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

www.sci.utah.edu

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
- Trust region methods

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.

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

The anatomy of essentially every descent method is as follows:

1. Begin with an initial guess x_0, set $k = 0$
2. Identify a descent direction d_k at x_k
3. Identify a stepsize $\alpha_k > 0$
4. Define $x_{k+1} = x_k + \alpha_k d_k$
5. If x_{k+1} is good enough, stop. Otherwise set $k \leftarrow k + 1$, 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_k
- A strategy for computing a stepsize α_k
- 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 $k = 0$
2. Identify a descent direction d_k at x_k
3. Identify a stepsize $\alpha_k > 0$
4. Define $x_{k+1} = x_k + \alpha_k d_k$
5. If x_{k+1} is good enough, stop. Otherwise set $k \leftarrow k + 1$, 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_k
- A strategy for computing a stepsize α_k
- A way to determine when an iterate has converged, terminating the algorithm

Initial guess and termination

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

Common termination strategies:

- $\|x_k - x_{k-1}\| < \epsilon$ (Does this imply x_k is close to optimal?)
- $|f(x_k) - f(x_{k-1})| < \epsilon$ (Does this imply $f(x_k)$ is close to optimal?)
- $\|\nabla f(x_k)\| < \epsilon$ (Does this imply x_k 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,

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

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: $G_k \approx \nabla^2 f(x_k)$

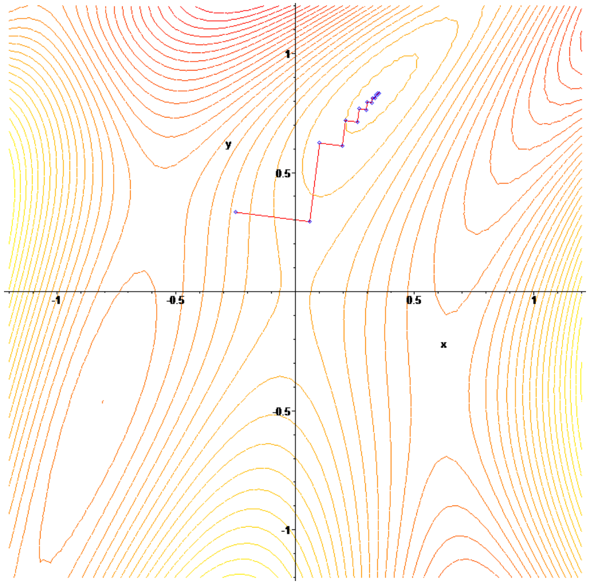
Steepest/gradient descent

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

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

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

Note that steepest descent need not be a good idea.



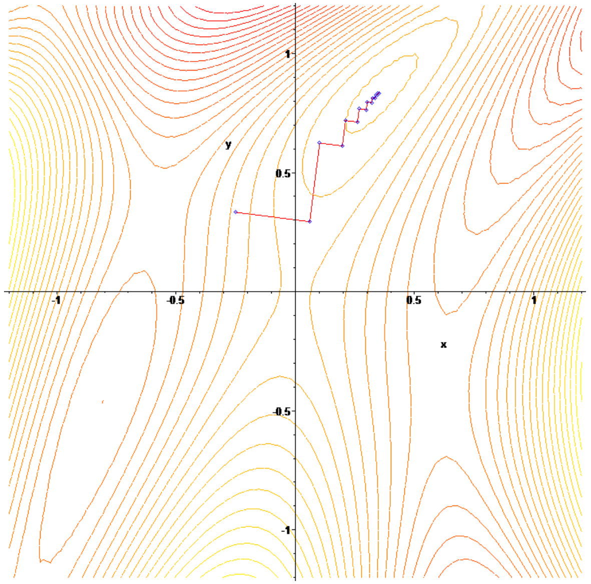
Steepest/gradient descent

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

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

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

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 $\|x - x_k\|$:

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

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:

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

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

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

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:

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

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:

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

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:

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

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

$$\nabla g(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} \nabla f(x) & \frac{\partial}{\partial x_2} \nabla f(x) & \cdots & \frac{\partial}{\partial x_n} \nabla f(x) \end{pmatrix} = \nabla^2 f(x),$$

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 <u>much</u> 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,

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

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

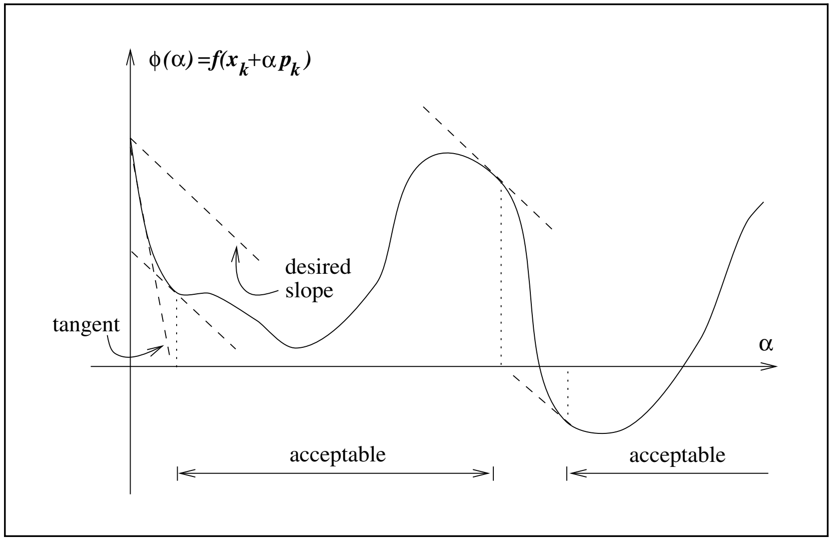


Image: Figure 3.5, Nocedal & Wright, *Numerical Optimization*

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

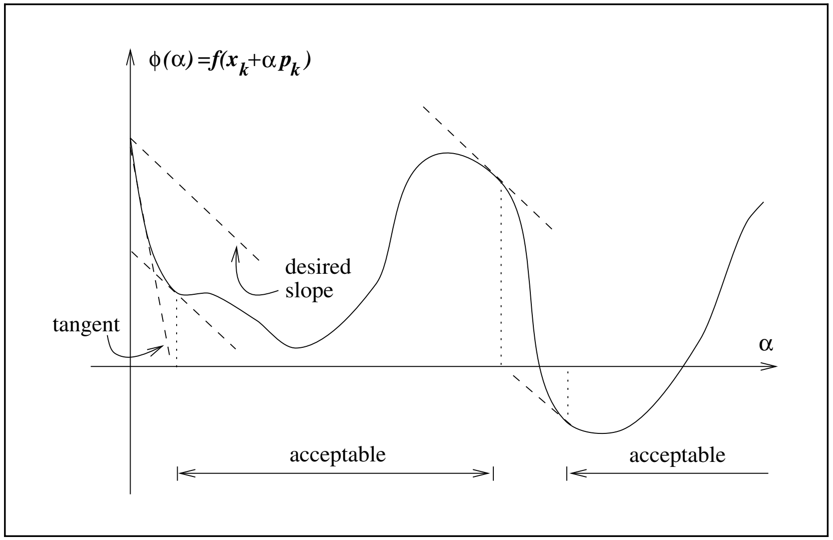


Image: Figure 3.5, Nocedal & Wright, *Numerical Optimization*

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.

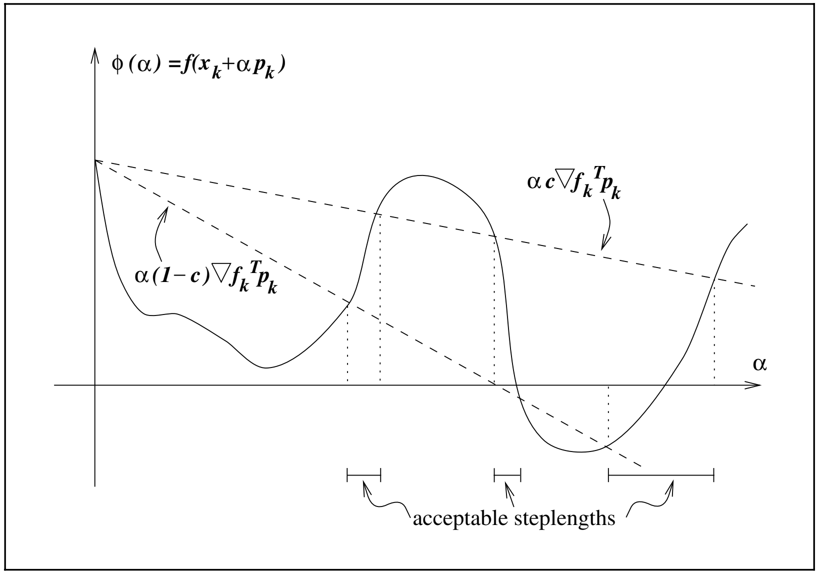
The Goldstein conditions

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

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.



$\phi(\alpha) = f(x_k + \alpha p_k)$
$\alpha c \nabla f_k^T p_k$
$\alpha (1-c) \nabla f_k^T p_k$
α
acceptable steplengths

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

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

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

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

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,

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

for some δ.

Then this implies:

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

i.e., global convergence.

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,

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

for some δ.

Then this implies:

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

i.e., global convergence.

Zoutendijk to convergence

$\kappa(A) = \frac{\sigma_{max}(A)}{\sigma_{min}(A)}$

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

What kind of descent directions satisfy this?

- Steepest descent:

$$\cos\theta_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{\|\nabla f(x_k)\|^2} = 1.$$

- Newton's method: Assume $\nabla^2 f(x_k) \succ 0$ for all k, and that

$$\kappa(\nabla^2(f(x_k))) = \frac{\lambda_n(\nabla^2 f(x_k))}{\lambda_1(\nabla^2 f(x_k))} \leqslant \kappa_0 < \infty, \qquad k \in \mathbb{N}.$$

disallow

Then

$$\cos\theta_k \geqslant \frac{1}{\kappa_0} > 0.$$

Why? $d_k = -(\nabla^2 f(x_k))^{-1} \nabla f(x_k) \longrightarrow -d_k^T \nabla f(x_k) = \nabla f(x_k)^T (\nabla^2 f)^{-1} \nabla f(x_k)$

Global convergence of descent methods

We have established that the Zoutendijk condition ensures global convergence for certain choices of descent direction. Adherence to the Zoutendijk condition is determined by an appropriate choice of step size.

Theorem

Let f be bounded below and continuously differentiable with Lipschitz continuous gradient. Assume an iteration,

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

where d_k is a descent direction and α_k. If α_k is chosen according to the Wolfe conditions, then $(\theta_k, \nabla f(x_k))_{k \geqslant 1}$ satisfies the Zoutendijk condition.

This result also holds for the strong Wolfe conditions, and for the Goldstein conditions (with some extra technical assumptions).

Convergence rates

A rather practical question of course is how quickly these methods converge.

First, some terminology: A method has a pth order convergence rate if,

$$\|x_{k+1} - x_*\|_2 \leqslant r \|x_k - x_*\|_2^p ,$$

for some constant r where x_* is the point the sequence converges to.

We have linear convergence for $p = 1$ and quadratic convergence for $p = 2$.

Note that, e.g., linear convergence is actually stronger in terms of the actual error since it implies,

$$\|x_k - x_*\|_2 \leqslant C r^k = C e^{k \log r},$$

for some constant C.

Convergence rates

A rather practical question of course is how quickly these methods converge.

First, some terminology: A method has a pth order convergence rate if,

$$\left\|x_{k+1} - x_*\right\|_2 \leqslant r \left\|x_k - x_*\right\|_2^p ,$$

for some constant r where x_* is the point the sequence converges to.

We have linear convergence for $p = 1$ and quadratic convergence for $p = 2$.

Note that, e.g., linear convergence is actually stronger in terms of the actual error since it implies,

$$\left\|x_k - x_*\right\|_2 \leqslant C r^k = C e^{k \log r},$$

for some constant C.

A simple example

Consider the quadratic function,

$$f(x) = \frac{1}{2} x^T A x - b^T x, \qquad\qquad A \succ 0,$$

The optimal solution is $x_* = A^{-1} b$.

Consider steepest descent with exact linesearch:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \qquad\qquad \alpha_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{(\nabla f(x_k))^T A \nabla f(x_k)}.$$

Theorem

The error for the above descent algorithm is given by,

$$\|x_{k+1} - x_*\| \leqslant \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right) \|x_k - x_*\|$$

A simple example

Consider the quadratic function,

$$f(x) = \frac{1}{2}x^T A x - b^T x, \qquad A \succ 0,$$

The optimal solution is $x_* = A^{-1}b$.

Consider steepest descent with exact linesearch:

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k), \qquad \alpha_k = \frac{\nabla f(x_k)^T \nabla f(x_k)}{(\nabla f(x_k))^T A \nabla f(x_k)}.$$

Theorem

The error for the above descent algorithm is given by,

$$\|x_{k+1} - x_*\| \leqslant \left(\frac{\kappa(A)-1}{\kappa(A)+1}\right)\|x_k - x_*\|$$

same as $\dfrac{\lambda_n(A) - \lambda_1(A)}{\lambda_n(A) + \lambda_1(A)} \longrightarrow$ Small (close to 0) when λ_1, λ_n are "similar"

Large (close to 1) when $\lambda_n - \lambda_1$ is large.

First-order convergence

The previous, simple example motivates the general result.

Theorem

Assume that f is continuously differentiable, and we use steepest descent with exact linesearch, which converges to x_*. If $\nabla^2 f(x_*) > 0$, then for large k we have,

$$f_{k+1} - f(x_*) \leqslant c^2 (f_k - f(x_*)),$$

where c is any number satisfying,

$$\frac{\lambda_n - \lambda_1}{\lambda_n + \lambda_1} < c < 1,$$

with λ_j the ordered eigenvalues of $\nabla^2 f(x_*)$.

Note that generally, first-order convergence is slow.

Newton's method convergence

Newton's method, being a local quadratic approximation, is second-order accurate.

Theorem

Assume $\nabla^2 f$ is Lipschitz continuous, and that the initial point x_0 is "close enough" to a local minimum x_*. Then:

$$\lim_{k \to \infty} x_k = x_*, \qquad \left\| x_{k+1} - x_k \right\| \leqslant C \left\| x_k - x_* \right\|^2 .$$

Furthermore, the gradient norm quadratically converges to 0:

$$\left\| \nabla f(x_{k+1}) \right\| \leqslant \widetilde{C} \left\| \nabla f(x_k) \right\|^2$$

The quadratic convergence guarantee of Newton's method is fantastic.

The problem is the nebulous condition on how close x_0 must be to x_*.

Qausi-Newton methods

$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ $\begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$

Quasi-Newton methods use a Newton-like update:

$$x_{k+1} = x_k + \alpha_k d_k, \qquad d_k = -G_k^{-1} \nabla f(x_k), \tag{2}$$

where $G_k \approx \nabla^2 f(x_k)$, and is easier to compute.

Note that since we only approximate the Hessian, quasi-Newton methods introduce a stepsize α_k.

Theorem

Suppose that $\nabla^2 f$ is continuously differentiable, and that we use the quasi-Newton update where α_k is chosen to satisfy the Wolfe conditions[1] Assume that $x_k \to x_*$ that is a stationary point with $\nabla^2 f(x_*) \succ 0$. If $G_k \approx \nabla^2 f(x_*)$ in the following sense,

$$\lim_{k\to\infty} \frac{\left\| \left(G_k - \nabla^2 f(x_*)\right) d_k \right\|}{\|d_k\|} = 0,$$

then (i) there is some $k_0 \in \mathbb{N}$ such that $\alpha_k = 1$ is a Wolfe condition-admissible choice for $k \geqslant k_0$, and (ii) choosing $\alpha_k = 1$ for all $k \geqslant k_0$ results in x_k converging superlinearly to x_*.

[1] A special choice of the constants is also required.

Convergence rate summary

Our three main convergence guarantees are:

- Steepest Descent: linear convergence if linesearch conditions are met.
  Pretty cheap to implement.
- Newton's method: quadratic convergence if we start close enough to a local minimum.
  Relatively expensive – computing $\nabla^2 f$ is not cheap.
- quasi-Newton methods: superlinear convergence if a "good enough" Hessian approximation is chosen.
  Cost on the order of steepest descent, typically requiring only gradients.

Many "canned" solvers use a convex combination of steepest descent and quasi-Newton methods:

1. Steepest descent is used initially to quickly reach a neighborhood of an optimum.
2. Quasi-/Newton methods then take over to bring iterates to the minimum quickly.

Convergence rate summary

Our three main convergence guarantees are:

- Steepest Descent: linear convergence if linesearch conditions are met.
  Pretty cheap to implement.

- Newton's method: quadratic convergence if we start close enough to a local minimum.
  Relatively expensive – computing $\nabla^2 f$ is not cheap.

- quasi-Newton methods: superlinear convergence if a "good enough" Hessian approximation is chosen.
  Cost on the order of steepest descent, typically requiring only gradients.

Many "canned" solvers use a convex combination of steepest descent and quasi-Newton methods:

1. Steepest descent is used initially to quickly reach a neighborhood of an optimum.
2. Quasi-/Newton methods then take over to bring iterates to the minimum quickly.

Basic quasi-Newton methods

Recall Newton's method:

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

Quasi-Newton methods (a) replace the Hessian with an approximation, and (b) introduce a stepsize to offset the potential mistake this makes

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

Why? The main issue is that computing the Hessian is expensive.

One main idea is to form a *different* quadratic approximation than that dictated by Taylor series.

Basic quasi-Newton methods

Recall Newton's method:

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

Quasi-Newton methods (a) replace the Hessian with an approximation, and (b) introduce a stepsize to offset the potential mistake this makes

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

Why? The main issue is that computing the Hessian is expensive.

One main idea is to form a *different* quadratic approximation than that dictated by Taylor series.

Local approximations

There are several quasi-Newton algorithms. The most popular ones (BFGS+DFP) use quadratic approximations:

Near x_k, the function f is approximated by,

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d,$$

where G_k is updated at every k.

Assuming $G_k > 0$, the above approximation to f has an explicitly computable minimum:

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

which is used to perform the update,

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

where α_k is chosen to satisfy the Wolfe conditions.

Local approximations

There are several quasi-Newton algorithms. The most popular ones (BFGS+DFP) use quadratic approximations:

Near x_k, the function f is approximated by,

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d,$$

where G_k is updated at every k.

Assuming $G_k \succ 0$, the above approximation to f has an explicitly computable minimum:

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

which is used to perform the update,

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

where α_k is chosen to satisfy the Wolfe conditions.

Key observations

The key advances in quasi-Newton methods utilize the following:

- G_{k+1} need not be entirely recomputed at every stage. Rather, it's chosen as a *low-rank* update to G_k.
- Inverses of low-rank augmented matrices are efficiently computable from original matrix inverses.

Theorem (Sherman-Morrison-Woodbury)

Let $A \in \mathbb{R}^{n \times n}$, and let $u in \mathbb{R}^n$ be such that $u^T A u \neq -1$. Then,

$$\left(A + uu^T\right)^{-1} = A^{-1} - \frac{(A^{-1}u)(A^{-1}u)^T}{1 + u^T A^{-1} u}.$$

This property is one key result that makes quasi-Newton methods so effective – it makes them computationally efficient.

Key observations

The key advances in quasi-Newton methods utilize the following:

- G_{k+1} need not be entirely recomputed at every stage. Rather, it's chosen as a *low-rank* update to G_k.
- Inverses of low-rank augmented matrices are efficiently computable from original matrix inverses.

Theorem (Sherman-Morrison-Woodbury)

Let $A \in \mathbb{R}^{n \times n}$, and let $\cancel{u i} n\mathbb{R}^n$ be such that $u^T A u \neq -1$. Then,

$u \in \mathbb{R}^n$

$$\left(A + uu^T\right)^{-1} = A^{-1} - \frac{(A^{-1}u)(A^{-1}u)^T}{1 + u^T A^{-1} u}.$$

This property is one key result that makes quasi-Newton methods so effective – it makes them computationally efficient.

Updating G_k

How are these low-rank updates performed?

Assume we are at location x_{k+1}, with $G_k \approx \nabla f^2(x_k)$ available from the previous step.

Our goal is to construct some Hessian approximation G_{k+1}.

Again, the main idea is that we form a local approximation around x_{k+1} for use in determining iterate $k+1$:

$$m_{k+1}(s) = f(x_{k+1}) + s^T \nabla f(x_{k+1}) + \frac{1}{2} s^T G_{k+1} s,$$

where s represents the deviation from x_{k+1}.

We impose the following (reasonable) requirements:

- The gradient of m_{k+1} at x_{k+1} matches that of f:
  $\nabla m_{k+1}(0) = \nabla f(x_{k+1})$ – this is already satisfied.
- The gradient of m_{k+1} at x_k matches that of f:
  $\nabla m_{k+1}(x_k - x_{k+1}) = \nabla f(x_k)$

Updating G_k

How are these low-rank updates performed?

Assume we are at location x_{k+1}, with $G_k \approx \nabla f^2(x_k)$ available from the previous step.

Our goal is to construct some Hessian approximation G_{k+1}.

Again, the main idea is that we form a local approximation around x_{k+1} for use in determining iterate $k+1$:

$$m_{k+1}(s) = f(x_{k+1}) + s^T \nabla f(x_{k+1}) + \frac{1}{2} s^T G_{k+1} s,$$

where s represents the deviation from x_{k+1}. $\left[s = 0 \Longleftrightarrow x = x_{k+1} \right)$

We impose the following (reasonable) requirements:

- The gradient of m_{k+1} at x_{k+1} matches that of f: $\nabla m_{k+1}(0) = \nabla f(x_{k+1})$ – this is already satisfied.
- The gradient of m_{k+1} at x_k matches that of f: $\nabla m_{k+1}(x_k - x_{k+1}) = \nabla f(x_k)$

The secant equation

We seek to impose

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

Using $x_{k+1} = x_k + \alpha_k d_k$, this results in,

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

Simplified notation for this is often used,

$$s_k := x_{k+1} - x_k, \qquad y_k := \nabla f(x_{k+1}) - \nabla f(x_k),$$

resulting in,

$$G_{k+1} s_k = y_k$$

This is called the secant equation.

The secant equation

We seek to impose

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

Using $x_{k+1} = x_k + \alpha_k d_k$, this results in,

$$G_{k+1}\left(\alpha_k d_k\right) = \nabla f(x_{k+1}) - \nabla f(x_k).$$

Simplified notation for this is often used,

$$s_k := x_{k+1} - x_k, \qquad\qquad y_k := \nabla f(x_{k+1}) - \nabla f(x_k),$$

resulting in,

$$G_{k+1} s_k = y_k$$

This is called the secant equation.

The curvature condition

The secant equation requires,

$$G_{k+1} s_k = y_k \xrightarrow{s_k^T} s_k^T G_{k+1} s_k = s_k^T y_k$$

Recall that G_{k+1} should be positive-definite – this means we require

$$s_k^T y_k > 0$$

The above is the curvature condition.

If s_k and y_k are such that the curvature condition is satisfied, it is always possible to find a (symmetric, positive-definite) solution G_{k+1}.

How to satisfy the curvature condition?

Lemma

Choosing α_k to satisfy the Wolfe conditions ensures that the curvature condition is satisfied.

The curvature condition

The secant equation requires,

$$G_{k+1} s_k = y_k$$

Recall that G_{k+1} should be positive-definite – this means we require

$$s_k^T y_k > 0$$

The above is the curvature condition.

If s_k and y_k are such that the curvature condition is satisfied, it is always possible to find a (symmetric, positive-definite) solution G_{k+1}.

How to satisfy the curvature condition?

Lemma

Choosing α_k to satisfy the Wolfe conditions ensures that the curvature condition is satisfied.

The curvature condition

The secant equation requires,

$$G_{k+1} s_k = y_k$$

Recall that G_{k+1} should be positive-definite – this means we require

$$s_k^T y_k > 0$$

The above is the curvature condition.

If s_k and y_k are such that the curvature condition is satisfied, it is always possible to find a (symmetric, positive-definite) solution G_{k+1}.

How to satisfy the curvature condition?

Lemma

Choosing α_k to satisfy the Wolfe conditions ensures that the curvature condition is satisfied.

Computing G_{k+1}

$$G_{k+1} s_k = y_k$$

Under the curvature conditions, we can find a solution G_{k+1}. How to find a unique one?

Let's find the solution "closest" to the previous iterate G_k:

$$G_{k+1} = \operatorname*{arg\,min}_{G=G^T} \|G - G_k\| \text{ subject to } G s_k = y_k$$

The choice of norm is, of course, up for grabs.

A convenient norm that makes the solution explicit is the weighted Frobenius norm,

$$\|A\|_{W,F} := \|\sqrt{W} A \sqrt{W}\|_F, \qquad W = \int_0^1 \nabla^2 f(x_k + \beta \alpha_k d_k) \mathrm{d}\beta,$$

i.e., W is an averaged Hessian.

Computing G_{k+1}

$$G_{k+1} s_k = y_k$$

Under the curvature conditions, we can find a solution G_{k+1}. How to find a unique one?

Let's find the solution "closest" to the previous iterate G_k:

$$G_{k+1} = \underset{G=G^T}{\arg\min} \|G - G_k\| \text{ subject to } G s_k = y_k$$

The choice of norm is, of course, up for grabs.

A convenient norm that makes the solution explicit is the weighted Frobenius norm,

$$\|A\|_{W,F} := \|\sqrt{W} A \sqrt{W}\|_F, \qquad W = \int_0^1 \nabla^2 f(x_k + \beta \alpha_k d_k) \mathrm{d}\beta,$$

i.e., W is an averaged Hessian.

Computing G_{k+1}

$$G_{k+1} s_k = y_k$$

Under the curvature conditions, we can find a solution G_{k+1}. How to find a unique one?

Let's find the solution "closest" to the previous iterate G_k:

$$G_{k+1} = \underset{G=G^T}{\arg\min} \|G - G_k\| \text{ subject to } G s_k = y_k$$

The choice of norm is, of course, up for grabs.

A convenient norm that makes the solution explicit is the weighted Frobenius norm,

$$\|A\|_{W,F} := \|\sqrt{W} A \sqrt{W}\|_F, \qquad W = \int_0^1 \nabla^2 f(x_k + \beta \alpha_k d_k) \mathrm{d}\beta,$$

i.e., W is an averaged Hessian.

The Davidon-Fletcher-Powell method

The solution in this weighted norm is actually explicit:

$$\begin{aligned} G_{k+1} &= (I - \rho_k y_k s_k^T) G_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T \\ &= G_k - \rho_k (y_k s_k^T G_k + G_k s_k y_k^T) + \rho_k y_k y_k^T, \end{aligned}$$

where $\rho_k = 1/(y_k^T s_k)$.

This choice of G_{k+1} (along with choosing α_{k+1} via the Wolfe conditions) is the Davidon-Fletcher-Powell (DFP) method.

For numerical efficiency, instead of updating G_k, its inverse $H_k := G_k^{-1}$ is updated.

The above is a rank-two update of G_k, implying via SMW that its inverse is also computable via a rank-two update:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \rho_k s_k s_k^T.$$

This shows the strength of quasi-Newton methods: only s_k and y_k are needed (gradients), and approximations to the Hessian H_{k+1} are quickly computable.

The Davidon-Fletcher-Powell method

The solution in this weighted norm is actually explicit:

$$\begin{aligned} G_{k+1} &= (I - \rho_k y_k s_k^T) G_k (I - \rho_k s_k y_k^T) + \rho_k y_k y_k^T \\ &= G_k - \rho_k (y_k s_k^T G_k + G_k s_k y_k^T) + \rho_k y_k y_k^T, \end{aligned}$$

where $\rho_k = 1/(y_k^T s_k)$.

This choice of G_{k+1} (along with choosing α_{k+1} via the Wolfe conditions) is the Davidon-Fletcher-Powell (DFP) method.

For numerical efficiency, instead of updating G_k, its inverse $H_k := G_k^{-1}$ is updated.

The above is a rank-two update of G_k, implying via SMW that its inverse is also computable via a rank-two update:

$$H_{k+1} = H_k - \frac{H_k y_k y_k^T H_k}{y_k^T H_k y_k} + \rho_k s_k s_k^T.$$

This shows the strength of quasi-Newton methods: only s_k and y_k are needed (gradients), and approximations to the Hessian H_{k+1} are quickly computable.

Closely related to DFP: BFGS

An approach *dual* to DFP is BFGS. Recall the secant equation:

$$G_{k+1} s_k = y_k$$

Writing $H_k := G_k^{-1}$, this is the same as

$$H_{k+1} y_k = s_k.$$

Instead of optimizing for the closest G to G_k (as DFP does), we could optimize for the closest H to H_k:

$$\underset{H_{k+1}}{\cancel{G_{k+1}}} = \underset{H=H^T}{\arg\min} \|H - H_k\| \text{ subject to } H y_k = s_k$$

Unsurprisingly, the solution is formulaically very similar to DFP:

$$\begin{aligned} H_{k+1} &= (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho s_k s_k^T \\ &= H_k - \rho_k (s_k y_k^T H_k + H_k y_k s_k^T) + \rho_k s_k s_k^T. \end{aligned}$$

The update above for H constitutes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method.

Closely related to DFP: BFGS

An approach *dual* to DFP is BFGS. Recall the secant equation:

$$G_{k+1} s_k = y_k$$

Writing $H_k := G_k^{-1}$, this is the same as

$$H_{k+1} y_k = s_k.$$

Instead of optimizing for the closest G to G_k (as DFP does), we could optimize for the closest H to H_k:

$$G_{k+1} = \underset{H=H^T}{\arg\min} \|H - H_k\| \text{ subject to } H y_k = s_k$$

Unsurprisingly, the solution is formulaically very similar to DFP:

$$\begin{aligned} H_{k+1} &= (I - \rho_k s_k y_k^T) H_k (I - \rho_k y_k s_k^T) + \rho s_k s_k^T \\ &= H_k - \rho_k (s_k y_k^T H_k + H_k y_k s_k^T) + \rho_k s_k s_k^T. \end{aligned}$$

The update above for H constitutes the Broyden-Fletcher-Goldfarb-Shanno (BFGS) method.

Quasi-Newton methods

DFP and BFGS are the most popular quasi-Newton methods.

- BFGS is typically considered "better" as it has better empirical self-correction properties (which have some theoretical underpinning)
- The initial Hessian H_0 needs to be chosen. Typically it's initialized as the exact Hessian (which requires only 1 full Hessian computation), based on some problem-specific knowledge, or is chosen as the identity.
- BFGS (with the Wolfe conditions on stepsize) is essentially the modern gold standard quasi-Newton method.

steepest descent	BFGS	Newton
1.827e-04	1.70e-03	3.48e-02
1.826e-04	1.17e-03	1.44e-02
1.824e-04	1.34e-04	1.82e-04
1.823e-04	1.01e-06	1.17e-08

Nocedal & Wright, ***Numerical Optimization***

Steepest descent is slow to converge (5264 iterations), but fast to compute.

Newton's method is fast to converge (21 iterations), but slow to compute.

Quasi-Newton methods are typically fast to converge (34 iterations) and fast to compute.

Moving on from linesearch

Linesearch methods were our starting point:

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

We choose d_k (steepest descent, quasi-/Newton), and subsequently try to choose the "best" α_k.

An alternative collection of strategies are trust region methods.

Loosely speaking, trust region methods

- choose a value for α_k (rather, a region of "trust" in a local approximation to f)
- choose a descent direction that is best inside this region
- iterates the procedure, making the trust region smaller if adequate objective decrease is not observed

Trust region methods

The basic anatomy of one iteration of a trust region method is as follows:

- "Construct" a *model* $m_k(\cdot)$ to $f(\cdot)$. This construction can use information from the current iteration (e.g., gradients) along with history.
- Decide on a trust region radius $\Delta_k > 0$ where m_k is deemed sufficiently accurate
- Solve the optimization problem:

$$x_{k+1} - x_k = \underset{\Delta_k \; d}{\arg\min}\, m_k(d) \quad \text{subject to} \quad \|d\| \leqslant \Delta_k,$$

where any norm $\|\cdot\|$ can be chosen. ($\|\cdot\| = \|\cdot\|_2$ is the most common choice.)

Trust region methods therefore solve subproblems at each step.

How to choose trust region radii?

The trust region radius Δ_k is typically chosen heuristically, based on a computable indicator:

$$d_k = x_{k+1} - x_k \cancel{x_k} \qquad \rho_k = \frac{f(x_k) - f(x_{k+1})}{m_k(0) - m_k(d_k)}.$$

The basic idea (without details): Given some initial value of Δ_k,

1. Solve the trust region subproblem for d_k
2. Compute ρ_k above
3. If ρ_k is "big enough"
   - If $\|d_k\| = \Delta_k$, make Δ_{k+1} larger than Δ_k
   - If $\|d_k\| < \Delta_k$, set $\Delta_{k+1} = \Delta_k$
4. If ρ_k is "too small":
   - Make Δ_{k+1} smaller than Δ_k
   - Set $d_k = 0$ and $x_{k+1} = x_k$ (i.e., rewind the $k+1$ step)

How to choose trust region radii?

The trust region radius Δ_k is typically chosen heuristically, based on a computable indicator:

$$d_k = x_{k+1} - x_k \qquad\qquad \rho_k = \frac{f(x_k) - f(x_{k+1})}{m_k(0) - m_k(d_k)}.$$

The basic idea (without details): Given some initial value of Δ_k,

1. Solve the trust region subproblem for d_k
2. Compute ρ_k above
3. If ρ_k is "big enough" (this is good)
   - If $\|d_k\| = \Delta_k$, make Δ_{k+1} larger than Δ_k
   - If $\|d_k\| < \Delta_k$, set $\Delta_{k+1} = \Delta_k$
4. If ρ_k is "too small": (this is bad)
   - Make Δ_{k+1} smaller than Δ_k
   - Set $d_k = 0$ and $x_{k+1} = x_k$ (i.e., rewind the $k+1$ step)

A simple specialization

The most common model m_k is a quadratic model based on a Taylor expansion:

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d,$$

where $G_k \approx \nabla^2 f(x_k)$ is typically sought.

If we use the Euclidean norm for the trust region optimization subproblem, we are solving:

$$x_{k+1} - x_k = \arg\min_d f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

Note: if $G_k > 0$, and $\|G_k^{-1} \nabla f(x_k)\| \leqslant \Delta_k$, then the solution to the above problem is exactly,

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

Most of the time, we do not achieve this unconstrained minimum, and instead must actually solve a constrained optimization problem.

The silver lining is that in practice only an approximate solution is required.

A simple specialization

The most common model m_k is a quadratic model based on a Taylor expansion:

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d,$$

where $G_k \approx \nabla^2 f(x_k)$ is typically sought.

If we use the Euclidean norm for the trust region optimization subproblem, we are solving:

$$x_{k+1} - x_k = \arg\min_d f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

Note: if $G_k \succ 0$, and $\|G_k^{-1} \nabla f(x_k)\| \leqslant \Delta_k$, then the solution to the above problem is exactly,

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

Most of the time, we do not achieve this unconstrained minimum, and instead must actually solve a constrained optimization problem.

The silver lining is that in practice only an approximate solution is required.

Trust region Hessians

If we choose a quadratic model,

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d,$$

how are the G_k approximations chosen?

Exact Hessians: Newton Trust Region methods.
Approximate Hessians: quasi-Newton Trust Region methods.

It's important to note that Newton Trust Region methods are $\underline{\text{not}}$ the same as Newton's method with a stepsize restriction!

Solutions to the subproblem

We seek to compute solutions to

$$x_{k+1} - x_k = \arg\min_{d} f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d \quad \text{subject to} \quad \|d\| \leqslant \Delta_k .$$

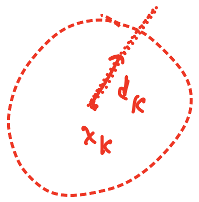
Note that this is a (fairly simple) constrained optimization problem – there are KKT conditions for its solution.

However, directly solving these is not really straightforward or cheap, so approximate methods are used.

We'll look at two of the simplest approaches:

- The Cauchy point method
- The dogleg method

The Cauchy point, I



$$x_{k+1} - x_k = \arg\min_d m_k(d) \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d$$

The simplest Cauchy point method is straightforward:

- Linearize m_k ($m_k \approx f(x_k) + d^T \nabla f(x_k)$)
- Choose d_k to minimize the linearized m_k within the trust region. Set $d_k \leftarrow d_k / \|d_k\|$.
- Minimize $m_k(\beta d_k)$ for $\beta \leqslant \Delta_k$ (linesearch)

Finding the unit norm d that minimizes the linearized m_k within the trust region is easy:

$$d_k = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

Since $m_k(\beta d_k)$ is now a quadratic function in one variable with an interval constraint on β, it is also easily minimized.

The Cauchy point, I

$$x_{k+1} - x_k = \arg\min_d m_k(d) \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d$$

The simplest Cauchy point method is straightforward:

- Linearize m_k ($m_k \approx f(x_k) + d^T \nabla f(x_k)$)
- Choose d_k to minimize the linearized m_k within the trust region. Set $d_k \leftarrow d_k / \|d_k\|$.
- Minimize $m_k(\beta d_k)$ for $\beta \leqslant \Delta_k$ (linesearch)

Finding the unit norm d that minimizes the linearized m_k within the trust region is easy:

$$d_k = -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|}$$

Since $m_k(\beta d_k)$ is now a quadratic function in one variable with an interval constraint on β, it is also easily minimized.

The Cauchy point, II

$$\begin{aligned} d_k &= -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \\ \beta_k &= \underset{\beta \in [0, \Delta_k]}{\arg\min}\, m_k(\beta d_k). \end{aligned}$$

The solution is:

$$\beta_k = \begin{cases} \Delta_k & \nabla f(x_k)^T G_k \nabla f(x_k) \leqslant 0. \\ \min\left\{ \frac{\|\nabla f(x_k)\|^3}{\nabla f(x_k)^T G_k \nabla f(x_k)}, \Delta_k \right\}, & \text{otherwise} \end{cases}$$

The final update is

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

The Cauchy point isn't very sophisticated: dependence on the Hessian is weak, playing a role only in the stepsize choice, and not in the descent direction.

The Cauchy point, II

$$\begin{aligned} d_k &= -\frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \\ \beta_k &= \underset{\beta \in [0, \Delta_k]}{\arg\min}\, m_k(\beta d_k). \end{aligned}$$

The solution is:

$$\beta_k = \begin{cases} \Delta_k & \nabla f(x_k)^T G_k \nabla f(x_k) \leqslant 0. \\ \min\left\{ \frac{\|\nabla f(x_k)\|^3}{\nabla f(x_k)^T G_k \nabla f(x_k)}, \Delta_k \right\}, & \text{otherwise} \end{cases}$$

The final update is

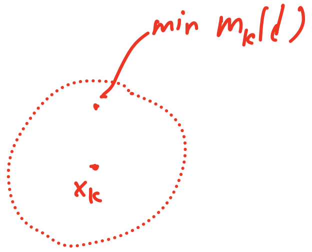
$$x_{k+1} = x_k + \beta_k d_k.$$

The Cauchy point isn't very sophisticated: dependence on the Hessian is weak, playing a role only in the stepsize choice, and not in the descent direction.

The dogleg method, I

$$x_{k+1} - x_k = \arg\min_d m_k(d) \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d$$



The dogleg method improves on the Cauchy point. We assume $G_k \succ 0$. Here is the idea:

If the global minimizer of m_k is within the trust region, we're done, so assume otherwise.

For infinitesimal Δ_k, m_k is approximately linear, so as a function of $\Delta \ll 1$, we have

$$-\Delta \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \approx \arg\min_d m_k(d),$$

which again just asserts that steepest descent is optimal for small Δ.

Along this direction, we can compute the exact unconstrained minimum of m_k:

$$\arg\min_{\beta \nabla f(x_k)} m_k(\beta \nabla f(x_k)) = -\frac{\|\nabla f(x_k)\|^2}{(\nabla f(x_k))^T G_k \nabla f(x_k)} \nabla f(x_k).$$

The dogleg method, I

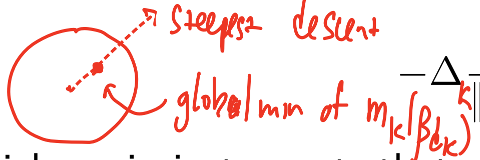
$$x_{k+1} - x_k = \arg\min_d m_k(d) \quad \text{subject to} \quad \|d\| \leqslant \Delta_k.$$

$$m_k(d) = f(x_k) + d^T \nabla f(x_k) + \frac{1}{2} d^T G_k d$$

The dogleg method improves on the Cauchy point. We assume $G_k \succ 0$. Here is the idea:

If the global minimizer of m_k is within the trust region, we're done, so assume otherwise.

For infinitesimal Δ_k, m_k is approximately linear, so as a function of $\Delta_k \ll 1$, we have



$$-\Delta_k \frac{\nabla f(x_k)}{\|\nabla f(x_k)\|} \approx \arg\min_d m_k(d),$$

which again just asserts that steepest descent is optimal for small Δ_k

Along this direction, we can compute the exact unconstrained minimum of m_k:

$$\arg\min_{\beta \nabla f(x_k)} m_k(\beta \nabla f(x_k)) = -\frac{\|\nabla f(x_k)\|^2}{(\nabla f(x_k))^T G_k \nabla f(x_k)} \nabla f(x_k).$$

The dogleg method, II

The dogleg method considers the following piecewise secant trajectory paramaterized by $\tau \in [0,2]$:

- $\tau \in [0,1]$: The line segment from the origin to the unconstrained minimum of m_k along $-\nabla f(x_k)$:

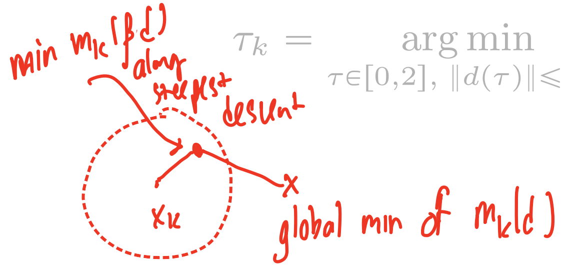
$$d(\tau) = -\tau \frac{\|\nabla f(x_k)\|^2}{(\nabla f(x_k))^T G_k \nabla f(x_k)} \nabla f(x_k), \qquad 0 \leqslant \tau \leqslant 1.$$

- $\tau \in \overset{[1,2]}{\cancel{[0,2]}}$: The line segment from $d(1)$ to the unconstrained minimum of m_k:

$$d(\tau) = d(1) + (\tau - 1)(-G_k^{-1}\nabla f(x_k) - d(1)), \qquad 1 < \tau \leqslant 2.$$

The method itself computes the minimum of m_k within the trust region along this secant trajectory:

$$\tau_k = \mathop{\arg\min}_{\tau\in[0,2],\ \|d(\tau)\|\leqslant\Delta_k} m_k(d(\tau)), \qquad x_{k+1} = x_k + d(\tau_k).$$



The dogleg method, II

The dogleg method considers the following piecewise secant trajectory paramaterized by $\tau \in [0, 2]$:

- $\tau \in [0, 1]$: The line segment from the origin to the unconstrained minimum of m_k along $-\nabla f(x_k)$:

$$d(\tau) = -\tau \frac{\|\nabla f(x_k)\|^2}{(\nabla f(x_k))^T G_k \nabla f(x_k)} \nabla f(x_k), \qquad 0 \leqslant \tau \leqslant 1.$$

- $\tau \in [0, 2]$: The line segment from $d(1)$ to the unconstrained minimum of m_k:

$$d(\tau) = d(1) + (\tau - 1)(-G_k^{-1} \nabla f(x_k) - d(1)), \qquad 1 < \tau \leqslant 2.$$

The method itself computes the minimum of m_k within the trust region along this secant trajectory:

$$\tau_k = \operatorname*{arg\,min}_{\tau \in [0,2],\ \|d(\tau)\| \leqslant \Delta_k} m_k(d(\tau)), \qquad x_{k+1} = x_k + d(\tau_k).$$

The dogleg method, II

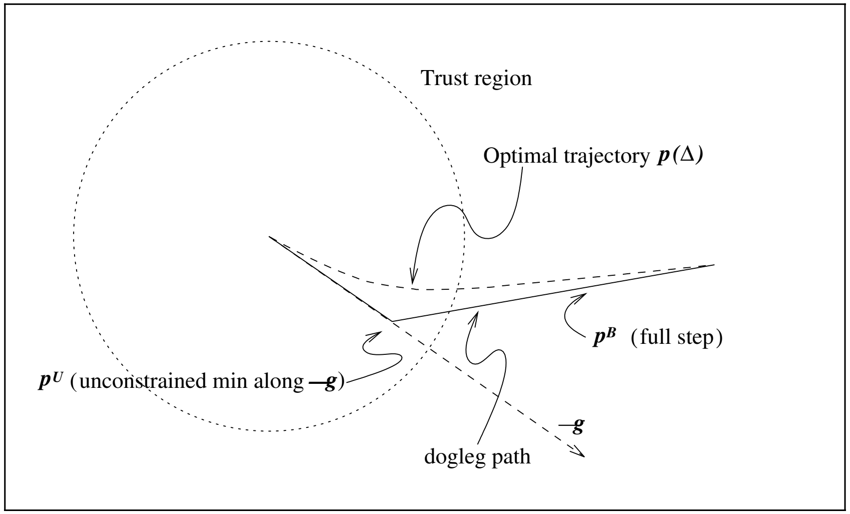

Trust region
Optimal trajectory $p(\Delta)$
p^B (full step)
p^U (unconstrained min along $-g$)
$-g$
dogleg path

Image: Figure 4.4, Nocedal & Wright, *Numerical Optimization*

One can show that m_k is decreasing along the dogleg path as τ increases. Thus, one needs only to pick τ as large as possible without leaving the trust region, which can be done explicitly.

References I

- Larry Armijo, *Minimization of functions having Lipschitz continuous first partial derivatives*, Pacific Journal of Mathematics **16** (1966), no. 1, 1–3.
- Amir Beck, *Introduction to Nonlinear Optimization*, MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics, October 2014.
- ______, *First-Order Methods in Optimization*, MOS-SIAM Series on Optimization, Society for Industrial and Applied Mathematics, October 2017.
- W. C. Davidon, *VARIABLE METRIC METHOD FOR MINIMIZATION*, Tech. Report ANL-5990, Argonne National Lab., Lemont, Ill., 1959.
- William C. Davidon, *Variable Metric Method for Minimization*, SIAM Journal on Optimization **1** (1991), no. 1, 1–17, Publisher: Society for Industrial and Applied Mathematics.
- J. E. Dennis and Robert B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Classics in Applied Mathematics, Society for Industrial and Applied Mathematics, 1996.
- Jr. Dennis, J. E. and Jorge J. Moré, *Quasi-Newton Methods, Motivation and Theory*, SIAM Review **19** (1977), no. 1, 46–89.

References II

R. Fletcher, *Practical Methods of Optimization*, Wiley, 1987, Google-Books-ID: W0zvAAAAMAAJ.

Jorge Nocedal, *Theory of algorithms for unconstrained optimization*, Acta Numerica **1** (1992), 199–242.

Jorge Nocedal and S. Wright, *Numerical Optimization*, 2 ed., Springer Series in Operations Research and Financial Engineering, Springer-Verlag, New York, 2006.

J. M. Ortega and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Classics in Applied Mathematics, 2000.

Philip Wolfe, *Convergence Conditions for Ascent Methods*, SIAM Review **11** (1969), no. 2, 226–235.

______, *Convergence Conditions for Ascent Methods. II: Some Corrections*, SIAM Review **13** (1971), no. 2, 185–188.