15-780: Optimization

J. Zico Kolter

March 14-16, 2015

Outline

Introduction to optimization

Types of optimization problems

Unconstrained optimization

Constrained optimization

Practical optimization

Outline

Introduction to optimization

Types of optimization problems

Unconstrained optimization

Constrained optimization

Practical optimization

Beyond linear programming

Linear programming

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & c^T x \\ \text{subject to} \quad & Gx \leq h \\ & Ax = b \end{aligned}$$

Beyond linear programming

Linear programming

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

General (continuous) optimization

$$\begin{array}{ll} \underset{x}{\text{minimize}} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{array}$$

where $x \in \mathbb{R}^n$ is optimization variable, $f : \mathbb{R}^n \to \mathbb{R}$ is the *objective function*, $g_i : \mathbb{R}^m \to \mathbb{R}$ are *inequality constraints*, and $h_i : \mathbb{R}^n \to \mathbb{R}$ are equality constraints

Example: image deblurring

Original image

Blurred image

Reconstruction

Figures from (Wang et. al, 2009)

Given corrupted $m \times n$ image represented as vector $y \in \mathbb{R}^{m \cdot n}$, find $x \in \mathbb{R}^{m \cdot n}$ by solving the optimization problem

$$\underset{x}{\text{minimize}} \quad \|K*x-y\|_2^2+\lambda \left(\sum_{i=1}^{n-1} |x_{mi} - x_{m(i+1)}| + \sum_{i=1}^{m-1} |x_{ni} - x_{n(i+1)}| \right)$$

where $K*$ denotes 2D convolution with some filter K

Example: machine learning

Virtually all machine learning algorithms can be expressed as minimizing a *loss function* over observed data

Given inputs $x^{(i)} \in \mathcal{X}$, desired outputs $y^{(i)} \in \mathcal{Y}$, hypothesis function $h_\theta : \mathcal{X} \to \mathcal{Y}$ defined by parameters $\theta \in \mathbb{R}^n$, and loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$

Machine learning algorithms solve optimization problem

$$\underset{\theta}{\text{minimize}} \quad \sum_{i=1}^{m} \ell\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$$

Example: robot trajectory planning

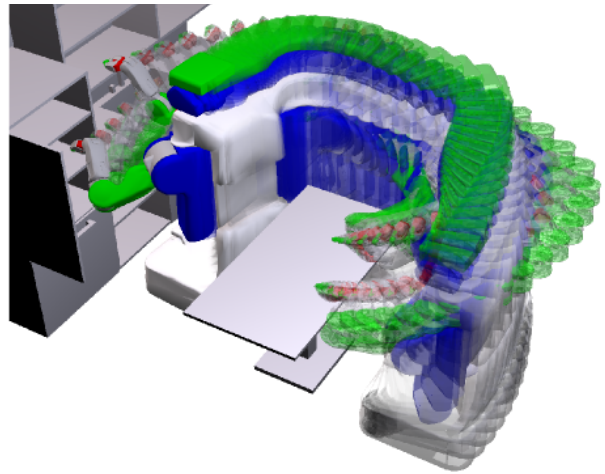


Figure from (Schulman et al., 2013)

Robot state x_t and control inputs u_t

$$\begin{aligned}
&\underset{x_{1:T}, u_{1:T-1}}{\text{minimize}} && \sum_{i=1}^{T-1} \|x_t - x_{t+1}\|_2^2 + \|u_t\|_2^2 \\
&\text{subject to} && x_{t+1} = f_{\text{dynamics}}(x_t, u_t), \quad \text{(robot dynamics)} \\
& && f_{\text{collision}}(x_t) \geq 0.1 \quad \text{(avoid collisions)} \\
& && x_1 = x_{\text{init}}, \; x_T = x_{\text{goal}}
\end{aligned}$$

Many other applications

We've already seen many applications (i.e., any linear programming setting is also an example of continuous optimization, but there are many other non-linear problems)

Applications in control, machine learning, finance, forecasting, signal processing, communications, structural design, any many others

The move to optimization-based formalisms has been one of the primary trends in AI in the past 15+ years

Outline

Introduction to optimization

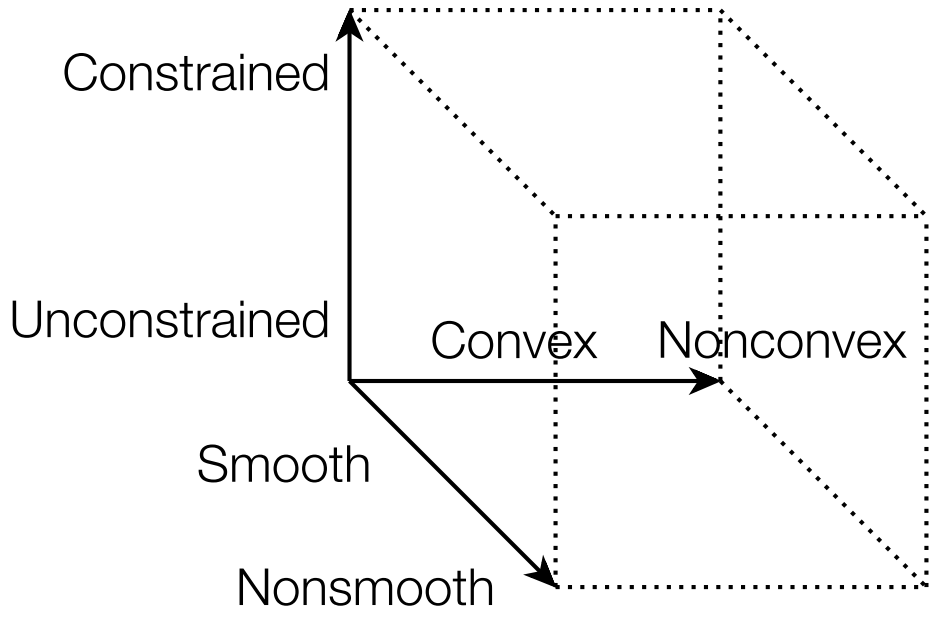
Types of optimization problems

Unconstrained optimization

Constrained optimization

Practical optimization

Classes of optimization problems



Many different classifications for (continuous) optimization problems (linear programming, nonlinear programming, quadratic programming, semidefinite programming, second order cone programming, geometric programming, etc) can get overwhelming

We focus on three distinctions: unconstrained vs. constrained, convex vs. nonconvex, and (less so) smooth vs. nonsmooth

Unconstrained vs. constrained optimization

$$\underset{x}{\text{minimize}} \quad f(x)$$

vs.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned}$$

In unconstrained optimization, every point $x \in \mathbb{R}^n$ is "feasible", so singular focus is on finding a low value of $f(x)$

In constrained optimization (where constraints truly need to hold exactly) it may be difficult to find an initial feasible point, and maintain feasibility during optimization

Typically leads to different classes of algorithms

Convex vs. nonconvex optimization

Originally researchers distinguished between linear (easy) and nonlinear (hard) optimization problems

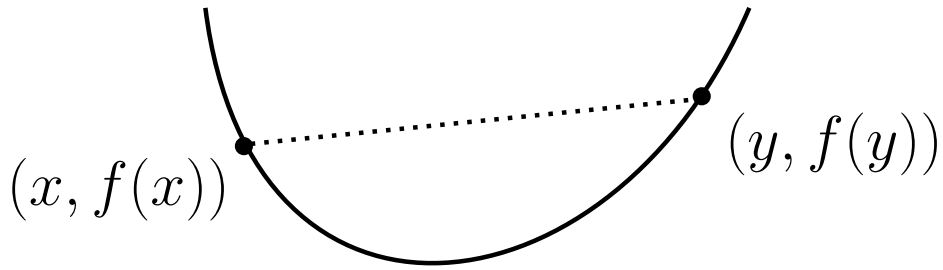
But in the 80s and 90s, it became clear that this wasn't the right line: the real distinction is between *convex* (easy) and *nonconvex* (hard) problems

The optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned}$$

if f and the g_i's are all *convex* functions and the h_i's are *affine* functions

Convex functions



A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if, for any $x, y \in \mathbb{R}^n$ and $\theta \in [0, 1]$,

$$f(\theta x + (1 - \theta)y) \leq \theta f(x) + (1 - \theta)f(y)$$

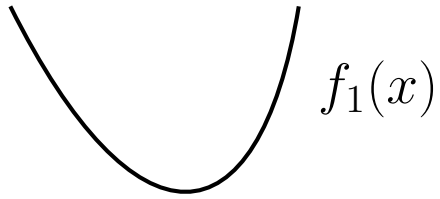
f is *concave* if $-f$ is convex

f is *affine* if it is both convex and concave, must take form

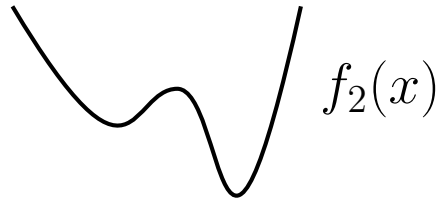
$$f(x) = a^T x + b$$

for $a \in \mathbb{R}^n$, $b \in \mathbb{R}$

Why is convex optimization easy?



Convex function



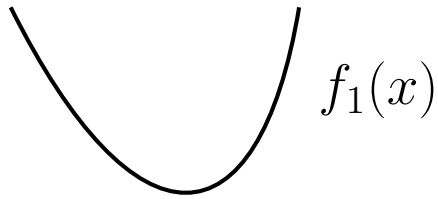
Nonconvex function

Convex function "curve upward everywhere", and convex constraints define a convex set (for any x, y that is feasible, so is $\theta x + (1-\theta)y$ for $\theta \in [0,1]$)

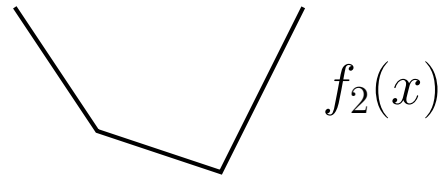
Together, these properties imply that any *local optima* must also be a *global optima*

Thus, for convex problems we can use local methods to find the globally optimal solution (cf. linear programming vs. integer programming)

Smooth vs. Nonsmooth optimization



Smooth function



Nonsmooth function

In optimization, we care about smoothness in terms of whether functions are (first or second order) *continuously differentiable*

A function f is first order continuously differentiable if it's derivative f' exists and is continuous; the Lipschitz constant of its derivative is a constant L such that for all x, y

$$|f'(x) - f'(y)| \leq L|x - y|$$

In the next section, we will use first and second derivative information to optimize functions, so whether or not these exist affect which methods we can apply.

Outline

Introduction to optimization

Types of optimization problems

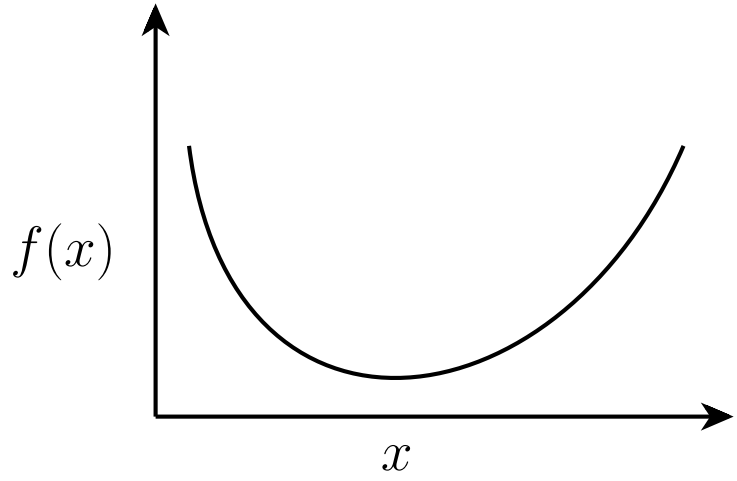
Unconstrained optimization

Constrained optimization

Practical optimization

Solving optimization problems

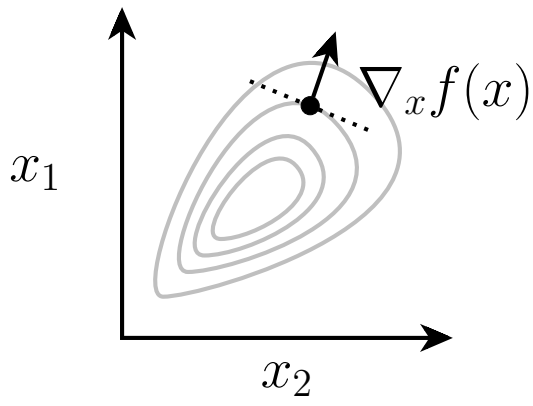
Starting with the unconstrained, smooth, one dimensional case



To find minimum point $x^\star$, we can look at the derivative of the function $f'(x)$: any location where $f'(x) = 0$ will be a "flat" point in the function

For convex problems, this is guaranteed to be a minimum (instead of a maximum)

The gradient

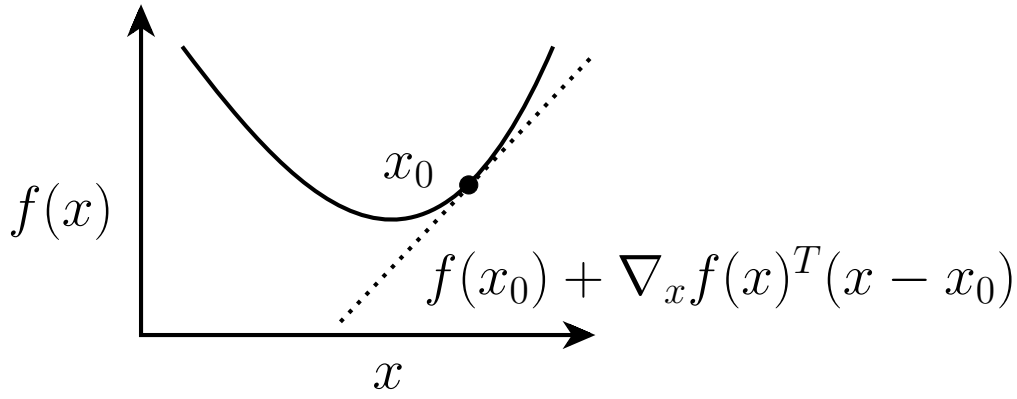


For a multivariate function $f : \mathbb{R}^n$, its *gradient* is a n-dimensional vector containing partial derivatives with respect to each dimension

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

For continuously differentiable f and unconstrained optimization, optimal point must have $\nabla_x f(x^\star) = 0$

Properties of the gradient



Gradient defines the first order *Taylor approximation* to the function f around a point x_0

$$f(x) \approx f(x_0) + \nabla_x f(x_0)^T (x - x_0)$$

For convex f, first order Taylor approximation is always an *underestimate*

$$f(x) \geq f(x_0) + \nabla_x f(x_0)^T (x - x_0)$$

Some common gradients

For $f(x) = a^T x$ gradient is given by $\nabla_x f(x) = a$

$$\frac{\partial f(x)}{\partial x_i} = \frac{\partial}{\partial x_i} \sum_{i=1}^{n} a_i x_i = a_i$$

For $f(x) = \frac{1}{2} x^T Q x$, gradient is given by $\nabla_x f(x) = \frac{1}{2}(Q + Q^T)x$
or just $\nabla_x f(x) = Qx$ if Q is symmetric ($Q = Q^T$)

How do we find $\nabla_x f(x) = 0$?

Direct solution: In some cases, it is possible to analytically compute the $x^\star$ such that $\nabla_x f(x^\star) = 0$

Example:

$$f(x) = 2x_1^2 + x_2^2 + x_1 x_2 - 6x_1 - 5x_2$$

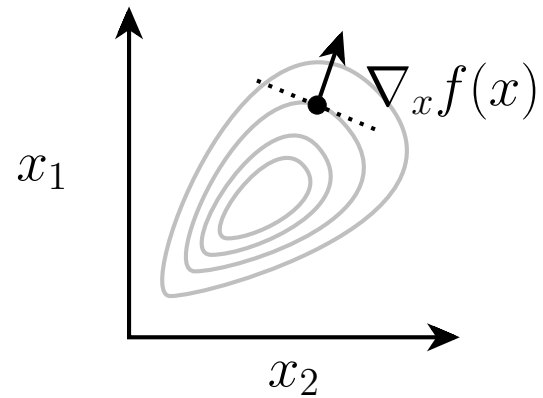
$$\implies \nabla_x f(x) = \begin{bmatrix} 4x_1 + x_2 + 6 \\ 2x_2 + x_1 + 5 \end{bmatrix}$$

$$\implies x^\star = \begin{bmatrix} 4 & 1 \\ 1 & 2 \end{bmatrix}^{-1} \begin{bmatrix} 6 \\ 5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Iterative methods: more commonly the condition that the gradient equal zero will not have an analytical solution, require iterative methods

Gradient descent

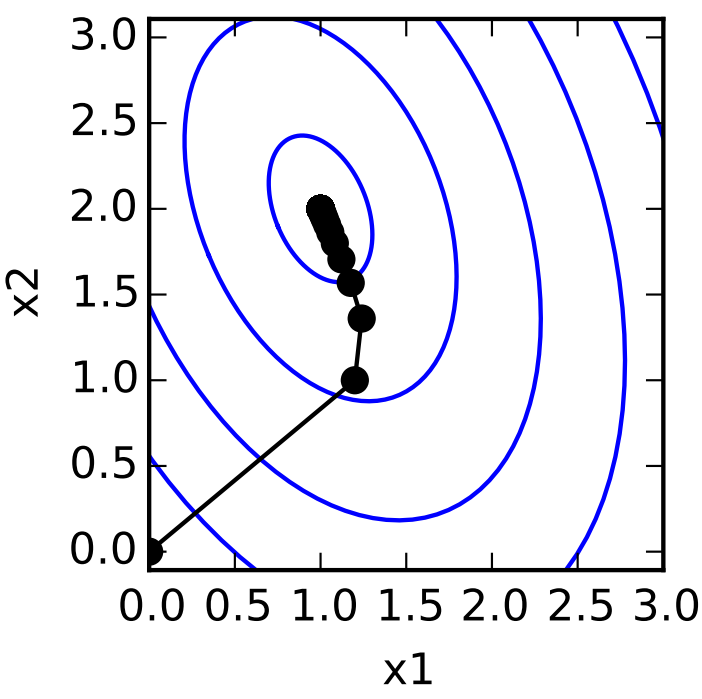
The gradient doesn't just give us the optimality condition, it also points in the direction of "steepest ascent" for the function f

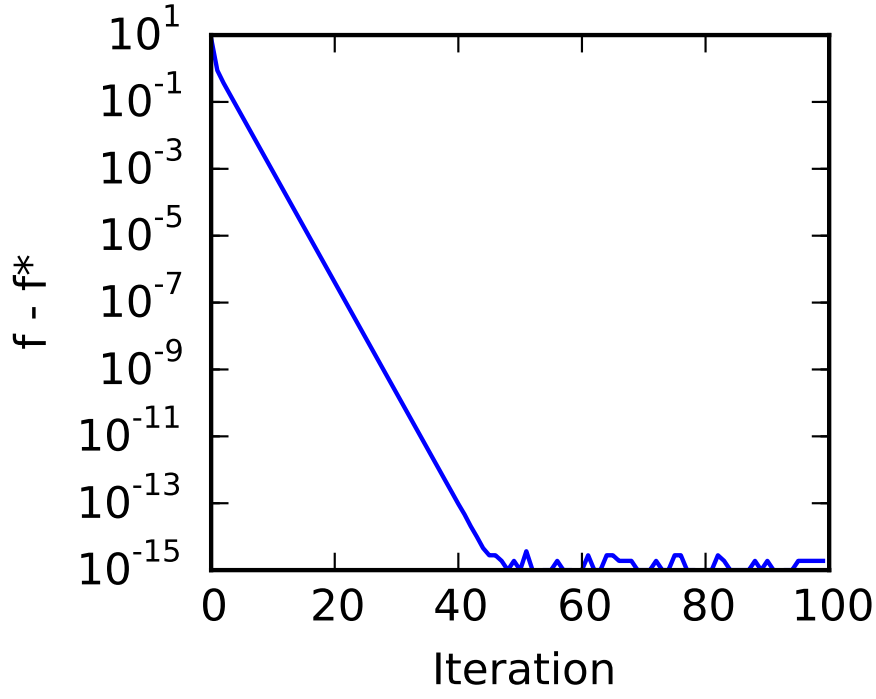


Motivates the *gradient descent* algorithm, which repeatedly takes steps in the direction of the negative gradient

$$\text{Repeat: } x \leftarrow x - \alpha \nabla_x f(x)$$

for some *step size* $\alpha > 0$



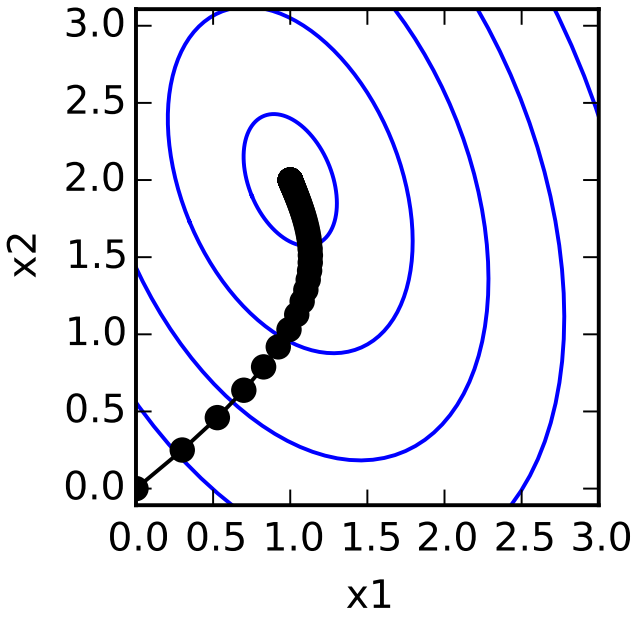


100 iterations of gradient descent on function

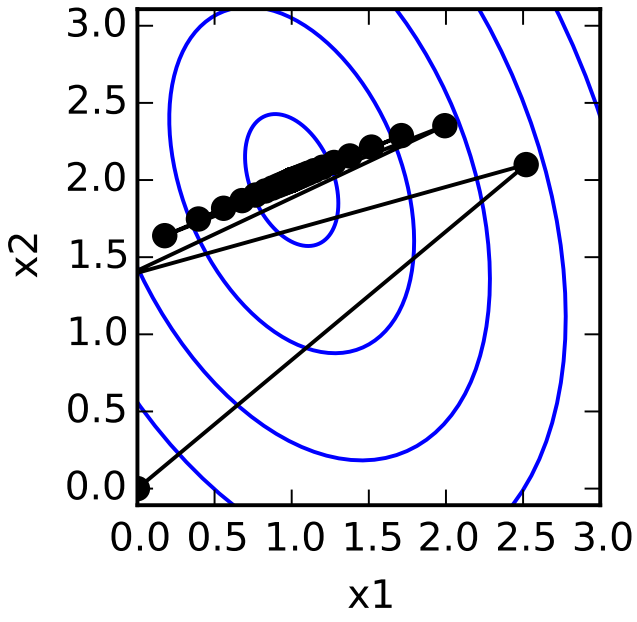
$$f(x) = 2x_1^2 + x_2^2 + x_1 x_2 - 6x_1 - 5x_2$$

How do we choose step size α?

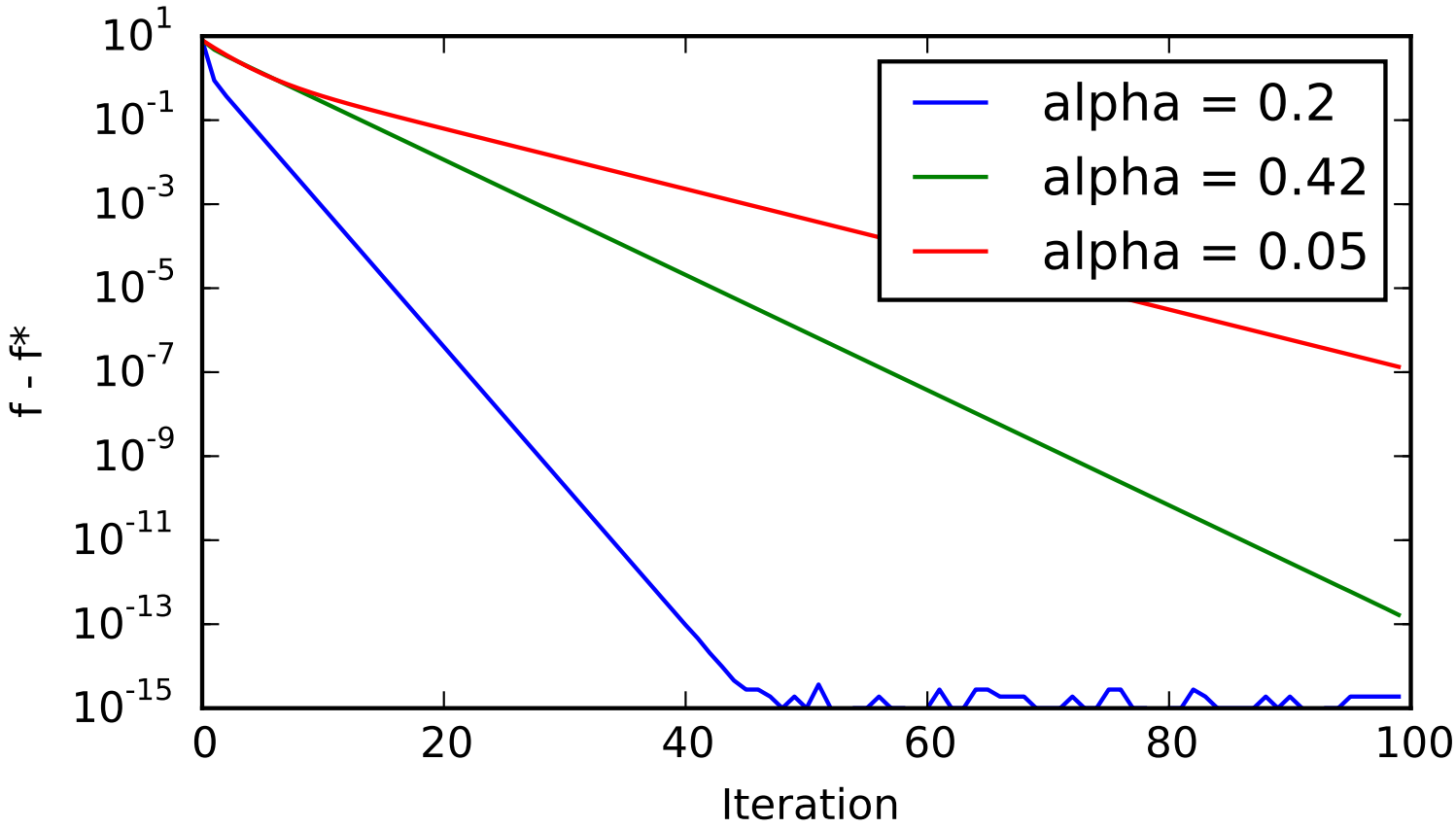
Choice of α plays a big role in convergence of algorithm



$\alpha = 0.05$



$\alpha = 0.42$



Convergence of gradient descent for different step sizes

If we know gradient is Lipschitz continuous with constant L, step size $\alpha = 1/L$ is good in theory and practice

But what if we don't know Lipschitz constant, or derivative has unbounded Lipschitz constant?

Idea #1 ("exact" line search): want to choose α to minimize $f(x_0 + \alpha \nabla f(x_0))$ for current iterate x_0; this is just another optimization problem, but with a *single* variable α

Idea #2 (backtracking line search): try a few α's on each iteration until we get one that causes a suitable decrease in the function

Backtracking line search

function $\alpha = \text{Backtracking-Line-Search}(x, f, \Delta x, \alpha_0, \gamma, \beta)$

// x: current iterate

// f: function being optimized

// Δx: descent direction (i.e., $\Delta x = -\nabla_x f(x)$)

// α_0: initial step size

// $\gamma \in (0, 0.5), \beta \in (0, 1)$: backtracking parameters

$\alpha \leftarrow \alpha_0$

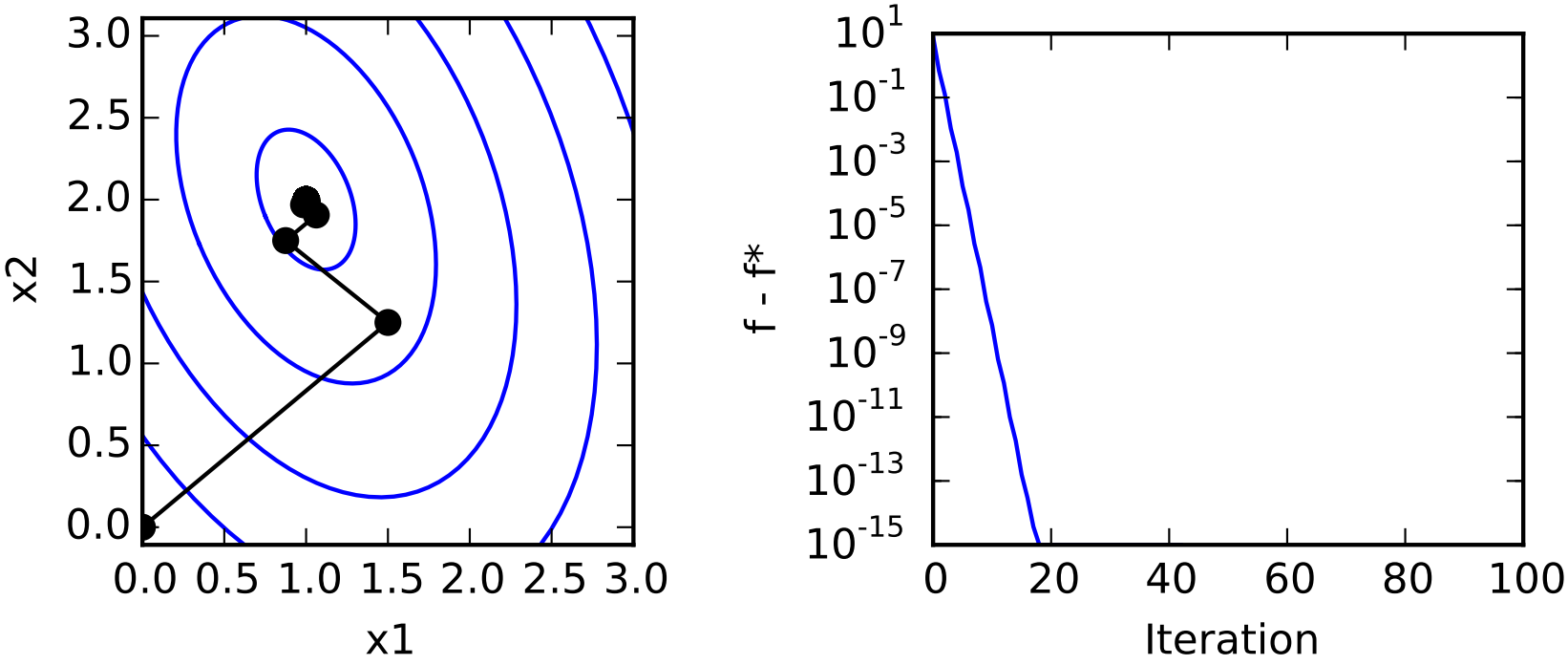
while $f(x + \alpha \Delta x) > f(x) + \gamma \alpha \nabla_x f(x)^T \Delta x$

$\alpha \leftarrow \alpha \cdot \beta$

return α

Common choices are $\gamma = 0.001$, $\beta = 0.5$

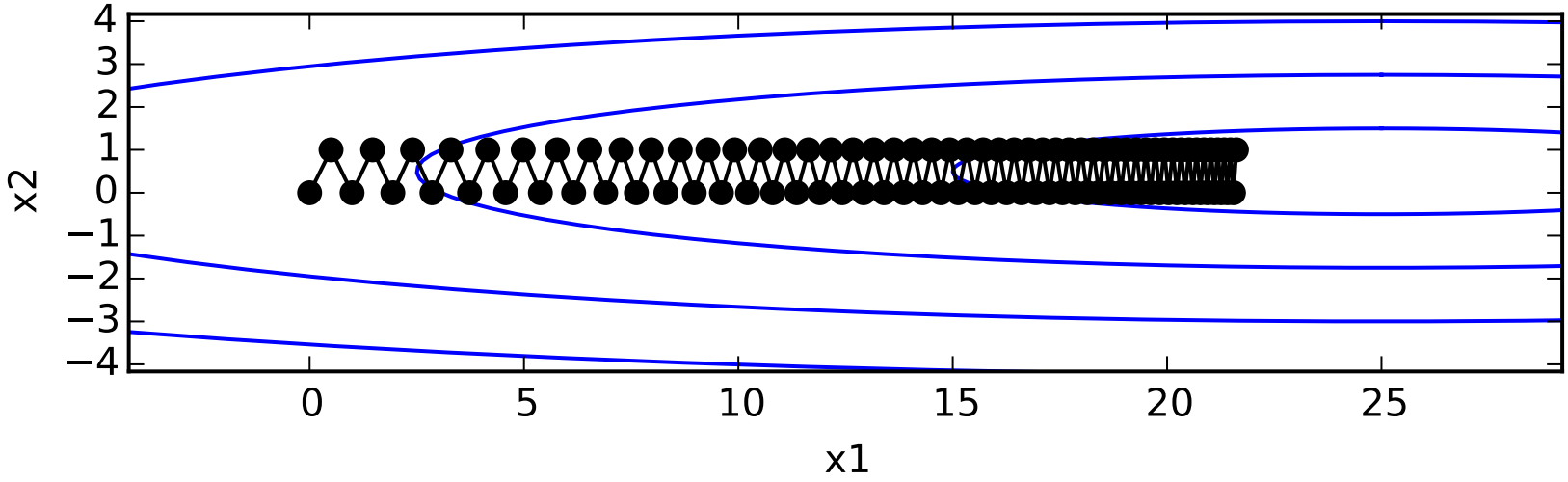
Intuition: for small α, by first order Taylor approximation $f(x + \alpha \Delta x) \approx f(x) + \alpha \nabla_x f(x)^T \Delta x$; backtracking line search makes α smaller until the inequality holds, but scaled by γ



100 iterations of gradient descent with backtracking line search on function

$$f(x) = 2x_1^2 + x_2^2 + x_1 x_2 - 6x_1 - 5x_2$$

Trouble with gradient descent



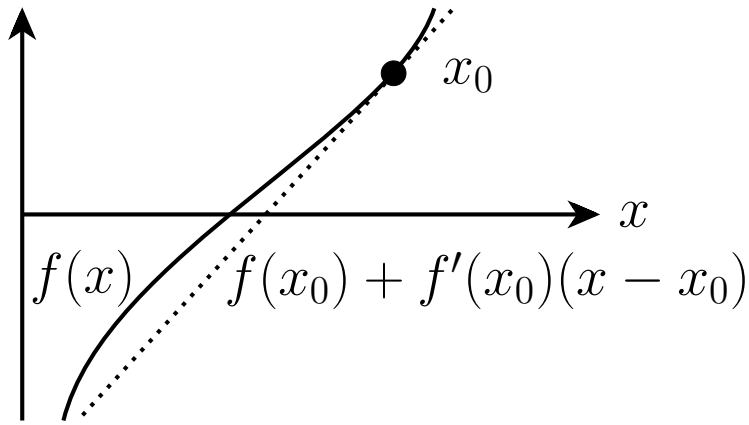
Gradient descent with backtracking line search on function

$$f(x) = 0.01x_1^2 + x_2^2 - 0.5x_1 - x_2$$

Gradient is given by $(0.02x_1 - 0.5, 2x_2 - 1)$ which is very poorly scaled (x_2 components are much bigger than x_1)

Motivates approaches that “automatically” find the right scaling

Newton's root finding method



Recall Newton's method for finding a zero (root) of a univariate function $f(x)$

$$\text{Repeat: } x \leftarrow x - \frac{f(x)}{f'(x)}$$

To optimize some univariate function g, we could use Newton's method to find a zero of the *derivative*, via the updates

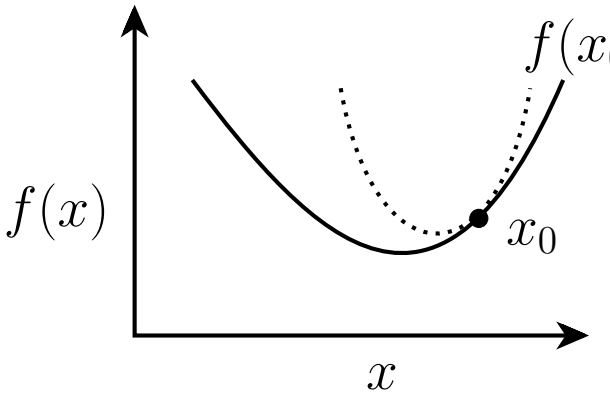
$$\text{Repeat: } x \leftarrow x - \frac{g'(x)}{g''(x)}$$

Hessian Matrix

To apply Newton's method to optimize a multivariate function, we need a generalization of the second derivative, called the *Hessian*

For a function $f : \mathbb{R}^n \to \mathbb{R}$, the Hessian is an $n \times n$ matrix of all second derivatives

$$\nabla_x^2 f(x) \in \mathbb{R}^{n \times n} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$



Hessian gives second order Taylor approximation of f at a point x_0

$$f(x) \approx f(x_0) + \nabla_x f(x_0)^T (x - x_0) + \frac{1}{2}(x - x_0)^T \nabla_x^2 f(x_0)(x - x_0)$$

Because $\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}$ (i.e., it doesn't matter which order we differentiate in), the Hessian for any function is always a symmetric matrix

For convex function f, the Hessian is *positive semidefinite* (all it's eigenvalues are greater than or equal to zero)

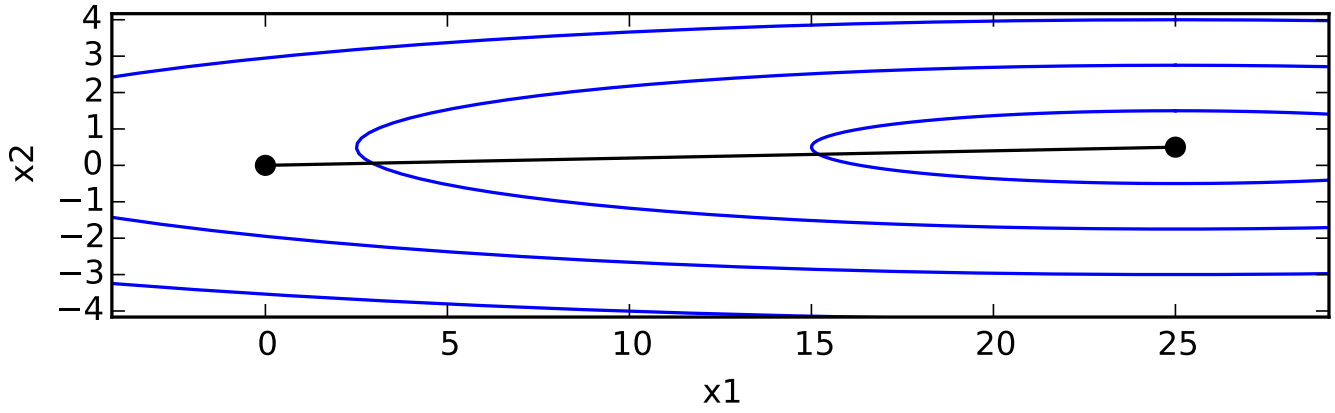
Newton's Method

(Multivariate) Newton's method optimizes a function $f : \mathbb{R}^n \to \mathbb{R}$ by the iterations

$$\text{Repeat: } x \leftarrow x - \alpha(\nabla_x^2 f(x))^{-1} \nabla_x f(x)$$

where α is a step size

Can choose α via backtracking line search, with $\Delta x = -(\nabla_x^2 f(x))^{-1} \nabla_x f(x)$ and with $\alpha_0 = 1$ (we always want to be able to take a "full" Newton step if possible)



For our previous example, Newton's method finds the *exact* solution in a single step (holds true for any convex quadratic function)

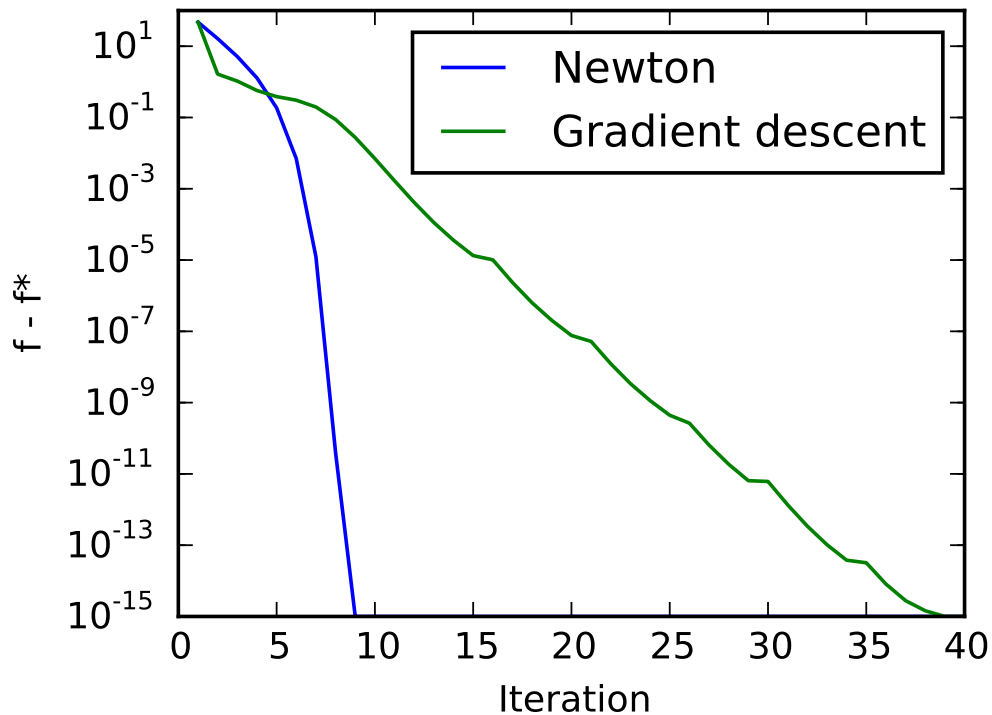
But for optimizing general convex functions, Newton's method is usually *much* faster than gradient descent, the preferred method *provided it is feasible to compute and invert the Hessian*

Newton's method is invariant to linear reparameterizations: if $g(x) = f(Ax)$, then optimizing g with Newton's method gives the exact same iterates as optimizing f

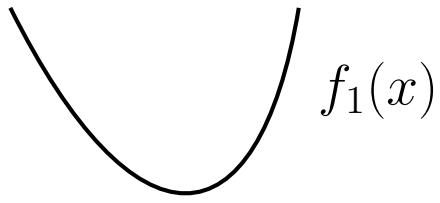
Example: Newton's method

$$f(x_1, x_2) = \exp(x_1 + 3x_2 - 0.1) + \exp(x_1 - 3x_2 - 0.1) + \exp(-x_1 - 0.1)$$

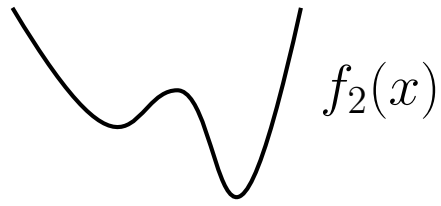
Convergence of Newton's method vs. gradient descent



Handling nonconvexity



Convex function



Nonconvex function

For nonconvex f, a choice: attempt to find a *global optimum* (hard, will require grid search in the worst case) or *local optimum* (relatively easy, can apply the methods above ignoring nonconvexity)

The issue with general nonconvex continuous optimization is that (unlike integer programming), there is relatively little structure to exploit, typically need to fall back to exponential-time grid search

In practice, for the vast majority of practical nonconvex problem (e.g. deep Networks, probabilistic graphical models with missing variables), people are satisfied with finding local optima

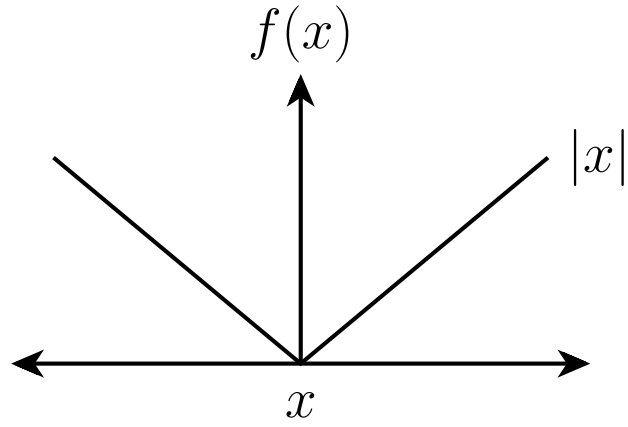
One subtle issue: because Hessian is not positive definite, Hessian is often poorly conditioned or non-invertible, leads to very poor Newton steps

In practice, gradient descent (or more generally, methods based upon just gradient evaluations) are more common for non-convex functions

Handling nonsmooth functions

Nonsmooth functions may not have gradient or Hessian

Example $f(x) = |x|$ does not have a derivative at $x = 0$

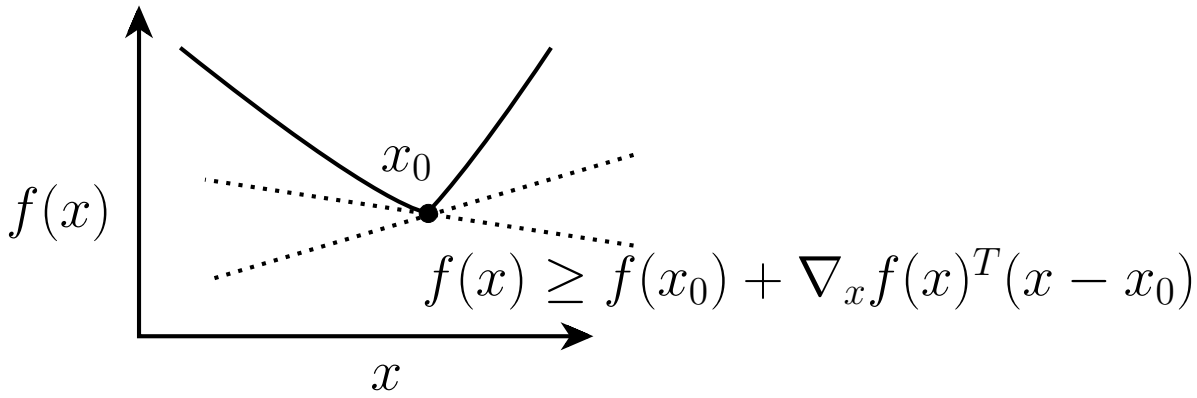


Not a minor issue because the function is “only” non-differentiable at one point, the problem is that the solutions often lie precisely at these non-differentiable points (cf. linear programming)

Subgradients

Although nonsmooth convex functions do not have gradients everywhere, they do have a *subgradient at every point*

A subgradient is something "like" a gradient, in that for a convex function it must lie below the function everywhere

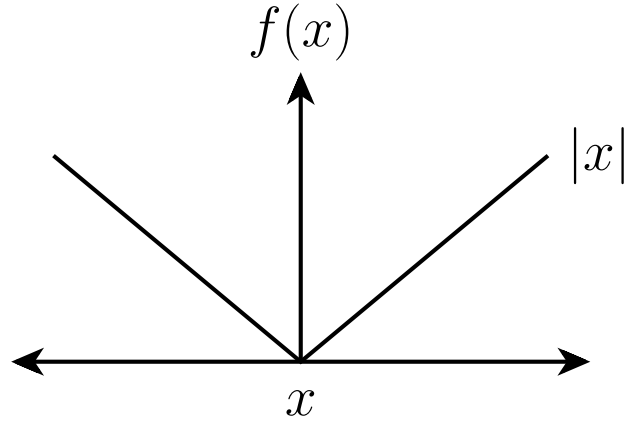


Example: $f(x) = |x|$, subgradients are given by

$$\nabla_x f(x) = \begin{cases} -1 & x < 0 \\ 1 & x > 0 \\ g \in [0, 1] & x = 0 \end{cases}$$

Can run gradient descent (now called *subgradient descent*) just as before

One subtle point: a constant step size (no matter how small) may never converge exactly (similar issues for backtracking line search)



Instead, common to use a decreasing sequence of step sizes, e.g.

$$\alpha_t = \alpha_0/(n_0 + t)$$

No “easy” analogue for Newton’s method when we only have subgradients

But, a great deal of recent work in how to solve nonsmooth optimization problems, especially in cases where the function has a smooth and a nonsmooth portion

Proximal algorithms have received a great deal of attention in this setting in recent years (e.g. Parikh et al., 2014)

Outline

Introduction to optimization

Types of optimization problems

Unconstrained optimization

Constrained optimization

Practical optimization

Constrained optimization

Recall constrained optimization problem

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & h_i(x) = 0, \quad i = 1, \ldots, p \end{aligned}$$

and its convex variant

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{aligned}$$

with f and all g_i convex

Seemingly much more challenging, we need to both find a *feasible* solution and an optimal solution amongst these feasible solutions

Projected gradient methods

Suppose we can easily solve the *projection* subproblem

$$\mathrm{Proj}(v) = \operatorname*{argmin}_x \|x - v\|_2 \text{ subject to } \begin{array}{ll} g_i(x) \leq 0, & i = 1, \ldots, m \\ h_i(x) = 0, & i = 1, \ldots, p \end{array}$$

Then we can use a simple adjustment to gradient descent, called *projected gradient descent*

$$\text{Repeat: } x \leftarrow \mathrm{Proj}(x - \alpha \nabla_x f(x))$$

But doesn't solving the projection seem just as hard as solving the problem to begin with?

The good news: some constraints have very simple forms of project

Example: $x \geq 0$:

$$\text{Proj}(v) = \underset{x \geq 0}{\text{argmin}} \, \|x - v\|_2 = \max\{x, 0\} \text{ (applied elementwise)}$$

Example: $Ax = b$

$$\text{Proj}(v) = \underset{x : Ax = b}{\text{argmin}} \, \|x - v\|_2 = x - A^T(AA^T)^{-1}(Ax - b)$$

Important: just because we have a closed form projection for each of two sets, $\mathcal{C}_1$, $\mathcal{C}_2$, does *not* give us a closed form projection onto their intersection $\mathcal{C}_1 \cap \mathcal{C}_2$

Duality and the KKT conditions

For a general constrained problem, we consider the Lagrangian

$$\mathcal{L}(x, \lambda, \nu) = f(x) + \sum_{i=1}^{m} \lambda_i g_i(x) + \sum_{i=1}^{p} \nu_i h_i(x)$$

Just like for the linear programming case, we have

$$\max_{\lambda \geq 0, \nu} \mathcal{L}(x, \lambda, \nu) = \begin{cases} f(x) & x \text{ feasible} \\ \infty & \text{otherwise} \end{cases}$$

Thus, we can rewrite our optimization problem as

$$\underset{x}{\text{minimize}} \ \underset{\lambda \geq 0, \nu}{\text{maximize}} \ \mathcal{L}(x, \lambda, \nu)$$

Because optimal $(x^\star, \lambda^\star, \nu^\star)$ must have $f(x^\star) = \mathcal{L}(x^\star, \lambda^\star, \nu^\star)$ and $\nabla_x \Lambda(x^\star, \lambda^\star, \nu^\star) = 0$, we have the following conditions

1. $g_i(x^\star) \leq 0, \ \forall i = 1, \ldots, m$

2. $h_i(x^\star) = 0, \ \forall i = 1, \ldots, p$

3. $\lambda_i^\star \geq 0, \ \forall i = 1, \ldots, m$

4. $\sum_{i=1}^{m} \lambda_i^\star g_i(x^\star) = 0 \implies \lambda_i^\star g_i(x^\star) = 0, \ \forall i = 1, \ldots, m$

5. $\nabla_x f(x^\star) + \sum_{i=1}^{x} \lambda_i^\star \nabla_x g_i(x^\star) + \sum_{i=1}^{p} \nu_i^\star \nabla_x h_i(x^\star) = 0$

These are the *Karush-Kuhn-Tucker* (KKT) equations

Equality constrained optimization

For convex f, consider the optimization problem

$$\begin{aligned} &\underset{x}{\text{minimize}} && f(x) \\ &\text{subject to} && Ax = b \end{aligned}$$

KKT optimality conditions for this problem are

$$\begin{aligned} \nabla_x f(x^\star) + A^T \nu^\star &= 0 \\ Ax^\star - b &= 0 \end{aligned}$$

Just like for unconstrained case, after a bit of derivation, we can use Newton's method to find a zero of this system by repeatedly solving the linear system

$$\begin{bmatrix} \nabla_x^2 f(x) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \nu \end{bmatrix} = \begin{bmatrix} \nabla_x f(x) + A^T \nu \\ Ax - b \end{bmatrix}$$

and then updating

$$x \leftarrow x - \alpha \Delta x, \quad \nu \leftarrow \nu - \alpha \Delta \nu$$

One subtle point: because we are both maximizing and minimizing the Lagrangian over x and ν respectively, we need a slight modification of backtracking line search, updating $\alpha \leftarrow \beta\alpha$ until

$$\left\| \begin{bmatrix} \nabla_x f(x + \alpha \Delta x) + A^T(\nu + \Delta \nu) \\ A(x + \Delta x) - b \end{bmatrix} \right\|_2 \leq (1 - \gamma\alpha) \left\| \begin{bmatrix} \nabla_x f(x) + A^T \nu \\ Ax - b \end{bmatrix} \right\|_2$$

Inequality constrained optimization

Consider the full convex constrained optimization problem, with convex f, g_i

$$\begin{aligned} \underset{x}{\text{minimize}} \;\; & f(x) \\ \text{subject to} \;\; & g_i(x) \leq 0, \quad i = 1, \ldots, m \\ & Ax = b \end{aligned}$$

KKT optimality conditions for this problem are

$$\begin{aligned} r_x &\equiv \nabla_x f(x^\star) + \sum_{i=1}^{m} \lambda_i^\star g_i(x^\star) + A^T \nu^\star = 0 \\ (r_\lambda)_i &\equiv \lambda_i g_i(x) = 0, \quad i = 1, \ldots, p \\ r_\nu &\equiv Ax^\star - b = 0 \end{aligned}$$

plus the condition that $g_i(x) \leq 0$

To apply Newton's method to find a solution to this system, the constraint that $\lambda_i g_i(x) = 0$ is difficult, instead use the constraint that $\lambda_i g_i(x) = t$ and bring $t \to 0$ as algorithm progresses

Newton update becomes

$$\begin{bmatrix} \nabla_x^2 f(x) + \sum_{i=1}^m \lambda_i \nabla_x^2 g_i(x) & Dg(x)^T & A^T \\ \operatorname{diag}(\lambda) Dg(x) & \operatorname{diag}(f(x)) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta \lambda \\ \Delta \nu \end{bmatrix} = \begin{bmatrix} r_x \\ r_\lambda + t1 \\ r_\nu \end{bmatrix}$$

where

$$Dg(x) = \begin{bmatrix} \nabla_x g_1(x)^T \\ \vdots \\ \nabla_x g_m(x)^T \end{bmatrix}$$

This is the *primal-dual interior point method*, which is the state of the art in exactly solving convex optimization problems

Handling nonconvexity

Solving, or even finding a feasible solution for, general nonconvex constrained optimization problems is very challenging

Consider $h_i(x) = x_i(1 - x_i) = 0$, equivalent to $x \in \{0, 1\}$, and we know it is NP-hard to find a feasible solution to an IP

Nonetheless, there are a few methods that can work well in practice to (locally) find hopefully feasible solutions

One popular approach: sequential convex programming, iteratively attempt to solve *penalized, unconstrained* objective

$$\underset{x}{\text{minimize}} \; f(x) + \mu \sum_{i=1}^{m} |g_i(x)|_+ + \mu \sum_{i=1}^{p} |h_i(x)|$$

for $\mu > 0$ and $|x|_+ = \max\{0, x\}$

Approximate this with the convex problem

$$\underset{x:||x-x_0||\leq\epsilon}{\text{minimize}} \; \tilde{f}(x, x_0) + \mu \sum_{i=1}^{m} |\tilde{g}_i(x, x_0)|_+ + \mu \sum_{i=1}^{p} |\tilde{h}_i(x, x_0)|$$

where $\tilde{f}(x, x_0) = f(x_0) + \nabla_x f(x_0)(x - x_0)$ (and similarly for $\tilde{g}_i$, $\tilde{h}_i$) are first order Taylor approximations

For large enough μ, if there is a "nearby" feasible solution to x_0, we will often find a point that satisfies constraints exactly

Handling nonsmoothness

A wonderful property of many nonsmooth functions is that they can be represented by smooth constrained functions (so in some sense nonsmoothness is *easier* for constrained problems than unconstrained)

Consider problem

$$\underset{x}{\text{minimize}} \quad \|Ax - b\|_2^2 + \mu \sum_{i=1}^{n} |x_i|$$

This can be written as

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \|Ax - b\|_2^2 + \mu \sum_{i=1}^{n} y_i \\ \text{subject to} \quad & -y_i \leq x_i \leq y_i, \quad \forall i = 1, \ldots, n \end{aligned}$$

Outline

Introduction to optimization

Types of optimization problems

Unconstrained optimization

Constrained optimization

Practical optimization

Practically solving optimization problems

The good news: for many classes of optimization problems, people have already done all the “hard work” of developing numerical algorithms

A wide range of tools that can take optimization problems in “natural” forms and compute a solution

Some well-known libraries: CVX (MATLAB), YALMIP (MATLAB), AMPL (custom language), GAMS (custom language), cvxpy (Python)

cvxpy

Python library for specifying and solving convex optimization problems

Available at `http://www.cvxpy.org`

Under active development, but at a relatively stable point

Image deblurring


$$\underset{x}{\text{minimize}} \quad \|K * x - y\|_2^2 + \mu \left(\sum_{i=1}^{n-1} |x_{mi} - x_{m(i+1)}| + \sum_{i=1}^{m-1} |x_{ni} - x_{n(i+1)}| \right)$$

cvxpy code:

```
import cvxpy as cp
Y,K = ...
X = cp.Variable(Y.shape[0], Y.shape[1])
f = (cp.sum_squares(K*cp.vec(X) - cp.vec(Y)) +
     mu*cp.sum_entries(cp.abs(X[:,:-1] - X[:,1:])) +
     mu*cp.sum_entries(cp.abs(X[:-1,:] - X[1:,:])))
constraints = []
result = cp.Problem(cp.Minimize(f), constraints).solve()
```