Continuing the hierarchy from last time.

B5. Conic programs

- most general form of opt problem
- LPs and SDPS are forms of conic programming

A conic program is an optimization of the form

$min\ c^T x$
$such\ that$
$Ax\ =\ b$
$D(x)\ +\ d\ \in K$

i.e. some linear map of X + some d is in a closed convex cone K

Homework 1 uses CVX - should learn how to use this!

B6. Second-order cone program (SOCP)

- The cone has a very specific form

$min\ c^T x$
$such\ that$
$||D_i x\ +\ d_i||_2\ \leqslant\ e_i^T x\ +\ f_i,\ i = 1,\ ...,\ p$

Q: Why is this convex?

1. the objective function is convex
2. the constraint is convex
    - LHS: L2 norm of affine function is convex
    - RHS: is convex and concave. You can bring it to the LHS, negate it, to get something of the form convex function $\leqslant 0$, which fits the spec for a convex program

Q. Why is this a conic program?

Recall the second-order cone

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

Here we have $||D_i x\ +\ d_i||_2\ \leqslant\ e_i^T x\ +\ f_i\ <=>\ \left(D_i x\ +\ d_i, e_i^T x\ +\ f_i\right)\ \in Q$

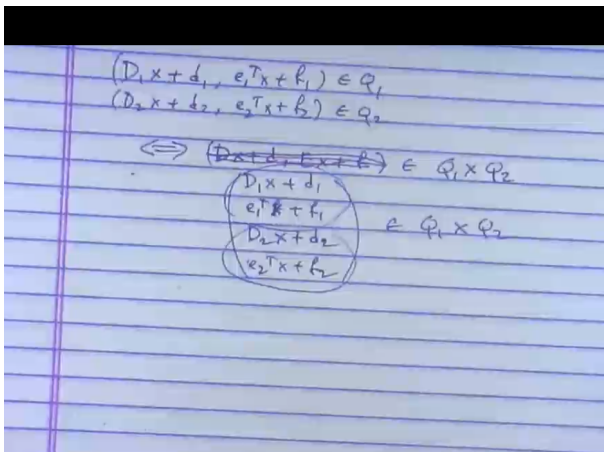
for second-order cone Q of appropriate dimension, for each i.

Now stack together these cones = cartesian product of cones

- Cartesian product of sets $A\ x\ B\ =\ \{(a,\ b):\ a \in A,\ b\ \in B\}$

So then if $D_1 x\ +\ d_1, e_1^T x\ +\ f_1\ \in Q_1,\ D_2 x\ +\ d_2, e_2^T x\ +\ f_2\ \in Q_2$

then $D_i x\ +\ d_i, e_i^T x\ +\ f_i\ \in\ Q_1\ x\ Q_2$



C. Proving the hierarchy: LP < QP < SDP < Conic

Observe that every LP is a SOCP.
Why?

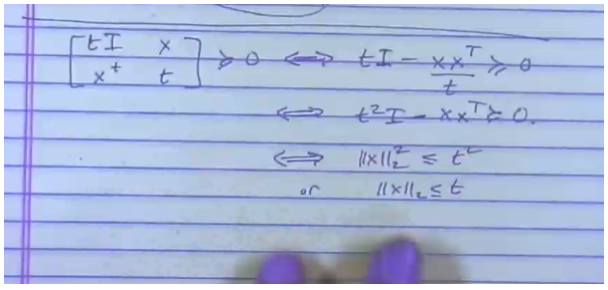
- set D_i matrix to 0, d_i to be zero
- Make it such that you compare $x_i \leqslant 0$ - standard form of LP

But every SCOP is also an SDP. And later we will see that every QP is also an SDP.
SOCPs are a tool to from from QP to SDP, i.e. we will use it to show that QP < SDP.

Fact 1. Every SOCP is an SDP.
Proof: Recognize that the norm constrain on x $||x||_2 \leqslant t$ is equivalent to the constraint
$\begin{bmatrix} tI & x \\ x^T & t \end{bmatrix} \succcurlyeq 0$
This comes from Schur complement theorem

- This theorem tries to to check if a block matrix is PSD
- $\begin{bmatrix} A & B \\ B^T & C \end{bmatrix} \succcurlyeq 0 \iff A - BC^{-1}B^T \succcurlyeq 0$

In this instance:



So far we know:
1. LP < QP

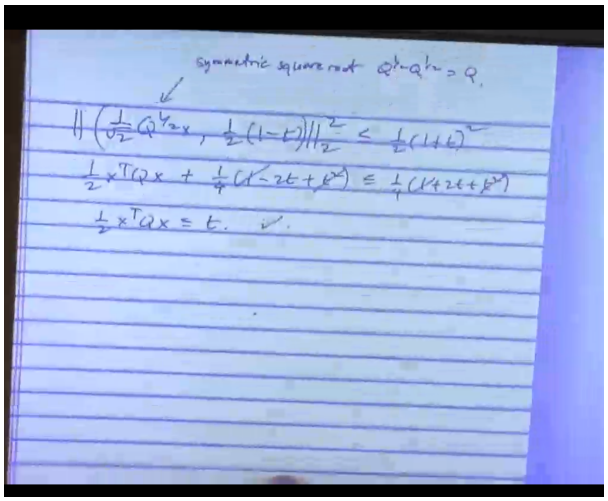
2. LP < SCOP < SDP < Conic
We need to figure out where QPs sit in the second hierarchy.

Turns out, QPs are SOCPs, which we can see by rewriting a QP as

$$min\ c^T x\ +\ t$$
$$such\ that$$
$$Dx\ \leqslant\ d,\ \frac{1}{2}x^T Qx\ \leqslant t$$

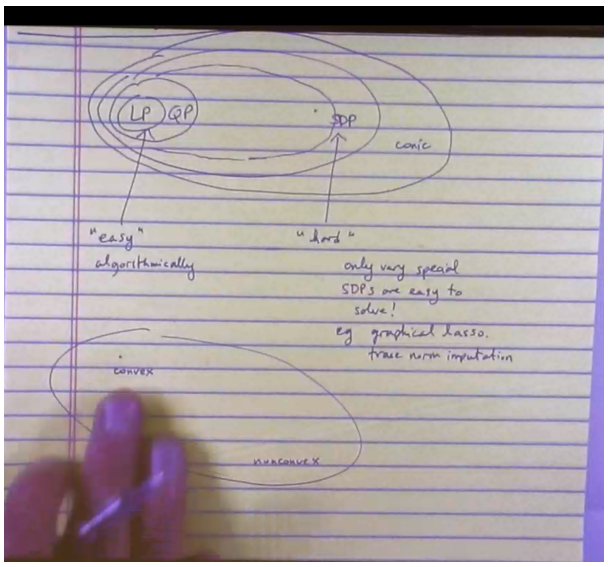
Q: To make this a SOCP, need to show that these constraint are second order cone constraints.

1. We know how to write linear constraints as so cone constraints
    - How?
2. But how do we write the quadratic constraint as a so cone constraint?
    - There is an equivalence here that is non intuitive!
    - A quadratic in x being $\leqslant$ t ≡ a particular norm of a linear fn of x and t being $\leqslant$ another linear funcion
    -



So LP < QP < SOCP < SDP < Conic

"Non-rigourous opinions on how easy/hard these problems are"



Algorithms to solve convex optimization problems(!)

First order methods

A. Gradient Descent

Consider unconstraint smooth convex optimization

$min\ f(x)$

$such\ that$

$f\ is\ convex\ and\ differentiable$

$dom(f)\ =\ R^n$

Let the optimal criterion value by $f^*\ =\ min_x f(x)$ and the solution by x^*

Gradient descent idea:

Choose initial point $x^{(0)} \in R^n$.

Then repeat:

$x^{(k)}\ =\ x^{(k-1)}\ -\ t_k\ .\ \nabla f\left(x^{(k-1)}\right),\ k\ =\ 1,\ 2,\ 3,\ ...$

- move in the direction of the negative gradient, which points us 'downhill'

Stop at some point

- we will find precise ways to stop later, inspired by duality

A1. An interpretation of gradient descent

At each iteration, consider the expansion/quadratic approximation

$$f(y)\ \approx f(x)\ +\ \nabla f(x)^T(y-x)\ +\ \frac{1}{2t}||y-x||_2^2$$

This is a second-order Taylor series approximation, except instead of the Hessian $\nabla^2 f(x)$, we use $\frac{1}{t}I$ - this throws away the curvature information

BUt this approximation is a convex function!

- It is a convex quadratic
- and we know that $dom(f) = R^n$

So the first-order characterization for convex functions applies.

And to minimize this approximation, i.e. find an approximation to the minima, we can find the gradient and set to zero.

Here we use $y = x^+$

$$\nabla f_{approx}(y) = \nabla f(x) + \frac{1}{t}(y - x) = 0 \; at \; minima$$

$$\implies y = x - t\nabla f(x)$$

So at every step, we pick y that minimizes this quadratic approximation for f(y)!

Another way to think about this approximation is as a linear approximation + a proximity term

- $\nabla f(x)^T(y - x)$ is the linear approximation: but unbounded, this is useless as you can pick y = - inf
- $\frac{1}{2t}||y - x||_2^2$ says 'stay close to x', weighted by the parameter t
- if t is high you get further away from x, because the penalty (in terms of the added last term) is lower
- if t is lower, you stay closer to x
- this aids in interpreting the update rule form as well!

Outline

How to pick step sizes
Convergence analysis
Non-convex functions
Gradient boosting

A2. Picking step sizes

1. Fixed step size

- the trouble is if t is too large, the algorithm can diverge far away from the minimum!
- too small, then algorithm convergence is too slow
- what is 'just right' here?
  - convergence analysis will help here!

2. Adaptive step size: Guess the best step size at every step using some heuristic

Arguably the most popular heuristic is **backtracking line search**.

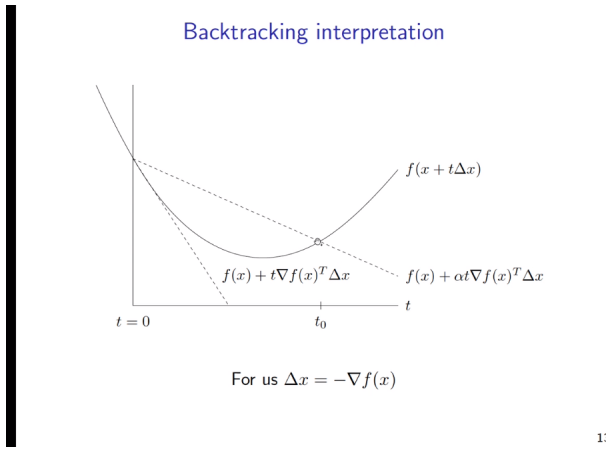
- First fix parameters $0 < \beta < 1, \; 0 < \alpha \leqslant 1/2$

- At each iteration
  - Start with $t = t_{init}$
  - While $f(x - t\nabla f(x)) > f(x) - \alpha t||\nabla f(x)||_2^2$
    * i.e. perform two approximations to the function
      · evaluate the function at the current gradient descent update x, and
      · evaluate a relaxed linear approximation to the function
      · if the function approx is worse than the linear approx, then you are probably diverging away from the minimum - so you need to reduce the step size!
    * shrink $t = \beta t$
  - Else perform gradient descent update $x^+ = x - t\nabla f(x)$

Simple and works well in practice! People often just set $\alpha = 1/2$

Q: What is the motivation for this?

- Think about a linear approximation to the function $f(y)$
- $f(y) \approx f(x) + \nabla f(x)^T(y-x)$, $where\ y = x + t\Delta x$
  $\approx f(x) + t\nabla f(x)^T\Delta x$
- By convexity, we know that the tangent line lies below the function
  - i.e. $\nabla f(x)^T(y-x)$ lies below $f(y-x)$
- So this linear approximation for $f(y)$ will always undershoot the true $f(y)$ - you can never do as well as this (at minimizing the function)!
- So we relax the slope condition - weight the slop by α - and try do as well as this relaxed linear approximation!
  - 'make at least a fraction α of the progress that the tangent line would make'
- Bear in mind, for us we KNOW what Δx is - it is $-\nabla f(x)$
  - So the only unknown/variable quantity in this approxmation is t - which defines the line for $f(y)$!



3. Exact line search

Motivation: why use a heuristic as in 2? You can just find the exact best t for every iteration

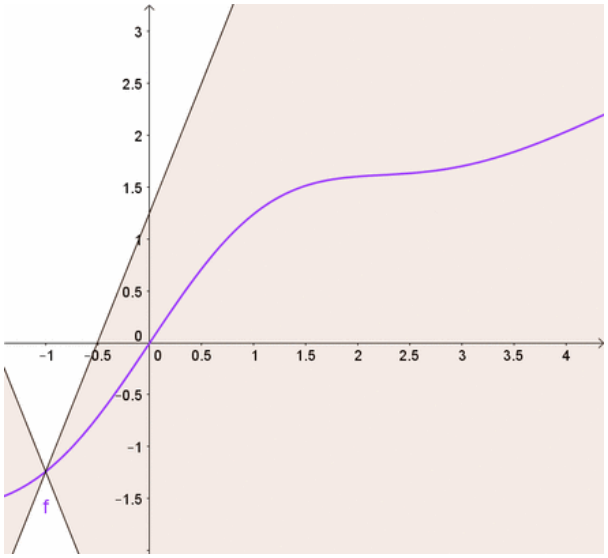
$t = argmin_{s \geqslant 0} f(x - s\nabla f(x))$

- look at every value for s
- x is fixed, so $\nabla f(x)$ is fixed
- we know f is convex, and we are looking at an affine transfomation of s, so this is a convex function of s - we shuold be able to calculate the absolute minima!

But this is approch is usually not very efficient, and sometimes it is not possible to do this opt exactly.
In practice, the heuristic method is often better, except perhaps for small problems

A3. Convergence analysis

Assume that f is convex, differentiable, dom(f) = R^n and additionally that ∇f is Lipschitz continous with constant L > 0



This means that $||\nabla f(x) - \nabla f(y)||_2 \leqslant L||x - y||_2$ for any x, y
This also means that the largest eigenvalue of the Hessian of f is L for all x

- One interpretation is that this is some bound on the curvature of the function

Theorem: Gradient descent with fixed step size $t \leqslant 1/L$ satisfies
$f\left(x^{(k)}\right) - f^* \leqslant ||x^{(0)} - x^*||_2^2 \ / \ 2tk$
If t = 1/L, then you can imagine this saying
$f\left(x^{(k)}\right) - f^* \leqslant L||x^{(0)} - x^*||_2^2 \ / \ 2k$

Proof?

- uses the property that having a gradient that is lipschitz gradient means you can upper

bound the function by some quadratic

Interpretation

"After k iterations, the gap between where you started and the minimum goes down by a factor of 1/k"

Note that the numerator is constant - it only uses the init guess for x.

"How much better than your original guess are you doing after k steps?"

So we say gradient descent has convergence rate O(1/k)

Epsilon convergence is another way of expressing the same property

- The setup here is: "how many steps k will it take to get to a solution $f\left(x^{(k)}\right) - f^* \leqslant \epsilon$?"
- By the theorem this is saying $||x^{(0)} - x^*||_2^2 \; / \; 2tk$ = c/k = upper bound on the criterion diff $\leqslant \epsilon$
- so k = c/ϵ

So this convergence is O(c/ϵ) $= O(1/\epsilon)$

If the convergence rate were O(1/k^2), then c/k^2 = eps → sqrt(c/e) = k → so it would be $O\left(1/\sqrt{\epsilon}\right)$

A3.1 Convergence under strong convexity

A strong convex function can be lower bounded by a quadratic - it must curve more than M

While Lipschitz says the function can be upper bounded by a quadratic - it must curve less than L

Convergence under strong convexity

Reminder: strong convexity of f means $f(x) - \frac{m}{2}\|x\|_2^2$ is convex for some $m > 0$

Assuming Lipschitz gradient as before, and also strong convexity:

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

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

where $0 < c < 1$

Rate under strong convexity is $O(c^k)$, exponentially fast! I.e., we find ϵ-suboptimal point in $O(\log(1/\epsilon))$ iterations

Note that this a (literally) exponential improvement on the rate of convergence!

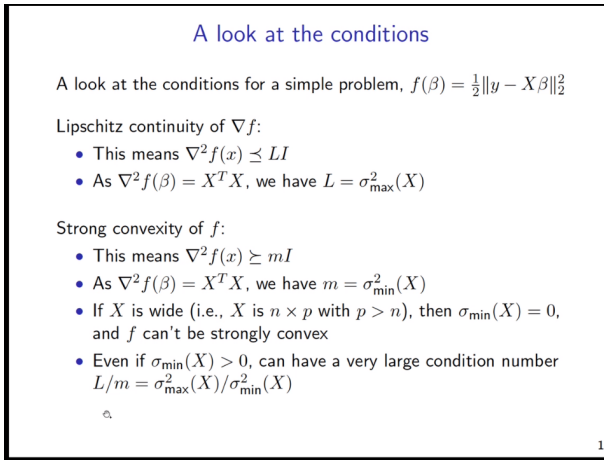
If you wanted high precision convergence, e.g. if $\epsilon = 2^{-6}$

Then $O(1/\epsilon)$ implies 2^6 steps
But $O(\log(1/\epsilon))$ implies 6 steps!

- This is called linear convergence?
  - because if you plot y axis on a log scale, it goes down linearly...
  - y axis = distance from function minimum
  - x axis = k i.e. number of iterations

The contraction factor c in rate depends adversely on the condition number L/m

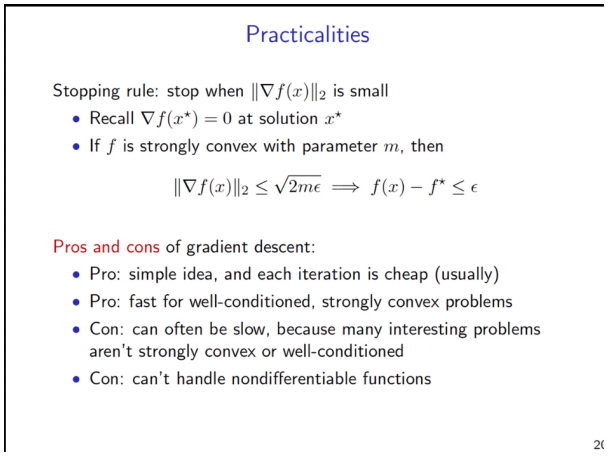
- the largest vs the smalles eigenvalue of the hessian
- if there are functions that are conditioned poorly - curved more in some directions than others - then grad desc does quite poorly
  - This is not just a theoretical upper bound - it shows up in practice as well!



- The last bullet suggests that if the predictors in X are correlated, then the smallest singular value will be small - the ratio will be large - so the performance of gradient descent will be bad
  - https://stats.stackexchange.com/questions/70899/what-correlation-makes-a-matrix-singular-and-what-are-implications-of-singularit
  - https://math.stackexchange.com/questions/759208/multicollinearity-and-svd
- Remember
  - psd = smallest eigenvalue $\geqslant 0$

A4. Practicalities

Stopping rule: stop when $||\nabla f(x)||_2$ is small



Can we do better?

For first-order methods, i.e. methods that rely only on the gradient, can we achieve a better convergence rate than gradient descent (for diff functions with Lipschitz gradients)?

First-order methods: iterative method which updates $x^{(k)}$ in

$$x^{(0)} + span\left\{\nabla f\left(x^{(0)}\right), \nabla f\left(x^{(1)}\right), ..., \nabla f\left(x^{(0k)}\right)\right\}$$

In the 1980s, Nesterov proved a lower bound on the convergence rate for functoins with these assumptions using first-order methods.

$$f\left(x^{(k)}\right) - f^* \geqslant 3L||x^{(0)} - x^*||_2^2 / 32(k+1)^2$$

so the lower bound is 1/k^2, while the gradient descent rate is in 1/k

So this means vanilla gradient descent is not optimal!

- can use accelerated gradient descent