Math 6880/7875: Advanced Optimization
Descent algorithms, Part I

Akil Narayan[1]

[1]Department of Mathematics, and Scientific Computing and Imaging (SCI) Institute
University of Utah

Feburary 8, 2022

Descent algorithms

A foundational computational algorithmic idea for unconstrained + smooth optimization is a descent method.

Many optimization tools are variants of descent methods. We'll tour such methods.

- The basic descent method
- First- and second-order methods
- Convergence guarantees
- quasi-Newton methods

The descent method

Consider the unconstrained optimization, (assume f is smooth)

$$\min_{x \in \mathbb{R}^n} f(x)$$

When we cannot solve this analytically, algorithms are our recourse.

Most optimization algorithms are *iterative*, meaning that an initial guess is repeatedly improved.

The most common method for performing the "improvement" is to geometrically travel in a descent direction.

Definition

Given a continuous function f and a point $x_0 \in \mathbb{R}^n$, a vector $d \in \mathbb{R}^n$ is a *descent direction* for f at x_0 if, there exists some $\epsilon = \epsilon(f, x_0, d) > 0$ such that,

$$f(x_0 + \delta d) < f(x_0), \qquad \forall \ 0 < \delta \leqslant \epsilon.$$

If x_0 is a local minimum, there are no descent directions.

The descent method

Consider the unconstrained optimization,

$$\min_{x \in \mathbb{R}^n} f(x)$$

When we cannot solve this analytically, algorithms are our recourse.

Most optimization algorithms are *iterative*, meaning that an initial guess is repeatedly improved.

The most common method for performing the "improvement" is to geometrically travel in a descent direction.

Definition

Given a continuous function f and a point $x_0 \in \mathbb{R}^n$, a vector $d \in \mathbb{R}^n$ is a *descent direction* for f at x_0 if, there exists some $\epsilon = \epsilon(f, x_0, d) > 0$ such that,

$$f(x_0 + \delta d) < f(x_0), \qquad \forall \ 0 < \delta \leqslant \epsilon.$$

If x_0 is a local minimum, there are no descent directions.

Some pseudocode

all instances of "n" ⇒ "k"
n: dimension of x, k: iteration index

The anatomy of essentially every descent method is as follows:

1. Begin with an initial guess x_0, set ~~$n = 0$~~ $k=0$
2. Identify a descent direction d_n at x_n
3. Identify a stepsize $\alpha_n > 0$
4. Define $x_{n+1} = x_n + \alpha_n d_n$
5. If x_{n+1} is good enough, stop. Otherwise set $n \leftarrow n + 1$, return to step 2.

Things in blue are crucial decisions/inputs to the algorithm:

- An initial guess x_0
- A strategy for computing a descent direction d_n
- A strategy for computing a stepsize α_n
- A way to determine when an iterate has converged, terminating the algorithm

Some pseudocode

The anatomy of essentially every descent method is as follows:

1. Begin with an initial guess x_0, set $n = 0$
2. Identify a descent direction d_n at x_n
3. Identify a stepsize $\alpha_n > 0$
4. Define $x_{n+1} = x_n + \alpha_n d_n$
5. If x_{n+1} is good enough, stop. Otherwise set $n \leftarrow n + 1$, return to step 2.

Things in blue are crucial decisions/inputs to the algorithm:

- An initial guess x_0
- A strategy for computing a descent direction d_n
- A strategy for computing a stepsize α_n
- A way to determine when an iterate has converged, terminating the algorithm

Initial guess and termination

"n" ← "k"

Strategies for initial guess and termination criteria are in some sense the easiest to describe.

Common termination strategies:

- $\|x_n - x_{n-1}\| < \epsilon$ (Does this imply x_n is close to optimal?)
- $|f(x_n) - f(x_{n-1})| < \epsilon$ (Does this imply $f(x_n)$ is close to optimal?)
- $\|\nabla f(x_n)\| < \epsilon$ (Does this imply x_n is close to stationary?)
- Combinations of the above

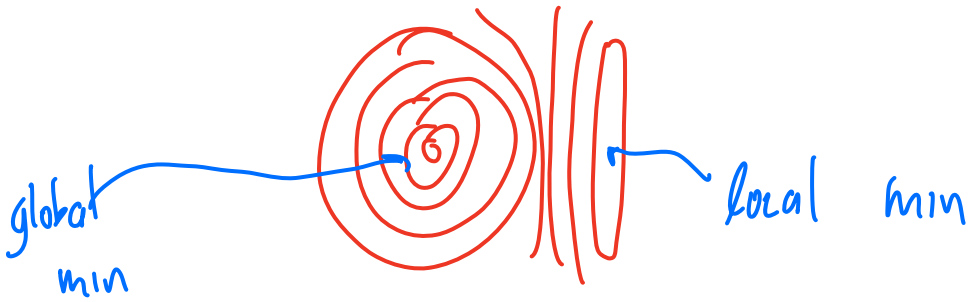
Frequently there is not a clear "good" choice.

Initial guess and termination

Strategies for initial guess and termination criteria are in some sense the easiest to describe.

Common initialization strategies:

- Start "close" to a local or global minimum
- Repitition: choose several initializations, optimize for all of them
- Randomization: randomly choose x_0
- Homotopy: Let f_m, $m > 0$ be some sequence of functions such that $f_m \to f$ in an appropriate sense.
  Choose x_0, optimize f_1 resulting in optimum $\widetilde{x}_1$.
  Set $x_0 = \widetilde{x}_1$, optimize f_2 resulting in optimum $\widetilde{x}_2$.
  $\vdots$
  This is sensible if f_m for small m is easier to optimize than f.



Initial guess and termination

Strategies for initial guess and termination criteria are in some sense the easiest to describe.

Deciding on a descent direction and stepsize are typically the meat of developing good optimization algorithms.

The "classical" stuff

Descent directions

$$x_{k+1} = x_k + \alpha_k d_k.$$

We assume d_k is a descent direction, and for smooth f this is the same as,

$$d_k^T \nabla f(x_k) < 0.$$

Nearly all descent algorithms use an update direction given by,

$$d_k = -G_k^{-1} \nabla f(x_k),$$

where G_k is some symmetric, positive-definite matrix.
(Note this condition on G_k guarantees d_k is a diescent direction.)
For example:

- Steepest/Gradient descent: $G_k = I$
- Newton's method: $G_k = \nabla^2 f(x_k)$
- quasi-Newton methods: $G_k \approx \nabla^2 f(x_k)$

Descent directions

$$x_{k+1} = x_k + \alpha_k d_k.$$

We assume d_k is a descent direction, and for smooth f this is the same as,

$$d_k^T \nabla f(x_k) < 0.$$

Nearly all descent algorithms use an update direction given by,

$$d_k = -G_k^{-1} \nabla f(x_k),$$

$\nabla f(x_k)^T d_k = -\nabla f(x_k)^T G_k^{-1} \nabla f(x_k) < 0$

where G_k is some symmetric, positive-definite matrix.
(Note this condition on G_k guarantees d_k is a diescent direction.)
For example:

- Steepest/Gradient descent: $G_k = I$
- Newton's method: $G_k = \nabla^2 f(x_k)$
- quasi-Newton methods: $G_k \approx \nabla^2 f(x_k)$

Steepest/gradient descent

$$x_{k+1} = x_k + \alpha_k d_k.$$

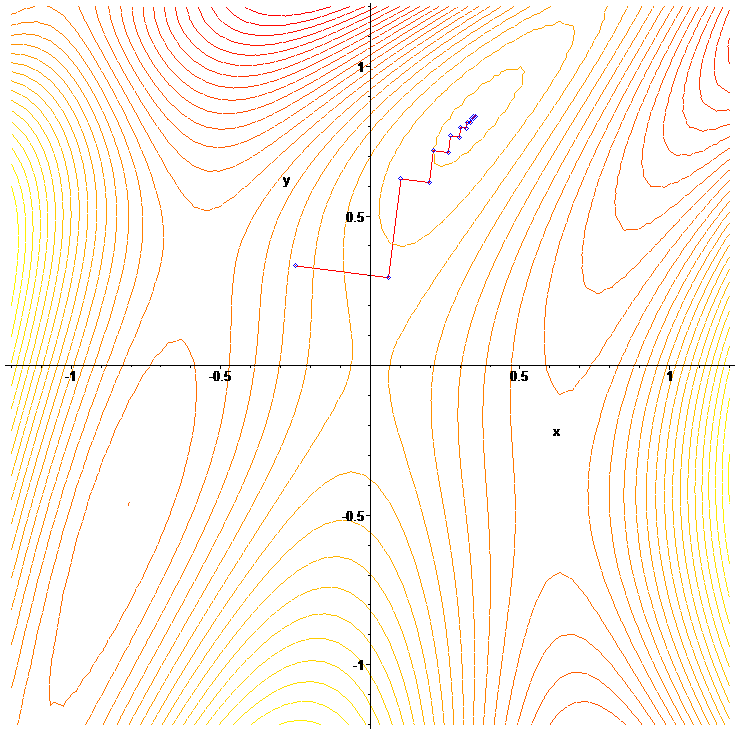
The direction d_k is chosen to infinitesimally decrease f the fastest:

$$d_k = -\nabla f(x_k).$$

($G_k = I$ on prev. slide.)

Note that steepest descent need not be a good idea.

"Cheap": ∇f is "ok" to compute



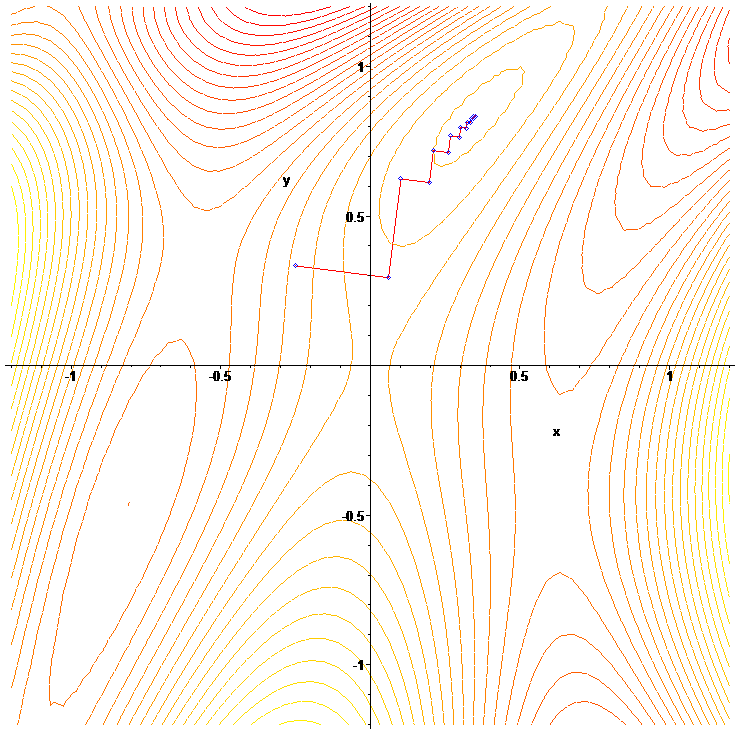
Steepest/gradient descent

$$x_{k+1} = x_k + \alpha_k d_k.$$

The direction d_k is chosen to infinitesimally decrease f the fastest:

$$d_k = -\nabla f(x_k).$$

Note that steepest descent need not be a good idea.



Steepest descent is first-order

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

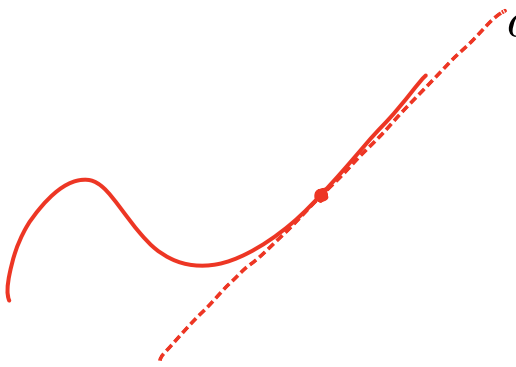
around $x = x_k$

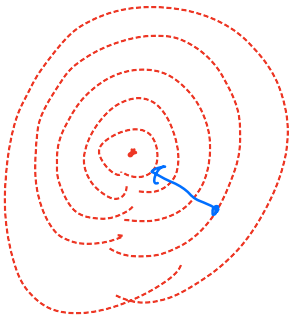
Steepest descent can be understood as a first-order Taylor expansion. First approximate:

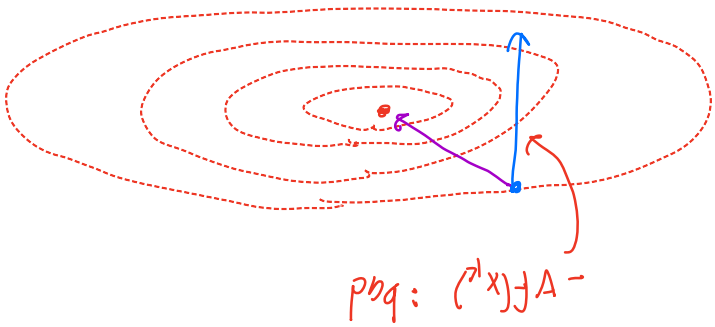
$$f(x) \approx f(x_k) + (x - x_k)^T \nabla f(x_k),$$

and choose x so that $(x - x_k)^T \nabla f(x_k)$ is minimized for fixed $\|x - x_k\|$:

$$d_k = x - x_k = -\nabla f(x_k)$$






bad : $-\nabla f(x_k)$

Scaling

One reason why steepest descent tends to produce poor iterates is because problems can be poorly scaled.

Consider

$$f(x) = x^T \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} x.$$

The global minimum is $x = 0$, but starting at $x = (0.5, 1)$ produces pretty bad descent directions.

Steepest descent is not *scale invariant*.

A simple strategy to mitigate poor scaling is diagonal scaling:

$$\min f(x) \quad \longrightarrow \quad \min g(x),$$

where $g(x) = f(Dx)$, with D a positive-definite diagonal matrix.

The hard part is computing D (which can change at every iteration).

Scaling

One reason why steepest descent tends to produce poor iterates is because problems can be poorly scaled.

Consider

$$f(x) = x^T \begin{pmatrix} 10 & 0 \\ 0 & 1 \end{pmatrix} x.$$

The global minimum is $x = 0$, but starting at $x = (0.5, 1)$ produces pretty bad descent directions.

Steepest descent is not *scale invariant*.

A simple strategy to mitigate poor scaling is diagonal scaling:

$$\min f(x) \quad \longrightarrow \quad \min g(x),$$

where $g(x) = f(Dx)$, with D a positive-definite diagonal matrix.

The hard part is computing D (which can change at every iteration).

How to chose D "efficently":

$$D = \sqrt{H_k^{-1}} \quad , \quad H_k^{-1} \simeq \text{diag}\left(\nabla^2 f(x_k)\right)$$

$$x^T A x \longrightarrow \underbrace{(x^T \sqrt{A})}_{\text{"}y\text{"}} (\sqrt{A}\, x)$$

2A: Hessian $(\nabla^2 f(x_k))$

$$\nabla^2 f(x_k) \simeq \text{diag}(\nabla^2 f(x_k))$$

$\text{diag}(\nabla^2 f(x_k))$: n elements

Newton's method

There are two equivalent ways to derive Newton's method.

For simplicity, we'll assume $\nabla^2 f(x_k) \succ 0$.

Newton's method

There are two equivalent ways to derive Newton's method.

For simplicity, we'll assume $\nabla^2 f(x_k) \succ 0$.

The first derivation: approximate f with a second-order Taylor expansion and minimize:

$$f(x) \approx f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T \nabla^2 f(x_k) d, \qquad d = x - x_k.$$

Newton's method

There are two equivalent ways to derive Newton's method.

For simplicity, we'll assume $\nabla^2 f(x_k) \succ 0$.

The first derivation: approximate f with a second-order Taylor expansion and minimize:

$$f(x) \approx f(x_k) + d\nabla f(x_k) + \frac{1}{2} d^T \nabla^2 f(x_k) d, \qquad d = x - x_k.$$

The right-hand side is a strictly convex function of d, so can be exactly minimized:

$$d = -\left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k).$$

Note that this relies on positive-definiteness of the Hessian, suggesting that this is only a good idea if f is locally convex around x_k....

Newton's method

There are two equivalent ways to derive Newton's method.

For simplicity, we'll assume $\nabla^2 f(x_k) \succ 0$.

The second derivation: Let's use Newton's method for nonlinear root-finding to compute stationary points.

Define,

$$g(x) := \nabla f(x) \quad \longrightarrow \quad \text{Solve for } x: \quad g(x) = 0$$

Note that $g : \mathbb{R}^n \to \mathbb{R}^n$.

Newton's method

There are two equivalent ways to derive Newton's method.

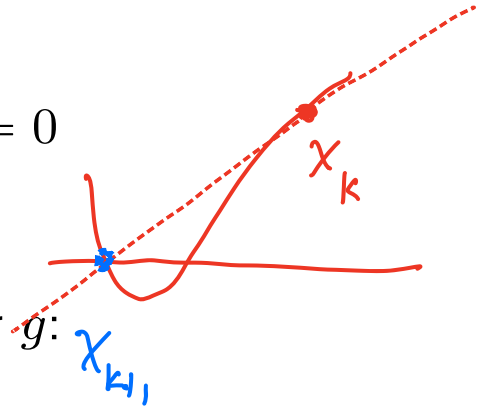
For simplicity, we'll assume $\nabla^2 f(x_k) \succ 0$.

The second derivation: Let's use Newton's method for nonlinear root-finding to compute stationary points.

Define,

$$g(x) := \nabla f(x) \quad \longrightarrow \quad \text{Solve for } x: \quad g(x) = 0$$

Note that $g : \mathbb{R}^n \to \mathbb{R}^n$.



Given a current iterate x_k, Newton's method for rootfinding for g:

$$x_{k+1} = x_k - (\nabla g(x_k))^{-1} g(x_k).$$

The quantity ∇g is a Jacobian matrix, with entries,

$$\nabla g(x) = \left(\frac{\partial}{\partial x_1} \nabla f(x) \quad \frac{\partial}{\partial x_2} \nabla f(x) \quad \cdots \quad \frac{\partial}{\partial x_n} \nabla f(x) \right) = \nabla^2 f(x),$$

so that we have

$$x_{k+1} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k) \implies x_{k+1} - x_k = - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k).$$

Newton's method and scaling

Note that Newton's method *exactly* minimizes positive-definite quadratic functions in a single step.

In particular, even poorly scaled quadratic functions are exactly minimized.

For this reason, Newton-type methods are called *scale invariant*.

Naturally there is a price to pay: computing $\nabla^2 f$ is much more expensive than ∇f.

Newton's method and scaling

Note that Newton's method *exactly* minimizes positive-definite quadratic functions in a single step.

In particular, even poorly scaled quadratic functions are exactly minimized.

For this reason, Newton-type methods are called *scale invariant*.

Naturally there is a price to pay: computing $\nabla^2 f$ is <u>much</u> more expensive than ∇f.

Stepsizes

We'll now discuss choosing stepsizes. Our update takes the form,

$$x_{k+1} = x_k + \alpha_k d_k.$$

We will always consider d_k to be a descent direction.

For notational simplicity, we'll assume k is fixed, and remove dependence of α_k, d_k on k. I.e., we have,

$$x_{k+1} = x_k + \alpha d.$$

Note: α and d typically always depend on k!

Let's assume d is chosen and fixed (as a descent direction).

What are some common ways that α is chosen?

Stepsizes

We'll now discuss choosing stepsizes. Our update takes the form,

$$x_{k+1} = x_k + \alpha_k d_k.$$

We will always consider d_k to be a descent direction.

For notational simplicity, we'll assume k is fixed, and remove dependence of α_k, d_k on k. I.e., we have,

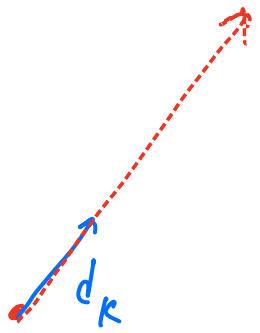
$$x_{k+1} = x_k + \alpha d.$$

Note: α and d typically always depend on k!

Let's assume d is chosen and fixed (as a descent direction).

What are some common ways that α is chosen?

Exact linesearch



The simplest approach is, unfortunately, the least practical.

Exact linesearch determines α through an optimization,

$$\alpha = \operatorname*{arg\,min}_{\beta>0} f(x_k + \beta d).$$

Things to note:

- The above problem is guaranteed to have a solution since d is a descent direction.
- This optimization is in principle much easier than the original problem: the above is a one-dimensional optimization instead of an n-dimensional one.
- This is still quite an expensive problem since several evaluations of f ~~(and probably ∇f are required)~~

Most popular approaches are inexact linesearch methods, based on various conditions.

Sufficient decrease

A particularly simple inexact method is that of sufficient decrease.

Locally near x_k, the function f behaves like its linear Taylor series,

$$f(x_k + \alpha d) \approx f(x_k) + \alpha d^T \nabla f(x_k).$$

This gives us an *expected* decrease: for a given small α, the improvement in f is approximately,

$$f(x_k + \alpha d) - f(x_k) \approx \alpha d^T \nabla f(x_k).$$

Of course, unless we get lucky the actual decrease will be smaller than this.

Sufficient decrease, or the Armijo condition, imposes the following condition on α:

$$f(x_k + \alpha d) - f(x_k) \leqslant c\alpha d^T \nabla f(x_k),$$

for a constant $c \in (0, 1)$. Choosing c is a bit of an art.

Sufficient decrease

A particularly simple inexact method is that of sufficient decrease.

Locally near x_k, the function f behaves like its linear Taylor series,

$$f(x_k + \alpha d) \approx f(x_k) + \alpha d^T \nabla f(x_k).$$

This gives us an *expected* decrease: for a given small α, the improvement in f is approximately,

$$f(x_k + \alpha d) - f(x_k) \approx \alpha d^T \nabla f(x_k).$$

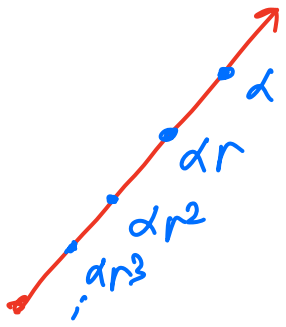
Of course, unless we get lucky the actual decrease will be smaller than this.

Sufficient decrease, or the Armijo condition, imposes the following condition on α:

$$f(x_k + \alpha d) - f(x_k) \leqslant c\alpha d^T \nabla f(x_k),$$

for a constant $c \in (0, 1)$. Choosing c is a bit of an art.

Backtracking



A very popular appraoch combines sufficient decrease with *backtracking*:

- Fix $c, r \in (0,1)$. Initialize some "large" $\alpha > 0$
- While sufficient decrease is *not* met, i.e., $f(x_k + \alpha d) - f(x_k) > c\alpha d^T \nabla f(x_k)$:
  - Set $\alpha \leftarrow \alpha r$

The while loop must terminate if d is a descent direction.
Typically, $r = r_k$ is chosen in a problem-dependent way.

The Wolfe conditions

One problem with sufficient decrease + backtracking: values α can be very small.

To mitigate small step sizes, we impose a stronger pair of conditions on α.

One condition is again sufficient decrease,

$$f(x_k + \alpha d) - f(x_k) \leqslant c\alpha d^T \nabla f(x_k), \tag{1a}$$

the second is the additional condition:

$$d^T \nabla f(x_k + \alpha d) \geqslant \tilde{c} d^T \nabla f(x_k). \tag{1b}$$

for some other constant $\tilde{c} \in (c, 1)$. The pair (1) are the Wolfe conditions.

The second, "curvature" Wolfe condition states that the one-dimensional function $\alpha \mapsto f(x_k + \alpha d)$ is less steep at x_{k+1} compared to x_k.

If this less steep condition fails for small α, it suggests that making α larger can significantly decrease the objective.

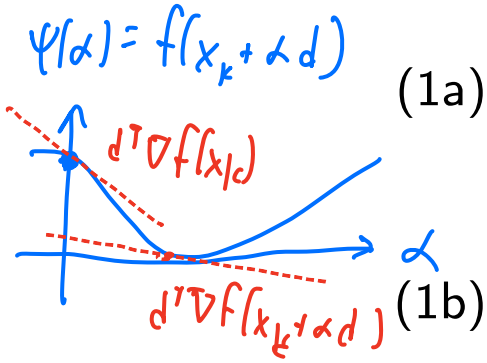
The Wolfe conditions

One problem with sufficient decrease + backtracking: values α can be very small.

To mitigate small step sizes, we impose a stronger pair of conditions on α.

One condition is again sufficient decrease,

$$f(x_k + \alpha d) - f(x_k) \leqslant c\alpha d^T \nabla f(x_k), \tag{1a}$$



the second is the additional condition:

$$d^T \nabla f(x_k + \alpha d) \geqslant \tilde{c} d^T \nabla f(x_k). \tag{1b}$$

for some other constant $\tilde{c} \in (c, 1)$. The pair (1) are the Wolfe conditions.

The second, "curvature" Wolfe condition states that the one-dimensional function $\alpha \mapsto f(x_k + \alpha d)$ is less steep at x_{k+1} compared to x_k.

If this less steep condition fails for small α, it suggests that making α larger can significantly decrease the objective.

Strong Wolfe conditions

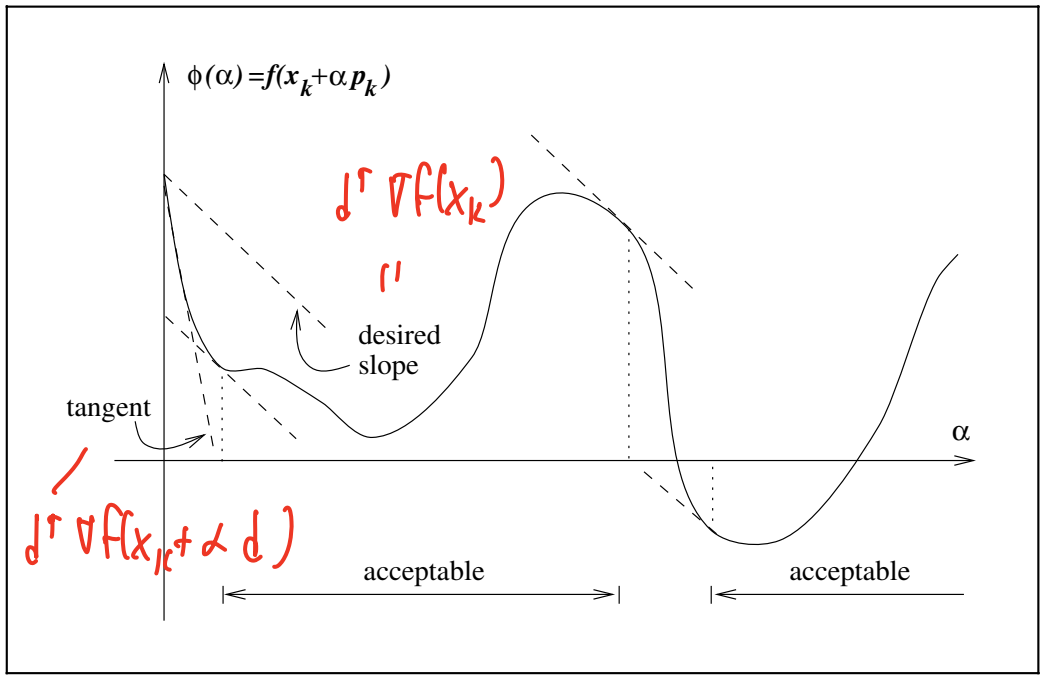


Image: Figure 3.5, Nocedal & Wright, *Numerical Optimization*

The *strong* Wolfe conditions strengthen the curvature condition to,

$$\left| d^T \nabla f(x_k + \alpha d) \right| \leqslant \tilde{c} \left| d^T \nabla f(x_k) \right|$$

which disallows large positive values of $f'(x_k + \alpha d)$.

Under mild assumptions on f and x_k, there is always some α satisfying the strong/Wolfe conditions.

Strong Wolfe conditions

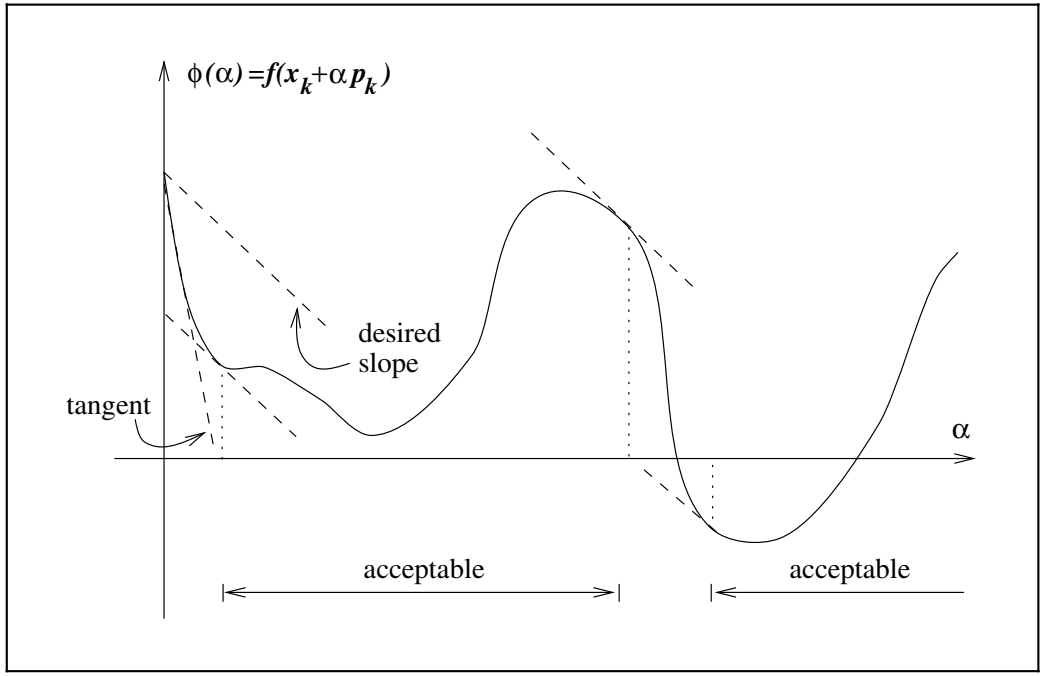


Image: Figure 3.5, Nocedal & Wright, *Numerical Optimization*

The *strong* Wolfe conditions strengthen the curvature condition to,

$$\left| d^T \nabla f(x_k + \alpha d) \right| \leqslant \tilde{c} \left| d^T \nabla f(x_k) \right|$$

which disallows large positive values of $f'(x_k + \alpha d)$.

Under mild assumptions on f and x_k, there is always some α satisfying the strong/Wolfe conditions.

The Goldstein conditions

$$x_{k+1} = x_k + \alpha d.$$

Yet *another* set of conditions are the Goldstein conditions on α:

$$f(x_k) + (1-c)\alpha d^T \nabla f(x_k) \leqslant f(x_k + \alpha d) \leqslant f(x_k) + c\alpha d^T \nabla f(x_k), \quad 0 < c < \frac{1}{2}.$$

Again, the goal is to mitigate small step sizes, but this can be too aggressive.

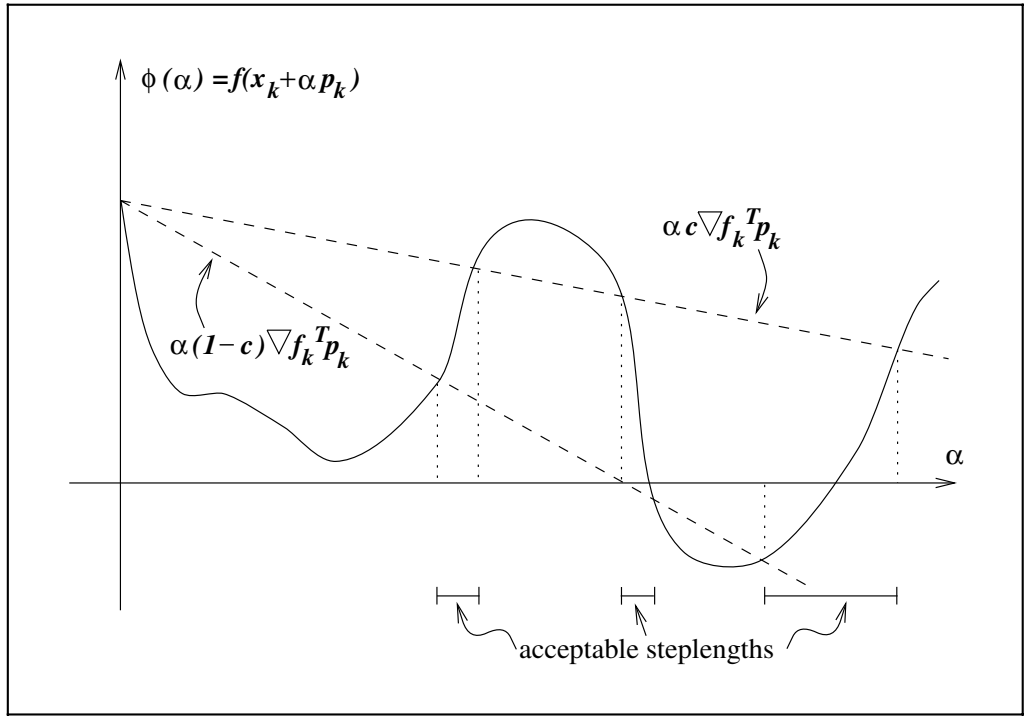


Image: Figure 3.6, Nocedal & Wright, *Numerical Optimization*

Convergence

The goal of descent algorithms is to compute a stationary point.
(Asking for more without extra conditions is unreasonable.)

A method is *globally convergent* if we can guarantee:

$$\lim_{k \uparrow \infty} \|\nabla f(x_k)\|_2 = 0.$$

To discuss convergence we rely on a measure of how parallel our chosen descent direction d_k is to $\nabla f(x_k)$ at each step:

$$\cos \theta_k := \frac{-d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|}.$$

One of the basic tools in convergence theory of descent methods is the Zoutendijk condition, stating that

$$\lim_{k \to \infty} \cos^2 \theta_k \|\nabla f(x_k)\|^2 < \infty.$$

Convergence

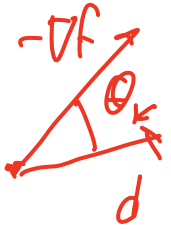
The goal of descent algorithms is to compute a stationary point.
(Asking for more without extra conditions is unreasonable.)

A method is *globally convergent* if we can guarantee:

$$\lim_{k \uparrow \infty} \|\nabla f(x_k)\|_2 = 0.$$

To discuss convergence we rely on a measure of how parallel our chosen descent direction d_k is to $\nabla f(x_k)$ at each step:

$$\cos\theta_k := \frac{-d_k^T \nabla f(x_k)}{\|d_k\| \|\nabla f(x_k)\|}.$$



One of the basic tools in convergence theory of descent methods is the Zoutendijk condition, stating that

$$\lim_{k \to \infty} \cos^2\theta_k \|\nabla f(x_k)\|^2 < \infty.$$

Zoutendijk to convergence

$$\sum_{k=1}^{\infty} \cos^2 \theta_k \|\nabla f(x_k)\|^2 < \infty.$$

Why is this useful? We can convert this into global convergence if,

$$|\cos \theta_k| \geqslant \delta > 0, \qquad k \in \mathbb{N}$$

for some δ.

$\cos\theta_k = 0 \iff \theta_k = \pi/2$

Then this implies:

$$\sum_{k=1}^{\infty} \|\nabla f(x_k)\|^2 < \infty \implies \lim_{k \uparrow \infty} \|\nabla f(x_k)\| = 0,$$

i.e., global convergence.