Overview of optimization

Yifan Sun

Preliminary

Purpose

A friendly introduction to optimization.

Disclaimer

I'm basically pulling all of this out of [1], which is a (freely available) canonical book on optimziation (authors are profs at Stanford and UCLA). Specifically, I'm pulling memorable examples from Chapters 1,4, and 9. Some of the linear algebra stuff I also pulled from [2], which is also a book written by a UCLA prof. When I say "pulled", I mean word for word, so please don't distribute this because I'm gonna be in such copyright hell if that happens.

This is good stuff, actually - could easily be turned into a teaching portfolio since I know you were worried about that.

Big ideas

There are two big ideas I really want to get across in this document.

One is the idea that the problem is as important, if not more so, than the solution. A big part of optimization theory is recognizing that most problems are essentially the same, and if you can frame your problem in a standardized way, you can easily recognize the class of solutions one can use to solve it.

← THANK you, says the global learner

The other is that matrices and vectors can be treated like scalars, but in completely nonintuitive ways. Throughout the document I will try to draw connections between the two regimes. One should keep in mind that these connections are purely "qualitiative, not quantitative"; they are not rigorous definitions, just ways of thinking.

ok? I'll keep an eye out.

Software

When doing optimization problems, I generally use cvx (http://cvxr.com/cvx/), which is a plugin for MATLAB. That choice is probably heavily influenced by the fact that Stephen Boyd sort of pioneered that, but actually it's pretty good software too.

Notation

One of the symptoms of rabbit-holing is I develop a lot of idiosyncratic notation that I can't live without, but sometimes is really confusing for others. This part isn't really important, but will hopefully eliminate confusion wherever possible.

- $\mathbb{R}$ refers to the space of real numbers.
- $\mathbb{R}^n$ refers to the space of vectors of length n, whose elements are real.
- $\mathbb{R}^{m\times n}$ refers to the space of matrices with m rows and n columns, whose elements are real.

- In general, if x and y are vectors, the notation $x(=,\leq,<)y$ means $x_i(=,\leq,<)y_i$ for each element x_i, y_i in x, y.
- In general, if A and B are matrices, the notation $A = B$ means each element in A is equal to that element in B, and vice versa.
- The notation $A(>,\geq)B$ does not mean element-by-element inequality. Rather, this means that $(A-B)(>,\geq)0$, where $X(>,\geq)0$ means the eigenvalues of X are strictly positive or nonnegative, respectively. In many ways, this is the better analogy between matrices and scalars than element-by-element comparison. For example, X has a "square root" $X = R^T R$ if and only if $X > 0$ and is symmetric. X has an inverse if and only if $X \neq 0$, that is, X has no zero eigenvalues. Otherwise, the result is very much like dividing by 0.
- $f : A \to B$ is a function that takes an input from the space A and outputs to the space B.
- $\forall, \exists$ means for all, and there exists, respectively.
- $|x|$ for a scalar x means absolute value. $||x||$ for a vector x means norm. Specifically, the 2-norm is

$$||x||_2 := \sqrt{\sum_i x_i^2}.$$

and the 1-norm is

$$||x||_1 := \sum_i |x_i|.$$

Problem statement

Every (numerically computable) optimization problem can be stated in the following form:

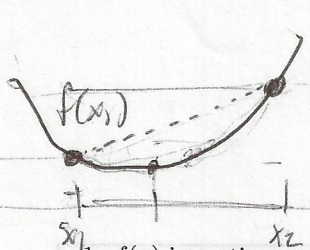
$$\begin{aligned} \min_x \quad & f(x) \\ \text{subject to} \quad & g(x) \leq 0 \\ & h(x) = 0 \end{aligned} \tag{1}$$

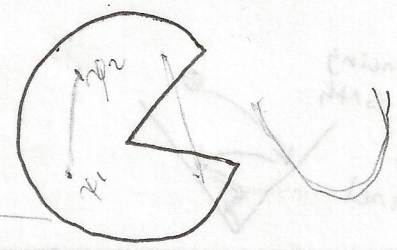
where $f(x)$ is referred to as the objective function (or a cost function), and $g(x)$ and $h(x)$ are the constraints. x can in general be a scalar, vector, or matrix. Note that to change this to a maximization problem, we simply change the sign on $f(x)$. The set of points $\mathcal{X} = \{x : g(x) \leq 0, h(x) = 0\}$ is referred to as the feasible set. If $\mathcal{X}$ is empty, then the problem is not feasible.

> **Example:** Consider you are manufacturing electronic parts. You would like to maximize your yield. So, you frame your yield based on parameters (number of machines, quality of machines, available quantity, etc.) This becomes the objective function. But most companies cannot operate unconstrained; there are cost constraints, and quality specifications. Some of these constraints might be ineqaulities (cost < budget), others might be equality (clock rate = 1GHz).

Although these few lines () look very simple and sparse, they are in fact very general. Basically anything you would optimize with a computer (sorry romantic relationships don't apply) you can frame in this form. In fact, this class of problems is actually far too difficult to solve. Without knowing more, the only solution that will work for every possible such problem would be a brute force search of the constraint space. That sucks.

We need to narrow our space a little. Rather than consider every possible optimization problem, let's start with the problems we know how to solve, and work our way up. For now, we will not consider constraints.





I understand what convex means, but I'm not sure how this notation transfers to it.

1. $f(x)$ is continuous, differentiable, and convex

   To be precise, we say a set $\mathcal{X}$ is convex if $\forall x_1, x_2 \in \mathcal{X}$, the convex combination $\theta x_1 + (1-\theta)x_2 \in \mathcal{X}$, where $0 < \theta < 1$. A line, the inside of a circle, and the inside of any regular polygon, for example, are convex. The inside of a Pacman shape is not convex, since the mouth part contains points that are not in the set, but can be expressed as a convex combination of the lower lip and forehead.

   this made me laugh on the plane

   We say a function $f(x)$ is convex if $\forall x_1, x_2,\ f(\theta x_1 + (1-\theta)x_2) \leq \theta f(x_1) + \theta f(x_2)$. In other words, convex functions are like bowls; they always dip below, never above. Convex functions are nice because if you find a local minimum, you know that's the global minimum. [1]

   $\leq \theta f(x_1) + (1-\theta) f(x_2)$

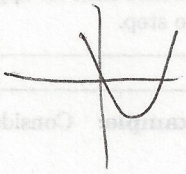
   A TYPO

   Jensen's inequality

   Example: As a refresher, let's take a look back at some calculus. Consider the function $f(x) : \mathbb{R} \to \mathbb{R}$

   $$f(x) = (2x-5)^2 + 8. \quad = 4x^2 - 20x + 33$$

   We wish to find x such that $f(x)$ is minimized. The function is everywhere continuous and differentiable, and the first and second derivatives are:

   $$\begin{aligned} f'(x) &= 8x - 20 \\ f''(x) &= 8 \end{aligned}$$

   By finding x where $f'(x) = 0$, we can determine all the stable points of $f(x)$, i.e. where $f(x)$ could be a minimum, maximum, or saddle point. These points are the local extrema of $f(x)$. Some algebra tells us the only such point is $x = 5/2$.

   ok, so calc is 1D optimization.

   Additionally, since $f''(x) = 8 > 0, \forall x$, then the function is convex everywhere. ($f(x)$ is like a bowl.) This tells us that $f(x)$ only has one minimum, and no maximum or saddle points. So we know that $f(x)$ is minimized if $x = 5/2$.

   I remember that!

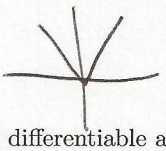
   What if, rather than having x as a scalar, x was instead a vector, or a matrix? Then we would expect similar things can be done with the "derivatives" of $f(x)$, using the gradient ($\nabla f(x)$) and hessian ($\nabla^2 f(x)$). More on this later. ok, I trust you.

   the hell? I don't remember that.

   In general, ~~such problems can be solved analytically~~, but rarely do they occur in practice. Still, this example gives us an idea of what to expect with convex functions, which is that by searching for local minima, we are assured to find global minima.

2. $f(x)$ is continuous and convex, but not differentiable.

   Example: Consider $f(x) : \mathbb{R} \to \mathbb{R}$

   (visual learner me!)

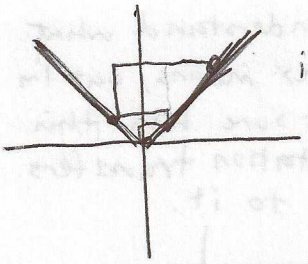
   $$f(x) = |x|$$

   The minimum of this function is $x = 0$, but since the function is not differentiable at $x = 0$, there's really no nice analytic way of arriving at that. But intuitively, this shouldn't be so hard to find. Consider the following algorithm:

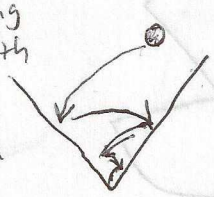
   Lipschitz continuity

[1] This concept is the foundation behind convex optimization. If you did not know a function was convex, you must assume that at best your algorithm will require $O(\epsilon^{-n})$ complexity, where ϵ is your tolerance and n is the number of points in your sample space. If, however, your problem is convex, your complexity might be polynomial, linear, or constant.

I'd like to derive this more in turn to understand it but will pass due to time now.



it's like a ball bouncing back + forth in a funnel



- Pick any starting point x_0, and find $f'(x_0)$. In this example, this derivative is just

$$f'(x_0) = \begin{cases} -1 & \text{if } x_0 < 0 \\ 1 & \text{if } x_0 > 0 \end{cases}$$

  If we are unlucky and pick $x_0 = 0$, then return not-a-number and pick another starting point. any way to flag "hey, this might be the answer"?
- Pick a starting step size, Δx_0.
- For $i = 0, \ldots,$ if $f'(x_i) > 0$, then $x_{i+1} = x_i - \Delta x_i$, and if $f'(x_i) < 0$, then $x_{i+1} = x_i + \Delta x_i$. Additionally, at each step, $\Delta x_{i+1} = \frac{1}{2}\Delta x_i$.
- Continue until Δx_i is within your tolerance.

This method is called bisection, and if you can manage to overstep in the first step, then this method will have linear convergence. (It's a geometric sequence!) Note, however, that this is nowhere near as appealing as in the previous example, where we found the exact minimum in one step.

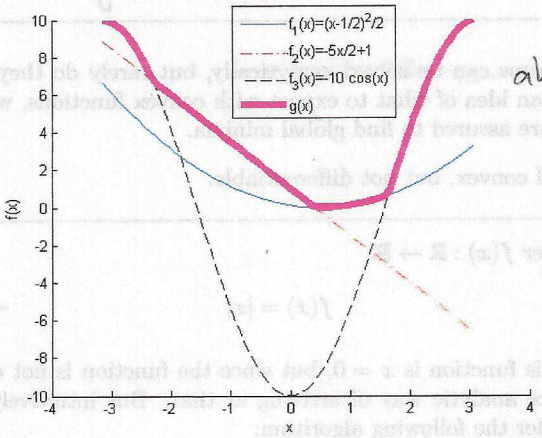
if you pick too small a Δ for your x_0 (ie "you don't overstep") you never get there- Xeno's paradox sort of limit convergence. So you really want to overstep.

Example: Consider the problem of minimizing the maximum of a set of convex functions:

$$g(x) = \max_i f_i(x), i = 1, \ldots, n$$

Each function is continuous and differentiable, but where they intersect, it might be pointy (i.e. not differentible). For example, Figure 1. You can prove that such a combination of convex functions is guaranteed to be convex (by using the definition of a convex function). This is actually more of what realistic problems might look like, and bisection can also be used here to find the minimum.

ooh, yes, you can. happy tiny math moment,



ah, I see the application, yep.

Figure 1: Example of three functions that are convex over the domain $-\pi < x < \pi$.

3. $f(x)$ is convex, but not continuous nor differentiable.

   There's a lot of literature on how to solve these types of problems (search subgradient methods). I'm still learning about this, so I won't go here. Suffice to say it's a solved problem, so if you ever run into a problem that falls in this category, rejoice, for ye shall have solutions.

4. $f(x)$ is continuous, but is not differentiable nor convex.

   As mentioned before, if you run into this catagory, you are in general out of luck; in general, you're stuck sampling and checking to see if you hit the minimum. If you know a lot about the problem structure, you might be clever about how you sample, but that's a whole other topic.

Neato! Hopefully by now you've gotten to a point where you're thinking, yeah I see how my current research problem might fit into one of these categories, but man, this is such an oversimplification! Real problems are rarely scalar functions, with possibly billions of variables and very unintuitive structure. Not to mention, what happens if you factor noise into the picture? If you really want to know the answer to these questions, you should read through the references and also play around with some optimization software (I use cvxopt because my adviser wrote that, but there are plenty good ones available on the interwebs). In these next sections I will just go through some very standard problems and known solutions, and a few examples that tickle me greatly.

Some types of convex problems

Least squares problem

Everybody remembers the linear regression problem from high school science class, where you're given a set of data, and told to fit a line to it using excel, or a calculator. If you are fancy, you might even have tried a polynomial fit. Well, what if I told you that these kind of fitting problems fall into a general (and easily solvable) class of problems called least squares, and the solution is really easy to calculate?

Example: I have a set of data, $y_1, \ldots, y_n$ taken at times $t_1, \ldots, t_n$, and I wish to find some linear correlation, that is, α, β such that $y_i \approx \alpha t_i + \beta$. To make notation easier, I will define the vector $y, t \in \mathbb{R}^n$ containing these datapoints as elements. I'll define $x = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ and a matrix $A = [t, \underline{1}] \in \mathbb{R}^{n \times 2}$, where $\underline{1}$ is a vector with all elements 1. Then what we want is

$$y \approx Ax$$

Another way to think about it is to define a residual, $r_i = y_i - \alpha t_i - \beta$ and the vector r containing these elements, and somehow minimize the "size" of r_i. For example, I might do

$$\min_x \sum_i |r_i|$$

but this in general is very hard. However, it turns out that the solution to

$$\min_x \sum_i |r_i|^2 = ||r||_2^2$$

can be derived analytically. (You can intuitively think of this as remembering that a differentiable function has an analytic minimum, but if it isn't differentiable then you have to use a method like bisection to find it, which requires more steps.) This is why this type of problem is called least squares, because you are minimizing the squared error rather than the actual error. There are advantages and disadvantages to this, which I will touch later when discussing regularization.

The least squares problem endeavors to solve

$$\min_x ||r||_2^2 = ||y - Ax||_2^2 = (y - Ax)^T(y - Ax)$$

where A has more rows than columns. (In the above example, A has n rows and 2 columns.) [2]

The solution to the least squares problem can be found using some fancy linear algebra to be the set of x such that $A^TAx = A^Tb$. (In general, unless A has full column rank, many such solutions exist.) You can pretty much find a proof of this in any grad level linear algebra textbook, but it might be interesting to derive this result in a different way. Remember our toy example with the convex, continuous, differentiable function, in which we found the minimum by just setting the derivative to 0 and assuring the function was convex? Let's try that same trick here. First, we find the gradient (first derivative) and hessian (second derivative):

$$\begin{aligned} \nabla f(x) &= \nabla((y - Ax)^T(y - Ax)) = \nabla(y^Ty - 2y^TAx + x^TA^TAx) = -2A^Ty + 2A^TAx \\ \nabla^2 f(x) &= \nabla(-2A^Ty + 2A^TAx) = 2A^TA \end{aligned}$$

Note that in general, if $f : \mathbb{R}^n \to \mathbb{R}$, then the gradient is a vector and the hessian is a matrix.

It turns out that any matrix written in the form A^TA is positive semidefinite (has no negative eigenvalues), and you can prove that if $\nabla^2 f(x) \geq 0$ then $f(x)$ is convex. So we can do the same trick, and find x the minimum of $f(x)$ if $-2Ay + 2A^TAx = 0 \Rightarrow A^TAx = A^Ty$. That's pretty cool!

Linear programs

This is a funny word that really confused me when I first got to UCLA, but apparently it's a pretty common term to describe a very common (but simple) class of problems. Basically, linear programs can be described as

$$\begin{aligned} \min_x \quad & c^Tx \\ st \quad & Ax = b \\ & Gx \leq h \end{aligned}$$

for vectors c, x, h and matrices A, G.

Example: Consider the optimal diet plan. Your weight depends on some linear combination of caloric intake, exercise, stress levels (for example). The importance of each factor depends on parameters $c = [c_1, \ldots, c_n]^T$ and how much you do them is in the variables $x = [x_1, \ldots, x_n]^T$. So then you want to minimize your weight:

$$\min_x c^Tx$$

This problem is clearly unconstrained. The solution will be $x_i = \infty$ if $c_i < 0$ (the factor decreases your weight) and $x_i = -\infty$ if $c_i > 0$ (if that factor increases your weight). Obviously you can't exercise infinitly often and you can't never eat, so this model is really silly. You need to add some constraints, like having a healthy level of food, and not overworking. You can frame those as linear constraints, and voila, you have a linear program to solve.

[2] If A had more columns and rows, this would become a sensing problem, rather than a data fitting problem–google compressive sensing for more info.

Example: This is a problem I literally ripped from the Convex Optimization textbook that I really liked, mostly because it used funny shapes. First, a few terms:

An n-ball is a set of points $B(x_c, r) = \{x_c + u | ||u||_2 \leq r\}$, where x_c and r are the center and radii of the ball, respectively. Here, x_c and u are vectors, and r is a scalar. For $n = 1, 2, 3$, this is a line segment, filled circle, and filled sphere, respectively. If $n > 3$, we might call it a hypersphere.

Similarily, a polyhedron in n dimensional space with m sides can be described as $P(A, b) = \{x \in \mathbb{R}^n | a_i^T x \leq b_i, i = 1, \ldots, m\}$. Here, x is a vector with n elements, b is a vector with m elements $(b_1, \ldots, b_m)$ and A is a matrix with n columns and m rows, where a_i is the vector describing the ith column of A. You can understand why this is true by first observing that the set $\{x \in \mathbb{R}^n | a_i^T x \leq b_i\}$ for a specific i is a half-space, (it's a partition of the n dimensional space). A polygon is simply an intersection of halfspaces. Figure 2 tries to illustrate this for $n = 2$.

The question is to find the largest ball to fit inside an arbitrary polygon. This, it turns out, is a linear program:

$$\begin{aligned} \max_{r, x_c} \quad & r \\ \text{subject to} \quad & a_i^T x_c + r(a_i^T a_i) \leq b_i, i = 1, \ldots, m \\ & r \geq 0 \end{aligned}$$

[3]

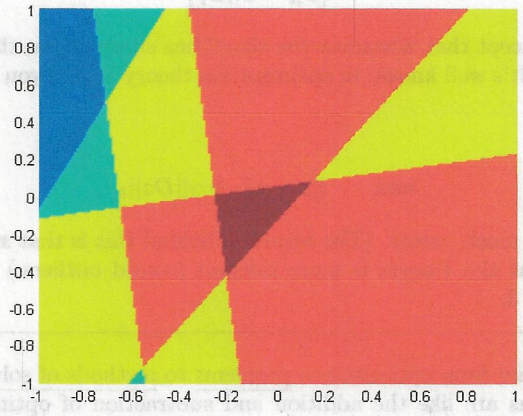


Figure 2: Example of partitions in 2-dimenstional space, creating halfspaces. If you take the intersection of a set of halfspaces, you will get a polygon.

Before getting to how to solve linear programs, I'll first do a detour into regularization, which is kind of a funky topic.

[3] One application for this type of problem is in communications. Imagine there are many cell phone companies servicing a region, and each company uses its own tower. The region is cut up into polygons, and each company has to decide where to put the cell tower such that there is maximum coverage, but the signal does not enter any other region (a hard boundary).

Regularization

Example: In this example, we'll attempt to fit a pure square wave to a noisy signal. (Scroll down to Figure 3 for a peak.) The idea here is simple: I have a signal, x_i, and I want to find z_i such that $x_i \approx z_i$ but the interpolation of the points in z_i is smooth. One way we might do this is to minimize both criteria jointly. So, like we did in the least squares example, we're going to fit a line by reducing the squared error, i.e.:

$$\min \quad ||z - x||_2 + \gamma ||Dz||_2$$

where

$$D = \begin{bmatrix} -1 & 1 & 0 & \dots & 0 & 0 & 0 \\ 0 & -1 & 1 & \dots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & -1 & 1 & 0 \\ 0 & 0 & 0 & \dots & 0 & -1 & 1 \end{bmatrix}$$

and therefore

$$Dx = \begin{bmatrix} x_2 - x_1 \\ x_3 - x_2 \\ \vdots \\ x_N - x_{N-1} \end{bmatrix}$$

This kind of works, except that whenever the signal has sharp edges, the 2-norm approximation gets lazy, and lops it off. It's well known in optimization theory that if you replace this with 1-norm regularization, that is,

$$\min \quad ||z - x||_2 + \gamma ||Dz||_1$$

then the edges will fit much better. (The intuition behind this is that x^2 grows a lot faster than $|x|$, so one might say that the 1-norm is more tolerant toward outliers.) You can see how much better the fit is in Figure 3.

At this point I will shift gears from optimization problems to methods of solving these problems. Least squares and linear programs are like the addition and subtraction of optimization. There are many different types of problems (data fitting, scheduling, cost-benefit analysis) that fall into these categories. But these problems really only scratch the surface of general convex problems, which in itself is a very special type of problem. If you want to learn more about them, I recommend watching some lectures by Stephen Boyd (available on youtube), or checking out the textbook in the references.

Methods for solving convex problems

Gradient Descent

The idea behind gradient descent is simple. If you're standing on a mountainside and you want to get to the valley, and you know the area you're standing in is bowl-shaped, then you keep stepping in the most downward direction possible until you get to where it feels flat. Now imagine your cost function

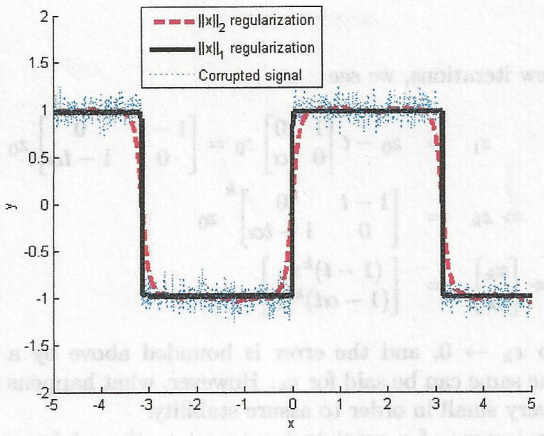


Figure 3: Result of regularization for data fitting. For 2-norm, I used $\gamma = 10$. For 1-norm, I used $\gamma = 1$.

is the height of the mountain ($f(x)$), and the minimization problem is to find the valley $x : \nabla f(x) = 0$. Then the gradient descent algorithm is:

1. Pick a starting point within the feasible set x_0.
2. For $i = 0, \ldots,$ evaluate the cost $f(x_i)$ and the gradient $\nabla f(x_i)$.
3. The gradient descent step direction is

$$\Delta x = -\Delta f(x)$$

and the update step is therefore

$$x_{n+1} = x_n + t\Delta x$$

where t is the step size. [4]

In general, gradient descent is considered the most straightforward method, but its convergence is usually deemed too slow. The following example will try to illustrate this:

Example: Consider the optimization problem:

$$\min_{x,y} \quad f(x,y) = \frac{1}{2}(x^2 + \alpha y^2)$$

We can analytically see that the minimum is $(x, y) = (0, 0)$. But, to get an idea of how gradient descent converges, let's try and see how fast we arrive at this answer numerically.

We have

$$\nabla f(x,y) = \begin{bmatrix} x \\ \alpha y \end{bmatrix} \Rightarrow \nabla f(z) = \begin{bmatrix} 1 & 0 \\ 0 & \alpha \end{bmatrix} z$$

[4] Determining the optimal step size is an art within itself. If the step size is too big, the method will become unstable (i.e. a ball bouncing inside a bowl so hard it bounces out). If the step size is too small, convergence is too slow. For now, we assume that the step size is just right; for details, see references.

for $z = \begin{bmatrix} x \\ y \end{bmatrix}$

If we run through a few iterations, we see

$$\begin{aligned} z_1 &= z_0 - t\begin{bmatrix} 1 & 0 \\ 0 & \alpha \end{bmatrix} z_0 = \begin{bmatrix} 1-t & 0 \\ 0 & 1-t\alpha \end{bmatrix} z_0 \\ \Rightarrow z_k &= \begin{bmatrix} 1-t & 0 \\ 0 & 1-t\alpha \end{bmatrix}^k z_0 \\ \Rightarrow \begin{bmatrix} x_k \\ y_k \end{bmatrix} &= \begin{bmatrix} (1-t)^k x_0 \\ (1-\alpha t)^k y_0 \end{bmatrix} \end{aligned}$$

In general, $t < 1$, so $x_k \to 0$, and the error is bounded above by a geometrically decaying sequence. If $t < (2/\alpha)$, the same can be said for y_k. However, what happens if α is very large? Then we are forced to make t very small in order to assure stability.

Figure 4 shows the trajectory of a point trying to get to the minimum. The black like is the gradient descent method, and as you can see, the trajectory is very indirect; it heavily favors steps in the y direction, and requires a small t not to overstep.

Given that we don't know initially what kind of problem we'll have (i.e. what α is) ahead of time, there's really nothing better we can do than take the convergence hit. But what if we did know? What if we could sense the system's structure before going into the optimization part, and could estimate α? Then we might try to fix our step direction so that, rather than taking steps heavily skewed toward the y direction, we actually went toward the center. For this type of problem, it is known that the correction factor should be

$$\Delta x = -P^{-1}\nabla f(x)$$

where P describes the ellipsoid. In this case,

$$P = \begin{bmatrix} 1 & 0 \\ 0 & \alpha \end{bmatrix} \Rightarrow P^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & \alpha^{-1} \end{bmatrix}$$

The red line shows the trajectory of the corrected step (also known as the steepest descent step). Note that there are no dots; it converges in one step, with $t = 1$.

Newton's method

As we see how the gradient descent direction can be so easily corrected, we are motivated to develop this next method, is a much more general approach toward quadratic approximation. The intuition here is that gradient descent is like a 1st order Taylor approximation to where the flat part of the valley is, and Newton's method is a 2nd order approximation. The derivation is as follows:

$$\nabla f(x + \Delta x) = \nabla f(x) + \nabla^2 f(x)\Delta x + \mathcal{O}(\Delta x^2)$$

which is just Taylor's theorem applied to a function $g(x) = \nabla f(x)$. As we noted before, the minimum of a function occurs when the derivative is zero, so if we take a step Δx, we want our location $x + \Delta x$ to be such that $\nabla f(x + \Delta x) \approx 0$. So the Newton step is:

$$\Delta x = -(\nabla^2 f(x))^{-1}\nabla f(x)$$

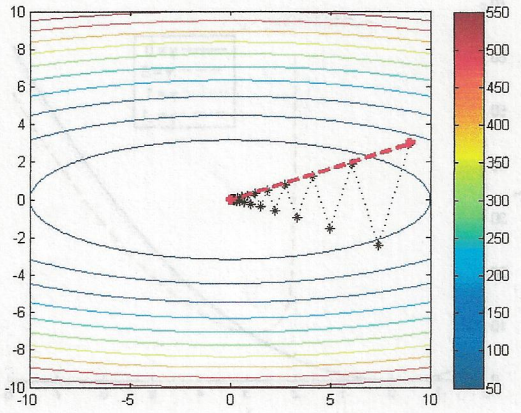


Figure 4: Gradient descent (black) and steepest descent (red) for $\alpha = 10$.

Note that, in the example for gradient descent, Newton's method also converges in one step. In fact, in general, if the problem is convex and follows certain parameters (nearly quadratic) it can be proven that if the method converges at all, it will converge in 50 steps or less. However, the tradeoff is that at each step, you need to evaluate the inverse of a Hessian, which is extremely nontrivial.

Logarithmic barrier

As one lasting thought, I'll leave you with one method on how to deal with constraints. Up until now, I've mostly focused on solving the unconstrained problem ,e.g. in (), there is no $g(x)$ or $h(x)$. I will just briefly mention one way in which inequality constraints $g(x) < 0$ can be incorporated into the objective function, using the logarithmic barrier function.

Consider the problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & g(x) < 0. \end{aligned}$$

If we remember some high school math, we recall that $\log(x)$ is only defined for positive values of x, and it gets really close to $-\infty$ as $x \to 0$. We are going to exploit this by basically building a "wall" on our objective function, making the objective function undefined if $g(x) > 0$ and the cost very very large as $x \to g^{-1}(0)$. We do this as follows:

$$f(x) \to f(x) - \gamma \log(g(x))$$

Example: Figure 5 shows the log barrier function acting on the objective $f(x) = x^2$ and the constraint $x > 2$. Note that as x gets really close to 2, the cost shoots up. The parameter γ tunes how much we actually modify the objective function; if γ is big, then the steps taken are smoother, but less accurate. If γ is really small, then the barrier shoots up very suddenly, and there's a high chance that you actually step out of the feasible region, but the objective function is well-preserved. In general, you would have γ change dynamically, starting with something big, and as you get closer to your minimum point, tune it up for better accuracy.

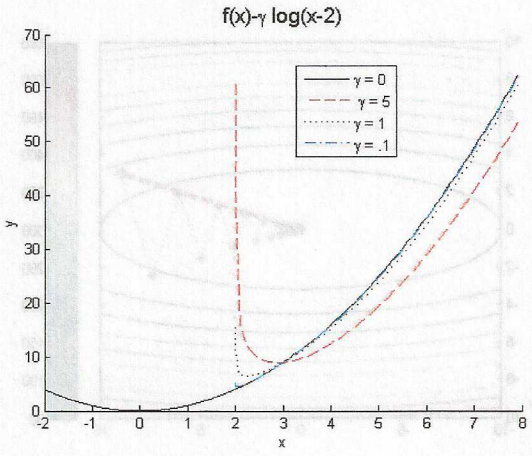


Figure 5: Log-barrier for the original objective function $f(x) = x^2$ and the constrant $x > 2$.

Conclusion

Well, that hopefully was a fun dive into some introductory optimization. I claim again that no original thought went into making this document, except that the topics I picked were the ones that stuck with me the deepest. This is in no means an exhaustive look; optimization is a very wide field, and even the stuff we do at UCLA is much wider than this, and goes beyond my wildest imaginations.

If you want a deeper look at this stuff, the course notes and homeworks are freely available here (http://www.ee.ucla.edu/ee236b/ee236b.html) as well as the book (in the references). Additionally, Stephen Boyd posts all his Stanford lectures on YouTube, in which he explains this stuff in more depth.

Hope that was enjoyable!

References

[1] S. Boyd, L. Vandenberghe Convex Optimization. Cambridge University Press, 2004. Available at `http://www.stanford.edu/~boyd/cvxbook/`

[2] A. Laub, Matrix Analysis for Scientists and Engineers. Society for Industrial and Applied Mathematics, 2005.