grows with problem size

e.g., Travelling Salesman Problem(TSP)

일반적으로 최적화 문제들은 일반적으로 풀기 어렵다는 것이다.

어렵다는 것에는 두가지 의미가 있다.

1. 답을 구하는 것 자체가 불가능 하다. 알려져 있지 않다.
2. 푸는 방법은 아는데 그것이 너무 오랜시간이 걸리더라.

문제의 사이즈를 크게 했을 때 기하급수적으로 늘어나면 어렵운 문제가 아닌가?

brute force = 전부 일일이 나열

Good news: convex optimization problems can be solved efficiently!

→ 볼록최적회문제는 n에 대해 polynomial하다

Why study convex optimization?

How complex is an optimization problem?

If the problem is convex, it can be efficiently solved (most cases)

- linear program, quadratic program, geometric program
- linear regression, logistic regression, support vector machine,...

모든 convex가 다 쉽게 풀리는 것은 아니지만 대부분이 쉽게, resonable하게 풀리더라

If the problem is nonconvex, it is typically difficult

- integer optimization, combinatorial optimization problems:

  NP-hard problems are nonconvex problems

  integer optimization → x를 정수로 넣으면서(constrain) 답을 찾는 것

  combinatorial optimization → 일일이 경우의 수를 세는 경우
- traveling salesman problem, deep neural networks

→ nonconvex이면 대부분이 difficult이다.

deep neural networks 이것을 최적화 시키는 것도 nonconvex하다.

→ 정확한 답이 아니라, 충분히 의미있는 답을 찾으면 공학적인 관점으로 답이라고 말하는 것임

Determining problem complexity is the very first step!

So knowing whether a problem is convex optimization

is the starting point

→ 이것은 공학적으로 중요한 부분이다.

Optimization: Mathematical Definition

Standard form of optimization problems

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0,\ i = 1, \ldots, m. \end{array}$$

- $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ (optimization variable)
- $f_0(x) : \mathbb{R}^n \to \mathbb{R}$ (objective function)
- $f_i(x) : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m$ (constraints)

maximize와 minimize는 사실 상 같은 문제이다.

maximize x^2 == minimize -x^2

목적함수에 마이너를 붙여버리면 그냥 같은 거다. (reformulation라는 것에 불과하다.)

maximize f_0(x) == minimize -f_0(x)

m개의 constraint가 동시에 만족되게 해야한다.

0≤x≤2를 다음과 같이 쓸 수 있다.

f1(x) = -x

f2(x) = x-2

f1(x) ≤ 0, f2(x) ≤ 0

- x∗ is called a minimizer

  if the objective function achieves the minimum value at x∗,

  i.e. f0(x∗) ≤ f0(x) for all x satisfying the constraint. → 제약조건을 만족시키는 x중에서 목적함수를 가장 작게 만드는 ㅌrkqt

x*는 최적화 문제의 solution이다.

What is convex optimization?

$$\begin{array}{ll} \text{minimize} & f_0(x) \\ \text{subject to} & f_i(x) \leq 0,\ i = 1, \ldots, m. \end{array}$$

- $f_0(x)$ (objective function) and $f_i(x) : \mathbb{R}^n \to \mathbb{R},\ i = 1, \ldots, m$ (constraints) are *convex* functions

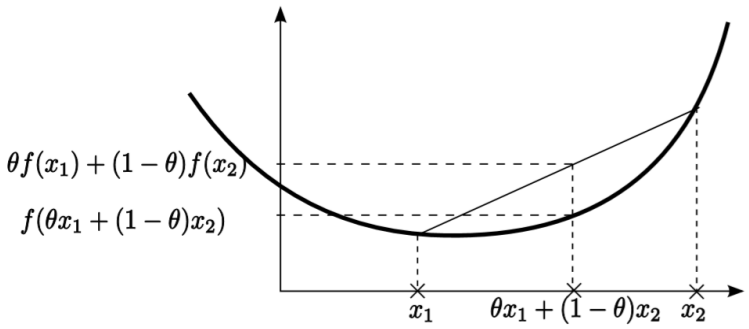
모양은 완전히 같아보이는데...

f0, f1, f2의 모든 함수들이 볼록함수일 때 이 문제는 convex optimization 문제가 된다.

볼록함수란 아래로 볼록한 함수를 말하는 것이다.

- $f(x) : \mathbb{R}^n \to \mathbb{R}$ is convex if, for any x_1 and x_2,

$$f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + (1-\theta) f(x_2),\ \theta \in [0, 1]$$



아무거나 곡선의 두점을 잡았을때 그 두 점의 선분이 그 곡선보다 위에 있다.라는 조건이다.

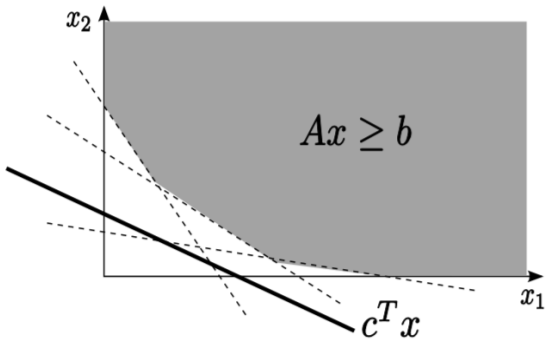
Example: Linear Programming (LP)

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

- $x \in \mathbb{R}^n$ is the optimization variable
- $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$
- Constraints and objectives are *linear* functions in x

두 백터의 내적을 최소화하는 것이다. 그 직선을 최소화하는 것이다.

그리고 제약조건은 1차함수들로 이루어져있다.



solution은 그 평면의 꼭짓점과 직선이 가장 아래에 만나는 지점일 것이다.

이것은 볼록 최적화가 맞다. 왜냐하면 여기에 나오는 모든 함수가 1차 함수들이고,

1차 함수들은 볼록 함수이다.

그 모양이 두점 선분 위로만 안 가면 된다.

- LP looks simple, but there is no analytical solution in general

→ 이 LP이 만만해 보이지만 그렇지 않다.

복잡도가 상당하다.

- Very large number of problems can be converted into LP,

  although it may not seem obvious at all.

→ 그런데 생각보다 많은 문제들이 LP카테고리 안에 들어있다.

LAPACK이라는 라이브러리로 LP는 풀 수가 있더라

variable들을 정리하여 라이브러리에 딱 넣어버리면 해결된다.

Example: Least-Squares problem

- Standard-form LS problem:

$$\boxed{\text{minimize} \quad ||Ax - b||_2^2}$$

- $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$

→ 벡터의 크기를 구하는 것이다. (norm)

→ constraint는 없다.

볼록함수의 최고차항의 계수는 양수여야 한다.

차이를 구하겠다는 것이다. 그 차이를 가장 줄여주는 x를 구하겠다는 것이다.

- Has the analytical solution (이건 실재로 풀 수가 있다.)

$$x^* = (A^T A)^{-1} A^T b$$

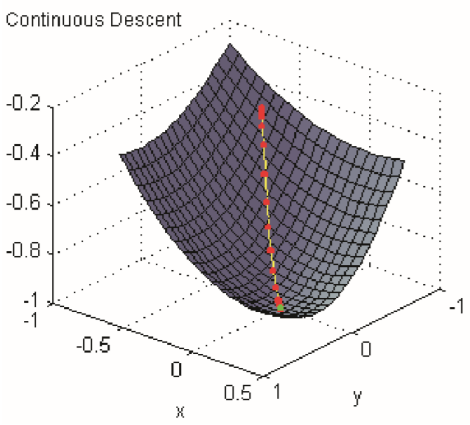
- Complexity governed by matrix inversion → 역행렬 때문에 실재로 수학적인 계산이 어렵다.
- Usually the objective function models some sort of errors e.g., least-square linear regression.

  → 그래서 linear regression을 사용한다.

Optimization algorithms

- A number of iterative algorithms are known and well established in theory

  LS처럼 구하면 쉬울 수 있지만, 그게 안되면 numerical하게 한다. solution에 근접하게 되는 것이다.
- Algorithms are efficient and fast, most of the time these algorithms exhibit exponential convergence to the solution (그 알고리즘이 잘 알려져 있다.)
- Based on known algorithms, we may develop algorithms with faster convergence exploiting special problem structures, for example A can be sparse in LP constraint



한걸음 한걸음 가장 가파른 곳을 찾아서 조금씩 움징이는 것이다.

iterative algorithm은 이것을 반복하는 것이다.

→ 딥러닝에 사용하는 것이 다 이런식이다.

Procedures of problem solving and goals

In general, the process of problem solving will be...

- Formulate problems of interest, find objectives and constraints

  → 문제를 수립한다. objectives and constraints을 만든다.

- Identify whether your problem is convex optimization

  → convex냐 아니냐 판단한다.

- If the problem is not convex, try to transform the problem into a convex optimization.

  → convex가 아니면convex로 한번 다시 만들어 볼 수도 있다.

  Or just find a suboptimal (maybe sufficiently good) solution

  → 대부분 적당히 좋은 답을 찾아보자는 것이다. suboptimal 은 practical하게 적용 가능한 solution을 찾자는 것이다. 아니면 suboptimal 이 optimal에 어느정도 가까이 있다고 까지 말할 수도 있다.

- Otherwise attempt to approximate the problem as convex optimization

  → 그것도 안되면 그것과 근사한 문제를 풀자는 것이다.

- Once convex problem has been formulated, use known algorithms to solve it. Devise faster algorithms if the problem has special structure (like sparsity).

  → 만일 볼록최적화 문제라면 알려진 알고리즘을 이용하여 풀자는 것이다.

  아니면 알고리즘을 개발해보자.