10-725/36-725: Convex Optimization **Fall 2019**

Lecture 5: September 11

Lecturer: Ryan Tibshirani *Scribes: Shreyan Bakshi, Jeremy Cohen, Austin Dill*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

5.1 Canonical Convex Problems

In the previous lecture, we discussed the basic terminology and foundation of convex optimization problems. We ended after introducing the Linear Program as a canonical convex problem.

5.1.1 Example: Basis Pursuit

Given $y \in \mathbb{R}^n$ and $X \in \mathbb{R}^{n \times p}$, where $p > n$. Suppose we want to find a solution to the linear system $X\beta = y$. Because $p > n$, this system is undetermined so there are many solutions. One option is to seek the *sparsest* possible solution.

A nonconvex formulation of this problem is the following.

$$\begin{aligned} \min_{\beta} \quad & ||\beta||_0 \\ \text{subject to} \quad & X\beta = y \end{aligned}$$

We will instead consider the convex relaxation of this problem known as *Basis Pursuit*.

$$\begin{aligned} \min_{\beta} \quad & ||\beta||_1 \\ \text{subject to} \quad & X\beta = y \end{aligned}$$

It can be shown that this is a linear program by reformulation.

$$\begin{aligned} \min_{\beta, z} \quad & 1^T z \\ \text{subject to} \quad & z \geq \beta \\ & z \geq -\beta \\ & X\beta = y \end{aligned}$$

5.1.2 Example: Dantzig Selector

Assume that the setup is the same as in the previous example, but we don't have exact equality. This means $X\beta \approx y$. This leads to another linear program known as the *Dantzig Selector* with tuning parameter λ.

$$\begin{aligned} \min_{\beta} \quad & ||\beta||_1 \\ \text{subject to} \quad & ||X^T(y - X\beta)||_\infty \leq \lambda \end{aligned}$$

5.1.3 Standard Form for LPs

A common format for expressing linear programs is the *standard form.*

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

All linear programs can be written in standard form.

5.1.4 Convex Quadratic Programs

A quadratic program or QP is an optimization problem of the form:

$$\begin{aligned} \min_{x} \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{subject to} \quad & Dx \leq d \\ & Ax = b \end{aligned}$$

Note that this is only convex if $Q \succeq 0$. We will always assume that QPs refer to convex QPs.

5.1.5 Example: LASSO

There are two forms for the familiar LASSO problem. Give $y \in \mathbb{R}^n, X \in \mathbb{R}^{n \times p}$, we can write the constrained form of the LASSO as follows:

$$\begin{aligned} \min_{\beta} \quad & ||y - X\beta||_2^2 \\ \text{subject to} \quad & ||\beta||_1 \leq s \end{aligned}$$

We can also write this in the so called Langrangian or penalized form.

$$\min_{\beta} \quad \frac{1}{2}||y - X\beta||_2^2 + \lambda||\beta||_1$$

5.1.6 Standard Form for QPs

The standard form for QPs is written as

$$\begin{aligned} \min_{x} \quad & c^T x + \frac{1}{2} x^T Q x \\ \text{subject to} \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

All quadratic programs can be written in this form.

5.1.7 Semidefinite Programs

One can motiate Semidefinite Programs (also known as SDPs) by generalizing LPs. For example, we will come up with a different partial order ($\succeq$) as well as an analogous inner product.

In order to do this, we will first review some facts about symmetric matrices.

Let $\mathbb{S}^n$ denote the space of $n \times n$ symmetric matrices. $\mathbb{S}^n_+$ will denote the space of positive semidefinite matrices and $\mathbb{S}^n_{++}$ the space of positive definite matrices.

Properties of Symmetric Matrices:

1. $X \in \mathbb{S}^n$ implies that all of the eigenvalues of X are real numbers.
2. $X \in \mathbb{S}^n_+$ implies that all of the eigenvalues of X are nonnegative real numbers.
3. $X \in \mathbb{S}^n_{++}$ implies that all of the eigenvalues of X are positive real numbers.
4. We can define an inner product over $\mathbb{S}^n$: $X \bullet Y = \operatorname{tr}(XY)$.
5. The partial ordering we discussed earlier is provided by $X \succeq Y \iff X - Y \in \mathbb{S}^n$

Now that we've laid the groundwork we can define SDPs. An SDP is a convex optimization problem of the following form:

$$\begin{aligned} \min_{x} \quad & c^T x \\ \text{subject to} \quad & x_1 F_1 + \cdots + x_n F_n \preceq F_0 \\ & Ax = b \end{aligned}$$

where $\forall j, F_j \in \mathbb{S}^d$, $A \in \mathbb{R}^{m \times n}$, $c \in \mathbb{R}^n$, and $b \in \mathbb{R}^m$. It follows from this definition that any linear program is an SDP.

5.1.8 Standard Form for SDPs

SDPs can also be written in standard form, which is derived from LPs with our analogous inner product and partial ordering.

$$\begin{aligned} \min_{X} \quad & C \bullet X \\ \text{subject to} \quad & A_i \bullet X = b_i, i = 1, \ldots, m \\ & X \succeq 0 \end{aligned}$$

5.1.9 Example: Lovasz Theta Function

Let $G = (N, E)$ be an undirected graph, where $N = \{1, \ldots, n\}$ are the vertices and $E \subset N \times N$ are the edges. The *clique number* of G, denoted $\omega(G)$, is the size of the largest clique in G. (Recall that a clique is a set of vertices that are all connected to each other by edges.) The *chromatic number* of G, denoted $\chi(G)$, is the smallest number of colors needed to color the vertices of G such that no connected vertices have the same color. Computing both of these quantities is NP-complete. Interestingly, however, one can compute in polynomial time (by solving an SDP) a value that is sandwiched between $\omega(G)$ and $\chi(G)$.

Define $\vartheta(G)$ as the solution to the following SDP:

$$\begin{aligned} \vartheta(G) = \max_X \quad & 11^T \bullet X \\ \text{subject to} \quad & I \bullet X = 1 \\ & X_{ij} = 0, \ (i,j) \notin E \\ & X \succeq 0 \end{aligned}$$

Lovasz proved that $\vartheta(\bar{G})$ — the theta function of $\bar{G}$, the complement graph of G — is sandwiched in between the chromatic number and the clique number of G:

$$\omega(G) \leq \vartheta(\bar{G}) \leq \chi(G).$$

5.1.10 Example: Trace Norm Minimization

Define the linear map $A : \mathbb{R}^{m \times n} \to \mathbb{R}^p$ as:

$$A(X) = \begin{bmatrix} A_1 \bullet X \\ \vdots \\ A_p \bullet X \end{bmatrix}$$

with $A_1, \ldots, A_p \in \mathbb{R}^{m \times n}$.

Consider finding the minimum-rank solution of the underdetermined linear system:

$$\begin{aligned} \min_X \quad & \text{rank}(X) \\ \text{subject to} \quad & A(X) = b \end{aligned}$$

where $b \in \mathbb{R}^p$.

This problem is non-convex and hard to solve. Therefore, a popular approximation is to instead minimize the trace of the matrix:

$$\begin{aligned} \min_X \quad & \|X\|_{\text{tr}} \\ \text{subject to} \quad & A(X) = b \end{aligned}$$

where $\|X\|_{\text{tr}}$ is the *trace norm* of the matrix X, defined as the sum of X's singular values. This problem can be shown to be an SDP.

5.1.11 Conic Programs

A conic program is an optimization problem of the form:

$$\begin{aligned} \min_x \quad & c^T x \\ \text{subject to} \quad & Ax = b \\ & D(x) + d \in K \end{aligned}$$

where $D : \mathbb{R}^n \to Y$ is a Euclideean map into some Euclidean space Y, $d \in Y$ and $K \subseteq Y$ is a closed, convex cone.

Both LPs and SDPs are conic programs. In linear programming, the cone is $K = \mathbb{R}^n_+$, the non-negative orthant. In semidefinite programming, the cone is $K = \mathbb{S}^n_+$, the cone of PSD matrices.

5.1.12 Second-order cone programs

One notable special case of conic programs are the *second-order cone programs*, defined as

$$\begin{aligned} \min_x \quad & c^T x \\ \text{subject to} \quad & \|D_i x + d_i\|_2 \le e_i^T x + f_i, \quad i = 1, \ldots, p \\ & Ax = b \end{aligned}$$

SOCP's are conic programs To see that this is indeed a conic program, recall the second-order cone

$$Q = \{(x, t) : \|x\|_2 \le t\}$$

Thus, the constraint $\|D_i x + d_i\|_2 \le e_i^T x + f_i$ can be viewed as a cone constraint,

$$(D_i x + d_i, e_i^T x + f_i) \in Q_i.$$

Therefore, the overall SOCP is a conic program with cone $K = Q_1 \times Q_2 \times \ldots \times Q_p$.

SOCPs are SDPs Every SOCP is actually an SDP, since the second-order cone constraint $(x, t) : \|x\|_2 \le t$ can be formulated as the semidefinite constraint

$$\begin{bmatrix} tI & X \\ x^T & t \end{bmatrix} \succeq 0.$$

QPs are SOCPs Quadratic programs are a subclass of SOCPs. To see this, rewrite the canonical QP program in the following form by introducing a variable t:

$$\begin{aligned} \min_{x,t} \quad & c^T x + t \\ \text{subject to} \quad & Dx \le d, \ \frac{1}{2} x^T Q x \le t \\ & Ax = b \end{aligned}$$

The constraint $\frac{1}{2} x^T Q x \le t$ is a second-order cone constraint:

$$\frac{1}{2} x^T Q x \le t \iff \left\| \left(\frac{1}{\sqrt{2}} Q^{1/2} x, \frac{1}{2}(1 - t) \right) \right\|_2 \le \frac{1}{2}(1 + t)$$

5.1.13 Convex Programming Hierarchy

We have established the following hierarchy of convex program classes:

$$\text{LPs} \subset \text{QPs} \subset \text{SOCPs} \subset \text{SDPs} \subset \text{conic programs}$$

5.2 Gradient Descent

Here, we talk about methods that only use first-order information (i.e., the gradient). Later in the course, we'll also see second-order methods.

Let's consider the unconstrained, smooth convex optimization

$$\min_x f(x)$$

We **assume** a few thing about the function f:

- f is convex and differentiable
- $\text{dom}(f) = \mathbb{R}^n$, i.e., it has full domain
- We also assume here, like everywhere else in the course, that a solution exists (there are convex problems that get minimized out in infinity, but we assume we aren't in that case).

  Under this assumption, we denote the optimal criterion value by

  $$f^* = \min_x f(x)$$

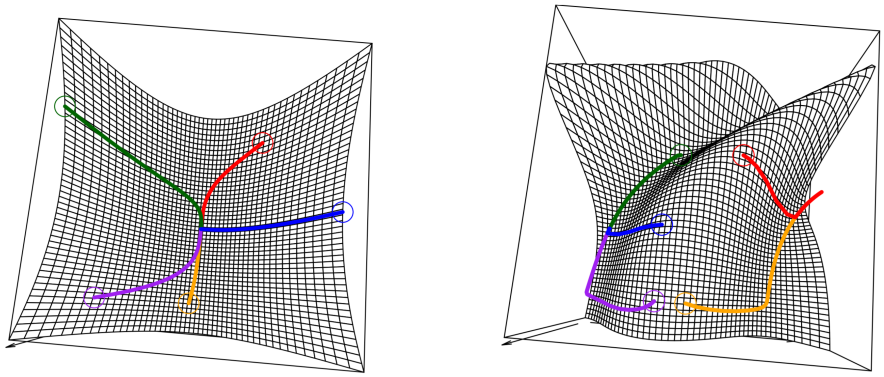
  with the solution at x^*.

The **Gradient Descent algorithm** is then defined as follows:

1. Choose an initial point $x^{(0)} \in \mathbb{R}^n$
2. Repeat:
   $$x^{(k)} = x^{(k-1)} - t_k \cdot \nabla f(x^{(k-1)}), \quad \text{for } k = 1, 2, 3, \ldots$$
3. Stop at some point

Above, after choosing some initial point $x^{(0)}$, we move it in the direction of the negative gradient (this points us in a direction where the function is decreasing) by some positive amount t_1, calling this x_1. And the same process is repeated.

We see some examples of Gradient Descent below:



(a) Convex case (b) Non-Convex case

Figure 5.1: Gradient Descent on convex and non-convex functions

In figure (a), a convex function in 2 dimensions is depicted. The axes represent x_1, x_2, and the mesh the values of the criterion. The minimum of the function is at the center, and the paths iterated when we run gradient descent starting at different points are depicted, all of them converging to the same (global) minima.

In figure (b), we run gradient descent on a non-convex problem in 2 dimensions. As we can see, where we start affects where we end up.

The global minimum is achieved at the right end of the figure (where the red and yellow lines meet), but due to the presence of a stationary point (zero derivative) on the left side, running gradient descent from some other initial points (blue, green, violet) could converge there instead (sub-optimal).

5.2.1 Interpretation

The second-order Taylor expansion of f gives us

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2} \, (y - x) \, \nabla^2 f(x) \, (y - x)$$

Consider the Quadratic approximation of f, replacing $\nabla^2 f(x)$ by $\frac{1}{t} I$ (replacing the curvature given by the Hessian with a much simpler notion of curvature – something spherical), we have

$$f(y) \approx f(x) + \nabla f(x)^T (y - x) + \frac{1}{2t} \, \|y - x\|^2$$

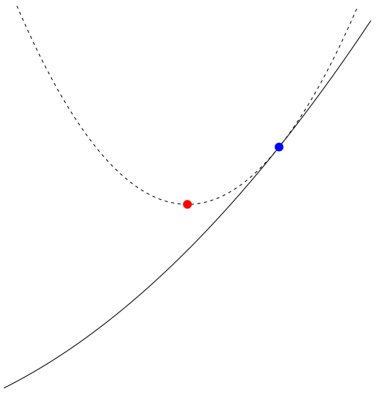
This is a convex quadratic, so we know we can minimize it just by setting its gradient to 0.

Minimizing this w.r.t. y, we get

$$\begin{aligned} \frac{\partial f(y)}{\partial y} &\approx \nabla f(x) + \frac{1}{2}(y - x) = 0 \\ \implies y &= x - t \, \nabla f(x) \end{aligned}$$

This gives us the gradient descent update rule. In other words, gradients descent actually chooses the next point to minimize this overall y.

Figure 5.2 shows pictorially the interpretation. The dotted function shows the quadratic approximation, and the red dot shows the minima of the quadratic approximation.



Blue point is x, red point is

$$x^+ = \underset{y}{\operatorname{argmin}} \; f(x) + \nabla f(x)^T(y-x) + \frac{1}{2t}\|y-x\|_2^2$$

Figure 5.2: Minimizing the Quadratic Approximation

Starting at the blue point, minimizing the quadratic approximation takes us to the red point, so we move to the point on the curve directly below the red point.

We can also think of the Quadratic approximation as a sum of 2 terms:

- A linear approximation to f given by $f(x) + \nabla f(x)^T(y-x)$
- A proximity term to x given by $\|y-x\|^2$, with weight $\frac{1}{2t}$.

Of course, if we tried to minimize a linear approximation to our function that wouldn't be very useful – we'd send y off to infinity in some direction. So the proximity term keeps us close to x. It has weight $\frac{1}{2t}$, so if t is very small we're going to stay close to x due to the high weight, and vice versa.

5.2.2 How to choose step sizes

An important aspect of gradient descent is pickings the step size. Below are some methods via which we can do this.

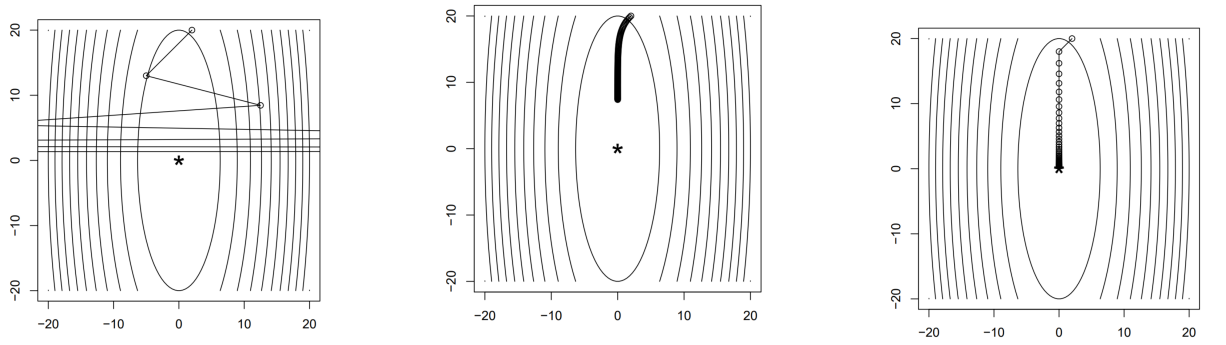
5.2.2.1 Fixed step size

The simplest strategy is to take the step sizes to be fixed. So we choose $t_k = t, \quad$ for all $k = 1, 2, 3, \ldots$

There are some problems with doing this:

- If t is **too large**, gradient descent can diverge.
- If t is **too small**, gradient descent can be (very) slow to converge.

Functions tend to converge nicely only when t is "**just right**", as we can see below.



(a) Too large, result after 8 steps (b) Too small, result after 100 steps steps (c) Just right, result after 40 steps

Figure 5.3: Gradient Descent with different step sizes

The figures above show contours of the function $f(x) = \left(10{x_1}^2 + {x_2}^2\right)/2$, along with the results of running gradient descent with different step sizes on it. Convergence analysis later will give us an idea of "just right".

5.2.2.2 Backtracking Line Search

An alternative which is very popular in practice is to use adaptive step sizes – not just something fixed, but instead trying to guess the right step size at every iteration via some heuristic.

The most popular such heuristic is called Backtracking Line Search. This works as follows:

1. First, fix parameters $0 < \beta < 1$ and $0 < \alpha \leq \frac{1}{2}$

2. At each iteration (of gradient descent), start with $t = t_{\text{init}}$ (something relatively large), and while

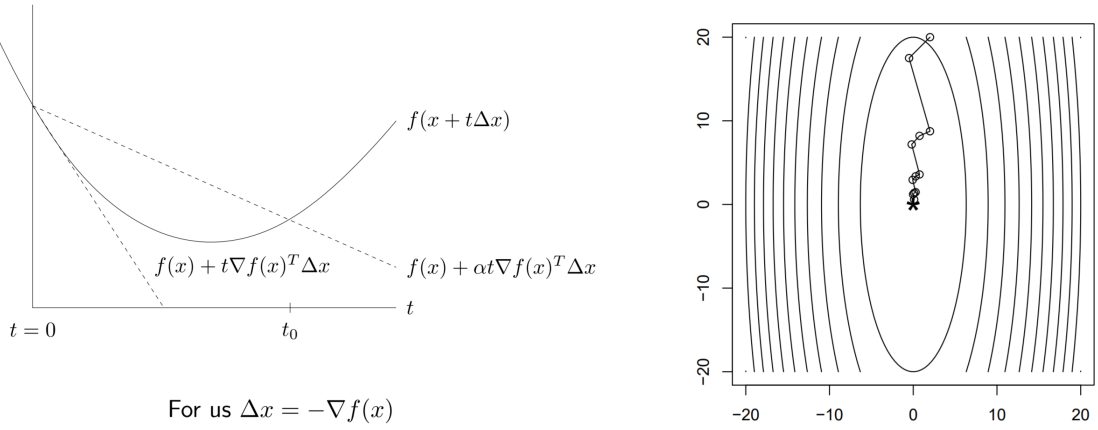
$$f\left(x - t\nabla f(x)\right) > f(x) - \alpha t \|\nabla f(x)\|_2^2$$

   shrink $t := \beta t$. Else, perform the gradient descent update

$$x^+ := x - \ t\ \nabla f(x)$$

The update criterion above denotes that, if the progress we make by going from x to $x - t\nabla f(x)$ is bigger than the progress we had $f(x) - \alpha t\|\nabla f(x)\|_2^2$, then we make t smaller (βt).

So by shrinking t, we move from right to left on the graph, as long as the line is beneath the function, and then stop when it exceeds it.



(a) Intuition

(b) In action

Figure 5.4: Backtracking Line Search

As we see in figure (b), backtracking (with $\alpha = \beta = 0.5$) picks up roughly the right step size (12 outer steps, 40 steps total).

5.2.2.3 Exact Line Search

We could also choose a step to do the best we can along the direction of the negative gradient, called Exact Line Search:

$$t = \underset{s \geq 0}{\operatorname{argmin}} f(x - s\nabla f(x))$$

x is fixed above (its the point we're currently at during gradient descent), and we find the value of s that allows us to do the absolute best we can along that line segment.

Trying to make this as small as possible over all s is a 1-dimensional optimization problem. In principle, we might think its a good idea to optimize this.

But this is usually **not** possible to do directly; and approximations to the same are not as efficient as general backtracking. So is **not** worth the effort (except for fairly simple problems maybe).

5.2.3 Convergence Analysis

5.2.3.1 Gradient Descent Convergence

We'll now state a convergence result for Gradient Descent that is going to be used as a basis for all the comparisons we make in future lectures, with different algorithms.

In general, a Convergence Rate tells us how quickly an algorithm converges.

We assume that f is convex and differentiable, with $\text{dom}(f) = \mathbb{R}^n$, and additionally

$$\|\nabla f(x) - \nabla f(y)\|_2 \leq L\|x - y\|_2 \text{ for any } x, y$$

The above is equivalent to saying that, if f has two derivatives, the largest eigenvalue of the Hessian of f is at most L.

Then, the following theorem holds:

Theorem: Gradient descent with fixed step size $t \leq 1/L$ satisfies

$$f(x^{(k)}) - f^\star \leq \frac{\|x^{(0)} - x^\star\|_2^2}{2tk}$$

and same result holds for backtracking, with t replaced by β/L

Figure 5.5: Gradient Descent Convergence

In words, the criterion value at step k minus the optimal value is upper bounded by the squared distance between where we started and **a** solution x^* (statement holds for all solutions if there's more than one) divided by $2tk$, where t ($\leq 1/L$) is the step size.

$t = 1/L$ is the biggest step size allowed here.

The same result holds if we use backtracking – we just need to replace t by β/L.

We say that gradient descent has convergence rate $O(1/k)$, i.e., it finds ϵ-suboptimal point in $O(1/\epsilon)$ iterations. We read this by saying that after k iterations, the gap between the criterion and where we are goes down by $1/k$.

5.2.3.2 Gradient Descent Convergence Strong Convexity

What happens when we have more information, i.e. that f is strongly convex ($f(x) - \frac{m}{2}x^2$ is convex for some $m > 0$), instead of just convex?

Assuming Lipschitz gradient as before, along with strong convexity, the following theorem holds:

Theorem: Gradient descent with fixed step size $t \leq 2/(m + L)$ or with backtracking line search search satisfies

$$f(x^{(k)}) - f^\star \leq \gamma^k \frac{L}{2}\|x^{(0)} - x^\star\|_2^2$$

where $0 < \gamma < 1$

Figure 5.6: Gradient Descent Convergence Strong Convexity

$t = 2/(m + L)$ is the biggest step size allowed here.

Gradient Descent convergence rate under strong convexity is $O(\gamma^k)$, i.e., it finds ϵ-suboptimal point in $O(\log(1/\epsilon))$ iterations. Exponentially fast!

If we want converge to a small ϵ guarantee, this is a lot faster because of the logarithm involved.