AMSC/CMSC 660 Scientific Computing I

Fall 2008

Unit 3: Optimization

Dianne P. O'Leary

Optimization: Fundamentals

Our goal is to develop algorithms to solve the problem

Problem P: Given a function $f : S \rightarrow \mathcal{R}$, find

$$\min_{\mathbf{x} \in S} f(\mathbf{x})$$

with solution $\mathbf{x}_{opt}$.

The point $\mathbf{x}_{opt}$ is called the minimizer, and the value $f(\mathbf{x}_{opt})$ is the minimum.

For unconstrained optimization, the set S is usually taken to be $\mathcal{R}^n$, but sometimes we make use of upper or lower bounds on the variables, restricting our search to a box

$$\{\mathbf{x} : \boldsymbol{\ell} \leq \mathbf{x} \leq \mathbf{u}\}$$

for some given vectors $\boldsymbol{\ell}, \mathbf{u} \in \mathcal{R}^n$.

The plan

1. Basics of unconstrained optimization
2. Alternatives to Newton's method
3. Fundamentals of constrained optimization

Part 1: Basics of unconstrained optimization

Plan for Part 1:

The plan:

- How do we recognize a solution?
- Some geometry.
- Our basic algorithm for finding a solution.
- The model method: Newton.
- How close to Newton do we need to be?
- Making methods safe:
  - Descent directions and line searches.
  - Trust regions.

How do we recognize a solution?

What does it mean to be a solution?

The point $\mathbf{x}_{opt}$ is a local solution to Problem P if there is a $\delta > 0$ so that if $\mathbf{x} \in S$ and $\|\mathbf{x} - \mathbf{x}_{opt}\| < \delta$, then $f(\mathbf{x}_{opt}) \leq f(\mathbf{x})$.

In other words, $\mathbf{x}_{opt}$ is at least as good as any point in its neighborhood.

The point $\mathbf{x}_{opt}$ is a global solution to Problem P if for any $\mathbf{x} \in S$, then $f(\mathbf{x}_{opt}) \leq f(\mathbf{x})$.

Note: It would be nice if every local solution was guaranteed to be global. This is true if f is convex. We'll look at this case more carefully in the "Geometry" section of these notes.

Some notation

We'll assume throughout this unit that f is smooth enough that it has as many continuous derivatives as we need. For this section, that means 2 continuous derivatives plus one more, possibly discontinuous.

The gradient of f at $\mathbf{x}$ is defined to be the vector

$$\mathbf{g}(\mathbf{x}) = \bigtriangledown f(\mathbf{x}) = \begin{bmatrix} \partial f/\partial x_1 \\ \vdots \\ \partial f/\partial x_n \end{bmatrix}.$$

The Hessian of f at $\mathbf{x}$ is the derivative of the gradient:

$$\mathbf{H}(\mathbf{x}) = \bigtriangledown^2 f(\mathbf{x}), \text{ with } h_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

Note that the Hessian is symmetric, unless f fails to be smooth enough.

How do we recognize a solution?

Recall from calculus Taylor series: Suppose we have a vector $\mathbf{p} \in \mathcal{R}^n$ with $\|\mathbf{p}\| = 1$, and a small scalar h. Then

$$f(\mathbf{x}_{opt} + h\mathbf{p}) = f(\mathbf{x}_{opt}) + h\mathbf{p}^T\mathbf{g}(x_{opt}) + \frac{1}{2}h^2\mathbf{p}^T\mathbf{H}(\mathbf{x}_{opt})\mathbf{p} + O(h^3)\,.$$

First Order Necessary Condition for Optimality

$$f(\mathbf{x}_{opt} + h\mathbf{p}) = f(\mathbf{x}_{opt}) + h\mathbf{p}^T\mathbf{g}(\mathbf{x}_{opt}) + \frac{1}{2}h^2\mathbf{p}^T\mathbf{H}(\mathbf{x}_{opt})\mathbf{p} + O(h^3)\,.$$

Now suppose that $\mathbf{g}(\mathbf{x}_{opt})$ is nonzero. Then we can always find a descent or downhill direction $\mathbf{p}$ so that

$$\mathbf{p}^T\mathbf{g}(\mathbf{x}_{opt}) < 0\,.$$

(Take, for example, $\mathbf{p} = -\mathbf{g}(\mathbf{x}_{opt})/\|\mathbf{g}(\mathbf{x}_{opt})\|$.)

Therefore, for small enough h, we can make $\frac{1}{2}h^2\mathbf{p}^T\mathbf{H}(\mathbf{x}_{opt})\mathbf{p}$ small enough that

$$f(\mathbf{x}_{opt} + h\mathbf{p}) < f(\mathbf{x}_{opt}).$$

Therefore, a necessary condition for $\mathbf{x}_{opt}$ to be a minimizer is that $\mathbf{g}(\mathbf{x}_{opt}) = \mathbf{0}$.

Second Order Necessary Condition for Optimality

So we know that if $\mathbf{x}_{opt}$ is a minimizer, then $\mathbf{g}(\mathbf{x}_{opt}) = \mathbf{0}$, so

$$f(\mathbf{x}_{opt} + h\mathbf{p}) = f(\mathbf{x}_{opt}) + \frac{1}{2}h^2\mathbf{p}^T\mathbf{H}(\mathbf{x}_{opt})\mathbf{p} + O(h^3)\,.$$

Now suppose that we had a direction $\mathbf{p}$ so that $\mathbf{p}^T\mathbf{H}(\mathbf{x}_{opt})\mathbf{p} < 0$. (We call this a direction of negative curvature.) Then again, for small enough h, we could make $f(\mathbf{x}_{opt} + h\mathbf{p}) < f(\mathbf{x}_{opt})$.

Therefore, a necessary condition for $\mathbf{x}_{opt}$ to be a minimizer is that there be no direction of negative curvature.

From linear algebra, this is equivalent to saying that the matrix $\mathbf{H}(\mathbf{x}_{opt})$ must be positive semidefinite. In other words, all of its eigenvalues must be nonnegative.

Are these conditions sufficient?

Not quite.

Example: Let f be a function of a single variable:

$$f(x) = x^3.$$

Then $f'(x) = 3x^2$ and $f''(x) = 6x$, so $f'(0) = 0$ and $f''(0) = 0$, so $x = 0$ satisfies the first- and second-order necessary conditions for optimality, but it is not a minimizer of f. []

We are very close to sufficiency, though: Recall that a symmetric matrix is positive definite if all of its eigenvalues are positive.

If $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ and $\mathbf{H}(\mathbf{x})$ is positive definite, then $\mathbf{x}$ is a local minimizer.

Some geometry

What all of this means geometrically

Imagine you are at point $\mathbf{x}$ on a mountain, described by the function $f(\mathbf{x})$, and it is foggy. (So $\mathbf{x} \in \mathcal{R}^2$.)

The direction $\mathbf{g}(\mathbf{x})$ is the direction of steepest ascent. So if you want to climb the mountain, it is the best direction to walk.

The direction $-\mathbf{g}(\mathbf{x})$ is the direction of steepest descent, the fastest way down.

Any direction $\mathbf{p}$ that makes a positive inner product with the gradient is an uphill direction, and any direction that makes a negative inner product is downhill.

If you are standing at a point where the gradient is zero, then there is no ascent direction and no descent direction, but a direction of positive curvature will lead you to a point where you can go uphill, and a direction of negative curvature will lead you to a point where you can descend.

If you can't find any of these, then you are at the bottom of a valley!

The basic algorithm

The basic algorithm

Our basic strategy is inspired by the foggy mountain:

Take an initial guess at the solution $\mathbf{x}^{(0)}$, our starting point on the mountain. Set $k = 0$.

Until $\mathbf{x}^{(k)}$ is a good enough solution,

> Find a search direction $\mathbf{p}^{(k)}$.
> Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$, where α_k is a scalar chosen to guarantee that progress is made.
> Set $k = k + 1$.

Initially, we will study algorithms for which $\alpha_k = 1$.

Unresolved details:

- testing convergence.
- finding a search direction.
- computing the step-length α_k.

The model method: Newton

Newton's method

Newton's method is one way to determine the search direction $\mathbf{p}^{(k)}$. It is inspired by our Taylor series expansion

$$f(\mathbf{x} + \mathbf{p}) \approx f(\mathbf{x}) + \mathbf{p}^T \mathbf{g}(\mathbf{x}) + \frac{1}{2} \mathbf{p}^T \mathbf{H}(\mathbf{x}) \mathbf{p} \equiv \hat{f}(\mathbf{p}).$$

Suppose we replace $f(\mathbf{x} + \mathbf{p})$ by the quadratic model $\hat{f}(\mathbf{p})$ and minimize that.

In general, the model won't fit f well at all ... except in a neighborhood of the point $\mathbf{x}$ where it is built. But if our step $\mathbf{p}$ is not too big, that is ok!

So let's try to minimize $\hat{f}$ with respect to $\mathbf{p}$. If we set the derivative equal to zero

$$\mathbf{g}(\mathbf{x}) + \mathbf{H}(\mathbf{x}) \mathbf{p} = \mathbf{0}$$

we see that we need the vector $\mathbf{p}$ defined by

$$\mathbf{H}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x})\,.$$

This vector is called the Newton direction, and it is obtained by solving the linear system involving the Hessian matrix and the negative gradient.

Picture.

Note that if the Hessian $\mathbf{H}(\mathbf{x})$ is positive definite, then this linear system is guaranteed to have a unique solution (since $\mathbf{H}(\mathbf{x})$ is nonsingular) and, in addition,

$$0 < \mathbf{p}^T\mathbf{H}(\mathbf{x})\mathbf{p} = -\mathbf{g}(\mathbf{x})^T\mathbf{p},$$

so in this case $\mathbf{p}$ is a downhill direction.

If $\mathbf{H}(\mathbf{x})$ fails to be positive definite, then the situation is not as nice.

- We may fail to have a solution to the linear system.
- We may walk uphill.

We can also get into trouble if $\mathbf{H}(\mathbf{x})$ is close to singular, since in that case it will be difficult to get a good solution to the linear system using floating point arithmetic, so the computed direction may fail to be downhill.

Bottom line:

To run the basic Newton method successfully, we need the Hessian $\mathbf{H}(\mathbf{x})$ to be positive definite everywhere we need to evaluate it.

Later, we will need to put in safeguards to handle these bad cases when $\mathbf{H}$ fails to be positive definite, but for now, we'll just study the basic Newton algorithm, in which we step from $\mathbf{x}$ to $\mathbf{x} - \mathbf{H}(\mathbf{x})^{-1}\mathbf{g}(\mathbf{x})$.

How well does the Newton Method work?

When it is good, it is very, very good!

Let $\mathbf{e}^{(k)} = \mathbf{x}^{(k)} - \mathbf{x}_{opt}$ be the error at iteration k.

Theorem: Suppose $f \in \mathcal{C}^2(S)$ and there is a positive scalar λ such that

$$\|\mathbf{H}(\mathbf{x}) - \mathbf{H}(\mathbf{y})\| \leq \lambda\|\mathbf{x} - \mathbf{y}\|$$

for all points $\mathbf{x}, \mathbf{y}$ in a neighborhood of $\mathbf{x}_{opt}$. Then if $\mathbf{x}^{(k)}$ is ffficiently close to $\mathbf{x}_{opt}$ and if $\mathbf{H}(\mathbf{x}_{opt})$ is positive definite, then there exists a constant c such that

$$\|\mathbf{e}^{(k+1)}\| \leq c\|\mathbf{e}^{(k)}\|^2 .$$

This rate of convergence is called quadratic convergence and it is remarkably fast. If we have an error of 10^{-1} at some iteration, then two iterations later the error will be about 10^{-4} (if $c \approx 1$). After four iterations it will be about 10^{-16}, as many figures as we carry in double precision arithmetic!

Newton's quadratic rate of convergence is nice, but Newton's method is not an ideal method:

- It requires the computation of $\mathbf{H}$ at each iteration.
- It requires the solution of a linear system involving $\mathbf{H}$.
- It can fail if $\mathbf{H}$ fails to be positive definite.

So we would like to modify Newton's method to make it cheaper and more widely applicable without sacrificing its fast convergence.

An important result

We can get superlinear convergence (convergence with rate $r > 1$) without walking exactly in the Newton direction.

Making the Newton method safe

When does Newton get into trouble?

We want to modify Newton whenever we are not sure that the direction it generates is downhill.

If the Hessian is positive definite, we know the direction will be downhill, although if $\mathbf{H}$ is nearly singular, we may have some computational difficulties.

If the Hessian is semidefinite or indefinite, we might or might not get a downhill direction.

Our strategy:

- We'll use the Hessian matrix whenever it is positive definite and not close to singular, because it leads to quadratic convergence.

- We'll replace $\mathbf{H}(\mathbf{x})$ by $\hat{\mathbf{H}}(\mathbf{x}) = \mathbf{H}(\mathbf{x}) + \hat{\mathbf{E}}$ whenever $\mathbf{H}$ is close to singularity or fails to be positive definite.

Conditions on $\hat{\mathbf{H}}$:

- $\hat{\mathbf{H}}$ is symmetric positive definite.
- $\hat{\mathbf{H}}$ is not too close to singular; in other words, its smallest eigenvalue is bounded below by a constant bigger than zero.

Modifying Newton's method

Sample Strategy: Levenberg-Marquardt method. This one was actually proposed for least squares problems, but it works here, too.

Replace $\mathbf{H}$ by

$$\hat{\mathbf{H}} = \mathbf{H} + \gamma \mathbf{I}\,.$$

This shifts every eigenvalue up by γ.

How do we choose γ? It is usually done by trial and error: seek a γ so that $\hat{\mathbf{H}}$ is positive definite and $\|\mathbf{p}^{(k)}\| \leq h^{(k)}$ where $\{h^{(k)}\}$ is a given sequence of numbers.

Note: If the h's are small enough, then we can avoid using a line search. (Line searches will be discussed later, but their disadvantage is that they require the function to be evaluated many times.)

Better Strategy: Cholesky Strategies: developed by Gill and Murray.

Background: Any symmetric positive definite matrix $\mathbf{A}$ can be factored as

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T$$

where $\mathbf{D}$ is a diagonal matrix and $\mathbf{L}$ is lower triangular with ones on its main diagonal.

Idea:

- While factoring, if any $d_{ii} \leq 0$, modify it so that it is positive. This changes the factored matrix from $\mathbf{H}$ to $\hat{\mathbf{H}}$.
- If modification is needed, try to keep $\|\mathbf{H} - \hat{\mathbf{H}}\|$ small so that we will have an almost-Newton direction.

- To keep close to Newton, we want $\|\mathbf{H} - \hat{\mathbf{H}}\| = 0$ if $\mathbf{H}$ is positive definite, and we want $\hat{\mathbf{H}}$ to be a continuous function of $\mathbf{H}$.
- Making $\|\mathbf{H} - \hat{\mathbf{H}}\| = 0$ if $\mathbf{H}$ is positive definite is not really possible, since we also need to modify $\mathbf{H}$ if any eigenvalue is positive but too close to zero.
- We choose to make $\hat{\mathbf{H}} = \mathbf{H} + \mathbf{E}$, where $\mathbf{E}$ is diagonal.

Two ways to modify H using Cholesky

1. $\hat{\mathbf{E}} = \gamma \mathbf{I}$ for some $\gamma \geq 0$. This is akin to Levenberg-Marquardt.
2. $\hat{\mathbf{E}}$ = a general diagonal matrix computed in the course of the Cholesky factorization.

What our algorithm now looks like

Recall: Until $\mathbf{x}^{(k)}$ is a good enough solution,

> Find a search direction $\mathbf{p}^{(k)}$.
> Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$, where α_k is a scalar chosen to guarantee that progress is made.

Now we have some details for Newton's method.
Find a search direction $\mathbf{p}^{(k)}$ means

> Calculate $\mathbf{g}^{(k)}$, $\mathbf{H}^{(k)}$.
> Factor $\mathbf{H}^{(k)} = \mathbf{L}\hat{\mathbf{D}}\mathbf{L}^T - \hat{\mathbf{E}}$.
> If $\|\mathbf{g}^{(k)}\| < \epsilon$ and $\hat{\mathbf{E}} = \mathbf{0}$, then halt with an approximate solution.
> Otherwise find a direction:
>
> > If $\|\mathbf{g}^{(k)}\| > \epsilon$, then solve $\mathbf{L}\hat{\mathbf{D}}\mathbf{L}^T\mathbf{p}^{(k)} = -\mathbf{g}^{(k)}$ to get a downhill direction.
> > Otherwise get a direction of negative curvature. (The details of this are different for each algorithm to modify L, but the cost is $O(n^2)$.)

What is missing? How long a step should we take in the direction $\mathbf{p}$?

Descent directions and line searches.

A backtracking line search

We take

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}\,.$$

How do we choose $\alpha^{(k)}$?

Let

$$F(\alpha) = f(\mathbf{x} + \alpha\mathbf{p})\,.$$

Then

$$F'(\alpha) = \mathbf{p}^T\mathbf{g}(\mathbf{x} + \alpha\mathbf{p})\,.$$

Backtracking line search:

> Choose $\alpha = 1$ (to give the full Newton step).
> While α is not good enough,
>
> > Choose a new $\alpha_{new} \in [0, \alpha]$ by interpolation, and set $\alpha = \alpha_{new}$.

Note: If **p** is not the Newton direction, then we may need an initial braketing phase to find a good upper bound on α by testing larger values.

Reference: See Section 9.3 for details.

How do we decide that α is good enough?

Our situation

- Have a downhill direction **p**, so we know that for very small α, $F(\alpha) < F(0)$.
- If **p** = the Newton direction, then we predict that $\alpha = 1$ is the minimizer.
- We want an upper bound on the αs to consider, since Newton's method is based on a quadratic model and is not expected to fit the function well if we go too far.
- We might have F' available.
- We really can't afford an exact line search. In an exact linesearch we find the value of α that exactly minimizes $f(\mathbf{x} + \alpha\mathbf{p})$. We can do this for quadratic functions, since in that case a formula for α can be derived, but in general exact linesearch is impossible and is only interesting because a lot of theorems demand it.

What do we do?

- Goldstein conditions
- Wolfe(1968)-Powell(1976) conditions

What do these conditions buy for us?

It can be shown that acceptable points exist as long as the minimizer is finite.

Typical theorem: Global convergence of descent methods. If

- f is continuously differentiable and bounded below,
- $\mathbf{g}$ is Lipschitz continuous for all $\mathbf{x}$, i.e., there exists a constant L such that, for all $\mathbf{x}$ and $\mathbf{y}$,
$$\|\mathbf{g}(\mathbf{x}) - \mathbf{g}(\mathbf{y})\| \leq L\|\mathbf{x} - \mathbf{y}\|$$

Then either $\mathbf{g}^{(k)} = \mathbf{0}$ for some k or $\mathbf{g}^{(k)} \to \mathbf{0}$.

[]

Trust regions.

Trust regions: an alternative to linesearch

Trust region methods determine α and $\mathbf{p}$ simultaneously.

Idea: Use $\mathbf{g}$ and $\mathbf{H}$ to form a quadratic model

$$f(\mathbf{x}+\mathbf{p}) \approx q(\mathbf{p}) = f(\mathbf{x}) + \mathbf{p}^T\mathbf{g} + \frac{1}{2}\mathbf{p}^T\mathbf{H}\mathbf{p}\,.$$

But we should only trust the model when $\|\mathbf{p}\| < h$ for some small scalar h.

Let $\mathbf{x}_{new} = \mathbf{x} + \mathbf{p}_{opt}$ where $\mathbf{p}_{opt}$ solves

$$\min_{\|\mathbf{p}\| \leq h} q(\mathbf{p})\,.$$

Note: Depending on the norm we choose, this gives us different geometries for the feasible set defined by $\|\mathbf{p}\| < h$.

Still to be determined:

- How to determine h and adapt it.
- How to find $\mathbf{p}_{opt}$.

How to find $\boldsymbol{p}_{opt}$

The answer changes, depending on norm we choose.

Suppose we choose the infinity norm:

$$\min_{|p_i| \leq h} q(\mathbf{p}) \,.$$

This is a quadratic programming problem with bound constraints.

How to choose h

Idea: h determines the region in which our model q is known to be a good approximation to f:

$$r \equiv \frac{f(\mathbf{x}+\mathbf{p}) - f(\mathbf{x})}{q(\mathbf{p}) - q(\mathbf{0})} \approx 1 \,.$$

Heuristic suggested by Powell:

- If r too small ($< 1/4$) then reduce h by a factor of 4.
- If r close to 1 ($> 3/4$) then increase h by a factor of 2.

Note that this can be done by modifying γ, the parameter in the Levenberg-Marquardt algorithm.

Pitfall in trust region methods: If the problem is poorly scaled, then the trust region will remain very small and we will never be able to take large steps to get us close to the solution.

Example: $f(\mathbf{x}) = f_1(x_1) + f_1(10000 x_2)$ where f_1 is a well-behaved function. []

Convergence of trust region methods

Typical theorem: Global convergence of trust region methods.
If

- $S = \{\mathbf{x} : f(\mathbf{x}) \leq f(\mathbf{x}^{(0)})\}$ is bounded.
- $f \in \mathcal{C}^2(S)$

Then the sequence $\{\mathbf{x}^{(k)}\}$ has an accumulation point $\mathbf{x}_{opt}$ that satisfies the first- and second-order necessary conditions for optimality.

Final words

We now know how to recognize a solution and compute a solution using Newton's method.

We have added safeguards in case the Hessian fails to be positive definite, and we have added a linesearch to guarantee convergence.

The resulting algorithm converges rather rapidly, but each iteration is quite expensive.

Next, we want to investigate algorithms that have lower cost per iteration.

Part 2: Alternatives to Newton's method

The plan:

Recall that Newton's method is very fast (if it converges) but it requires the gradient and the Hessian matrix to be evaluated at each iteration.

Next we develop some alternatives to Newton's method:

- Some alternatives that avoid calculation of the Hessian:
  - Quasi-Newton methods
  - finite-difference Newton methods
- Algorithms that avoid calculation of the Hessian and storage of any matrix:
  - steepest descent
  - nonlinear conjugate gradient methods
  - limited-memory Quasi-Newton methods
  - truncated Newton methods
- Technology that helps in the calculation of the Hessian: automated differentiation.
- Methods that require no derivatives
  - finite difference methods
  - Nelder&Meade simplex
  - pattern search

Methods that require only first derivatives

If the Hessian is unavailable...

Notation:

- $\mathbf{H}$ = Hessian matrix.
- $\mathbf{B}$ is its approximation.
- $\mathbf{C}$ is the approximation to $\mathbf{H}^{-1}$.

Problem: Solve

$$\min_{\mathbf{x}} f(\mathbf{x})$$

when $\mathbf{g}(\mathbf{x})$ can be computed, but not $\mathbf{H}(\mathbf{x})$.

Two options:

- Estimate $\mathbf{H}(\mathbf{x})$ using finite differences: discrete Newton. This can work well.
- Approximate $\mathbf{H}(\mathbf{x})$ using Quasi-Newton methods. (Also called variable metric.)

No Hessian Method 1: Quasi-Newton Method

Quasi-Newton methods could be the basis for a full course; there is a textbook by Dennis and Schnabel. We'll just hit the high points.

The idea behind Quasi-Newton methods

The Newton step:

$$\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$$

($\mathbf{H}$ and $\mathbf{g}$ evaluated at $\mathbf{x}^{(k)}$.)

The Quasi-Newton step: accumulate an approximation

$$\mathbf{B}^{(k)} \approx \mathbf{H}(\mathbf{x}^{(k)})$$

using free information!

What information comes free?

At step k, we know $\mathbf{g}(\mathbf{x}^{(k)})$ and we compute $\mathbf{g}(\mathbf{x}^{(k+1)})$ where $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{s}^{(k)}$.

$\mathbf{H}(\mathbf{x}^{(k)})$ satisfies

$$\mathbf{H}^{(k)}\mathbf{s}^{(k)} = \lim_{h\to 0} \frac{\mathbf{g}(\mathbf{x}^{(k)} + h\mathbf{s}^{(k)}) - \mathbf{g}(\mathbf{x}^{(k)})}{h},$$

In fact, if f is quadratic, then

$$\mathbf{H}^{(k)}\mathbf{s}^{(k)} = \mathbf{g}(\mathbf{x}^{(k)} + \mathbf{s}^{(k)}) - \mathbf{g}(\mathbf{x}^{(k)})$$

We'll ask the same property of our approximation $\mathbf{B}^{(k+1)}$ and call this the secant equation:

$$\mathbf{B}^{(k+1)}\mathbf{s}^{(k)} = \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}\,.$$

So we know how we want $\mathbf{B}^{(k+1)}$ to behave in the direction $\mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}$, and we have no new information in any other direction, so we could require

$$\mathbf{B}^{(k+1)}\mathbf{v} = \mathbf{B}^{(k)}\mathbf{v}, \text{ if } \mathbf{v}^T\mathbf{s}^{(k)} = 0\,.$$

There is a unique matrix $\mathbf{B}^{(k+1)}$ that satisfies the secant equation and the no-change conditions. It is called Broyden's good method:

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} - (\mathbf{B}^{(k)}\mathbf{s}^{(k)} - \mathbf{y}^{(k)})\frac{\mathbf{s}^{(k)T}}{\mathbf{s}^{(k)T}\mathbf{s}^{(k)}}$$

where

$$\begin{aligned}\mathbf{s}^{(k)} &= \mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\,,\\ \mathbf{y}^{(k)} &= \mathbf{g}^{(k+1)} - \mathbf{g}^{(k)}\,.\end{aligned}$$

Unquiz: Verify that Broyden's good method satisfies the secant equation and the no-change conditions. []

Notes: For Broyden's good method,

- $\mathbf{B}^{(k+1)}$ is formed from $\mathbf{B}^{(k)}$ by adding a rank-one matrix.
- $\mathbf{B}^{(k+1)}$ is not necessarily symmetric, even if $\mathbf{B}^{(k)}$ is. This is undesirable since we know $\mathbf{H}$ is symmetric.

In order to regain symmetry, we need to sacrifice the no-change conditions. Instead, we formulate the problem in a least change sense:

$$\min_{\mathbf{B}^{(k+1)}} \|\mathbf{B}^{(k+1)} - \mathbf{B}^{(k)}\|$$

subject to the secant condition

$$\mathbf{B}^{(k+1)}\mathbf{s}^{(k)} = \mathbf{y}^{(k)} .$$

The solution (again) depends on the choice of norm.

A refinement to this idea: We can impose other constraints, too.

- Perhaps we know that $\mathbf{H}$ is sparse, and we want $\mathbf{B}$ to have the same structure.
- We might want to keep $\mathbf{B}^{(k+1)}$ positive definite.

Some history

An alphabet soup of algorithms:

- DFP: Davidon 1959, Fletcher-Powell 1963
- BFGS: Broyden, Fletcher, Goldfarb, Shanno 1970
- Broyden's good method and Broyden's bad
- ...

An example: DFP

Still one of the most popular because it has many desirable properties.

We accumulate an approximation $\mathbf{C}$ to $\mathbf{H}^{-1}$ rather than to $\mathbf{H}$.

$$\mathbf{C}^{(k+1)} = \mathbf{C}^{(k)} - \frac{\mathbf{C}^{(k)}\mathbf{y}^{(k)}\mathbf{y}^{(k)T}\mathbf{C}^{(k)}}{\mathbf{y}^{(k)T}\mathbf{C}^{(k)}\mathbf{y}^{(k)}} + \frac{\mathbf{s}^{(k)}\mathbf{s}^{(k)T}}{\mathbf{y}^{(k)T}\mathbf{s}^{(k)}}$$

An example: BFGS

The most popular method.

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} - \frac{\mathbf{B}^{(k)}\mathbf{s}^{(k)}\mathbf{s}^{(k)T}\mathbf{B}^{(k)}}{\mathbf{s}^{(k)T}\mathbf{B}^{(k)}\mathbf{s}^{(k)}} + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)T}}{\mathbf{y}^{(k)T}\mathbf{s}^{(k)}}$$

Use of Quasi-Newton methods

The algorithm looks very similar to Newton's method:

Until $\mathbf{x}^{(k)}$ is a good enough solution,

> Compute a search direction $\mathbf{p}^{(k)}$ from $\mathbf{p}^{(k)} = -\mathbf{C}^{(k)}\mathbf{g}^{(k)}$ (or solve $\mathbf{B}^{(k)}\mathbf{p}^{(k)} = -\mathbf{g}^{(k)}$).
> Set $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$, where α_k satisfies the Goldstein-Armijo or Wolfe linesearch conditions.
> Form the updated matrix $\mathbf{C}^{(k+1)}$ (or $\mathbf{B}^{(k+1)}$).
> Increment k.

Initialization: Now need to initialize $\mathbf{B}^{(0)}$ (or $\mathbf{C}^{(0)}$) as well as $\mathbf{x}^{(0)}$. Take $\mathbf{B}^{(0)} = \mathbf{I}$ or some better guess.

Unanswered questions

- Near a stationary point, $\mathbf{H}^{-1}$ does not exist. How do we keep $\mathbf{C}$ from deteriorating?
- What happens if $\mathbf{H}$ is indefinite?
- How do we check optimality?

The first two questions concern stability of the algorithm.

Answering concerns about stability

Our dilemma:

- Updating $\mathbf{C}$ can be hazardous when $\mathbf{H}$ is close to singular.
- Updating $\mathbf{B}$ leaves the problem of solving a linear system at each iteration to determine the search direction.

An alternative: Update a factorization of $\mathbf{B}$. This makes it easy to enforce symmetry and positive definiteness.

Example: BFGS This algorithm updates the approximate Hessian $\mathbf{B}$. There are two economical alternatives for solving the necessary linear systems

1. Updating the inverse.

$$\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)} - \frac{\mathbf{B}^{(k)}\mathbf{s}^{(k)}\mathbf{s}^{(k)T}\mathbf{B}^{(k)}}{\mathbf{s}^{(k)T}\mathbf{B}^{(k)}\mathbf{s}^{(k)}} + \frac{\mathbf{y}^{(k)}\mathbf{y}^{(k)T}}{\mathbf{y}^{(k)T}\mathbf{s}^{(k)}}$$

   so, by the Sherman-Morrison-Woodbury formula for computing inverses of matrices updated by a rank-2 correction, we can obtain

$$\mathbf{B}^{(k+1)-1} = \mathbf{B}^{(k)-1} + \frac{\mathbf{y}^{(k)T}(\mathbf{B}^{(k)-1}\mathbf{y}^{(k)} + s^{(k)})}{(\mathbf{y}^{(k)T}\mathbf{s}^{(k)})^2}\mathbf{s}^{(k)}\mathbf{s}^{(k)T}$$

$$-\frac{\mathbf{s}^{(k)}\mathbf{y}^{(k)T}\mathbf{B}^{(k)-1} + \mathbf{B}^{(k)-1}\mathbf{y}^{(k)}\mathbf{s}^{(k)T}}{(\mathbf{y}^{(k)T}\mathbf{s}^{(k)})} .$$

2. Updating a factorization. If we have a Cholesky factorization of $\mathbf{B}^{(k)}$ as $\mathbf{B}^{(k)} = \mathbf{L}^{(k)}\mathbf{D}^{(k)}\mathbf{L}^{(k)T}$, then we want $\mathbf{B}^{(k+1)} = \mathbf{L}^{(k+1)}\mathbf{D}^{(k+1)}\mathbf{L}^{(k+1)^T}$. This can be formed by formulas analogous to the Sherman-Morrison-Woodbury formula, and we can use the ideas in the previous set of notes to modify it to preserve positive definiteness. The algorithms are $O(n^2)$. Details are given, for example, in a paper by Gill, Golub, Murray, and Saunders, *Math. Comp.* 28 (1974) pp.505-535, and in some textbooks.

Answering concerns about convergence

It is an unfortunate fact that software packages do not check the second-order optimality conditions. They just return a point at which the gradient is approximately zero and they can find no descent direction or direction of negative curvature.

Convergence rate

All of these methods have an n-, $2n$-, or $(n+2)$-step quadratic convergence rate if the line search is exact. An n-step quadratic convergence rate, for example, means that

$$\lim_{k\to\infty} \frac{\|\mathbf{x}^{(k+n)} - \mathbf{x}_{opt}\|}{\|\mathbf{x}^{(k)} - \mathbf{x}_{opt}\|^2} < \infty .$$

Weakening the line search to a Wolfe or Goldstein-Armijo search generally gives superlinear convergence.

No Hessian Method 2: Finite-difference Newton Method

Finite-difference Newton methods

One of our major time-sinks in using Newton's method is to evaluate the Hessian matrix, with entries

$$h_{ij} = \frac{\partial g_i}{\partial x_j} .$$

One way to avoid these evaluations is to approximate the entries:

$$h_{ij} \approx \frac{g_i(\mathbf{x} + \tau \mathbf{e}_j) - g_i(\mathbf{x})}{\tau}$$

where τ is a small number and $\mathbf{e}_j$ is the jth unit vector.

Cost: n extra gradient evaluations per iteration. Sometimes this is less than the cost of the Hessian evaluation, but sometimes it is more.

Practical matters:

- How to choose τ.
  - If τ is large, the approximation is poor and we have large truncation error.
  - If τ is small, then there is cancellation error in forming the numerator of the approximation, so we have large round-off error.

  Usually we try to balance the two errors by choosing τ to make them approximately equal.
- If the problem is poorly scaled, we may need a different τ for each j.

Convergence rate

There are theorems that say that if we choose τ carefully enough, we can get superlinear convergence.

Methods that require only first derivatives and store no matrices

Sometimes problems are too big to allow n^2 storage space for the Hessian matrix.

Some alternatives:

- steepest descent
- nonlinear conjugate gradient
- limited memory Quasi-Newton
- truncated Newton

Low-storage Method 1: Steepest Descent

Back to that foggy mountain

If we walk in the direction of steepest descent until we stop going downhill, we clearly are guaranteed to get to a local minimizer.

The trouble is that the algorithm is terribly slow.

If we apply steepest descent to a quadratic function of n variables, then after many steps, the algorithm takes alternate steps approximating two directions: those corresponding to the eigenvectors of the smallest and the largest eigenvalues of the Hessian matrix.

The convergence rate on quadratics is linear:

$$f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}_{opt}) \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^2 (f(\mathbf{x}^{(k)}) - f(\mathbf{x}_{opt}))$$

where κ is the ratio of the largest to the smallest eigenvalue of $\mathbf{H}$.

If steepest descent is applied to non-quadratic functions, using a good line search, then convergence is local and linear.

Advantages of steepest descent:

- No need to evaluate the 2nd derivative or to solve a linear system.
- Low storage: no matrices.

Disadvantages of steepest descent:

- Very slow.
- Very, very slow.

Use nonlinear conjugate gradients instead. Same advantages but better convergence.

Linear conjugate gradients

Reference: Section 28.2

The conjugate gradient method (Hestenes&Stiefel, 1952) is a method for solving linear systems of equations $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is symmetric and positive definite.

There are many ways to understand it, but for us, we think of it as minimizing the function

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{b}$$

which has gradient $\mathbf{g}(\mathbf{x}) = \mathbf{Ax} - \mathbf{b}$. So a minimizer of f is a solution to our linear system.

We could use steepest descent, but we want something faster.

See the notes on the linear algorithm to understand how conjugate gradient combines the concepts of descent and conjugate directions.

Summary of the linear conjugate gradient algorithm

Given $\mathbf{x}^{(0)}$, form $-\mathbf{g}(\mathbf{x}^{(0)}) = \mathbf{b} - \mathbf{Ax}^{(0)} = \mathbf{p}^{(0)}$.

For $k = 0, 1, \ldots$, until convergence,

> Use a line search to determine $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$.
> Set $\mathbf{p}^{(k+1)} = -\mathbf{g}(\mathbf{x}^{(k+1)}) + \beta^{(k+1)}\mathbf{p}^{(k)}$.

Linear conjugate gradients in more detail

Purpose: To solve $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is symmetric and positive definite. The only access to $\mathbf{A}$ is through a function call that returns the product of $\mathbf{A}$ with any given vector.

Given $\mathbf{x}^{(0)}$, form $\mathbf{r}^{(0)} = \mathbf{b} - \mathbf{Ax}^{(0)}$ and $\mathbf{s}^{(0)} = -\mathbf{g}(\mathbf{x}^{(0)})$ (Note that $\mathbf{r}$ is the negative gradient of $f(\mathbf{x}) = 1/2\,\mathbf{x}^T\mathbf{Ax} - \mathbf{x}^T\mathbf{b}$, and I am using $\mathbf{s}$ for the search direction to avoid confusion later). For $k = 0, 1, \ldots$, until convergence,

> Let $\mathbf{z}^{(k)} = \mathbf{As}^{(k)}$.
> Let the step length be $\alpha^{(k)} = (\mathbf{r}^{(k)T}\mathbf{s}^{(k)})/(\mathbf{s}^{(k)T}\mathbf{z}^{(k)})$.
> Let $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{s}^{(k)}$.
> Update the negative gradient $\mathbf{r}^{(k+1)} = \mathbf{r}^{(k)} - \alpha^{(k)}\mathbf{z}^{(k)}$.
> Let $\beta^{(k+1)} = (\mathbf{r}^{(k+1)^T}\mathbf{r}^{(k+1)})/(\mathbf{r}^{(k)T}\mathbf{r}^{(k)})$.
> Let the new search direction be $\mathbf{s}^{(k+1)} = \mathbf{r}^{(k+1)} + \beta^{(k+1)}\mathbf{s}^{(k)}$.

The nonlinear conjugate gradient algorithm

Given $\mathbf{x}^{(0)}$, form $\mathbf{p}^{(0)} = -\mathbf{g}(\mathbf{x}^{(0)})$.

For $k = 0, 1, \ldots$, until convergence,

Use a line search to determine $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\mathbf{p}^{(k)}$.
Set $\mathbf{p}^{(k+1)} = -\mathbf{g}(\mathbf{x}^{(k+1)}) + \beta^{(k+1)}\mathbf{p}^{(k)}$.

The parameter α is determined by a line search.

The parameter β has many definitions that are equivalent for the linear problem but different when we minimize nonlinear functions:

$$\beta^{(k+1)} = \frac{\mathbf{g}^{(k+1)T}\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)T}\mathbf{g}^{(k)}} \quad \text{Fletcher-Reeves}$$

$$\beta^{(k+1)} = \frac{\mathbf{y}^{(k)T}\mathbf{g}^{(k+1)}}{\mathbf{g}^{(k)T}\mathbf{g}^{(k)}} \quad \text{Polak-Ribiére}$$

$$\beta^{(k+1)} = \frac{\mathbf{y}^{(k)T}\mathbf{g}^{(k+1)}}{\mathbf{y}^{(k)T}\mathbf{p}^{(k)}} \quad \text{Hestenes-Stiefel}$$

Good theorems have been proven about convergence of Fletcher- Reeves, but Polak-Ribiére is generally better performing.

Note that this method stores no matrix. We only need to remember a few vectors at a time, so it can be used for problems in which there are thousands or millions of variables.

The convergence rate is linear, unless the function has special properties, but generally faster than steepest descent: for quadratics, the rate is

$$f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}_{opt}) \leq \left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^2 (f(\mathbf{x}^{(k)}) - f(\mathbf{x}_{opt}))$$

where κ is the ratio of the largest to the smallest eigenvalue of $\mathbf{H}$.

An important property of conjugate gradients:

If we run cg on a quadratic function, then it generates the same iterates as the Huang family of Quasi-Newton algorithms.

Low-storage Method 3: Limited-memory Quasi-Newton methods

Consider as an example the DFP formula

$$\mathbf{C}^{(k+1)} = \mathbf{C}^{(k)} - \frac{\mathbf{C}^{(k)}\mathbf{y}^{(k)}\mathbf{y}^{(k)T}\mathbf{C}^{(k)}}{\mathbf{y}^{(k)T}\mathbf{C}^{(k)}\mathbf{y}^{(k)}} + \frac{\mathbf{s}^{(k)}\mathbf{s}^{(k)T}}{\mathbf{y}^{(k)T}\mathbf{s}^{(k)}}$$

We've thought of this as a matrix stored in memory. Let's develop a different view.

How do we use this matrix? All we need to do is to multiply vectors, such as the gradient $\mathbf{g}^{(k)}$, by it.

Suppose we know $\mathbf{C}^{(0)}$, which is perhaps a diagonal matrix that takes only n storage locations, so we can easily multiply any vector by $\mathbf{C}^{(0)}$.

Also suppose that we have stored the vectors $\mathbf{s}^{(j)}$ and $\mathbf{y}^{(j)}$, $j = 0, \ldots, k-1$.

Consider these formulas:

$$\mathbf{C}^{(1)}\mathbf{z} = \mathbf{C}^{(0)}\mathbf{z} - \frac{\mathbf{y}^{(0)T}\mathbf{C}^{(0)}\mathbf{z}}{\mathbf{y}^{(0)T}\mathbf{C}^{(0)}\mathbf{y}^{(0)}}\mathbf{C}^{(0)}\mathbf{y}^{(0)} + \frac{\mathbf{s}^{(0)T}\mathbf{z}}{\mathbf{y}^{(0)T}\mathbf{s}^{(0)}}\mathbf{s}^{(0)} .$$

$$\mathbf{C}^{(2)}\mathbf{z} = \mathbf{C}^{(1)}\mathbf{z} - \frac{\mathbf{y}^{(1)T}\mathbf{C}^{(1)}\mathbf{z}}{\mathbf{y}^{(1)T}\mathbf{C}^{(1)}\mathbf{y}^{(1)}}\mathbf{C}^{(1)}\mathbf{y}^{(1)} + \frac{\mathbf{s}^{(1)T}\mathbf{z}}{\mathbf{y}^{(1)T}\mathbf{s}^{(1)}}\mathbf{s}^{(1)} .$$

To compute our search direction $\mathbf{C}^{(k)}\mathbf{g}$ we start by forming $\mathbf{C}^{(0)}\mathbf{g}$ and $\mathbf{C}^{(0)}\mathbf{y}^{(j)}$, $j = 0, \ldots, k-1$.

Next, we form $\mathbf{C}^{(1)}\mathbf{g}$ and $\mathbf{C}^{(1)}\mathbf{y}^{(j)}$, $j = 1, \ldots, k-1$, using the multiplication formula above (in red) applied to each vector.

Next, we form $\mathbf{C}^{(2)}\mathbf{g}$ and $\mathbf{C}^{(2)}\mathbf{y}^{(j)}$, $j = 2, \ldots, k-1$, using the multiplication formula above (in plum) applied to each vector.

We continue for k steps.

What have we accomplished? Recall that $\mathbf{C}^{(0)}$ is usually chosen to be simple, like a diagonal matrix. Therefore, instead of taking n^2 storage locations for $\mathbf{C}^{(2)}$, we only need $4n + 4$!

The idea behind limited memory quasi-Newton algorithms is to continue this process ℓ steps, until we don't want to store any more vectors. Then, we start

storing the new vectors in place of the oldest ones that we remember, always keeping the ℓ most recent updates.

For various more refined strategies, see reference 4 at the end of these notes.

Low-storage Method 4: Truncated Newton methods

Again we return to the way the Hessian approximation is used.

If we have the Hessian matrix, how do we use it? Newton's method determines the search direction by solving the linear system

$$\mathbf{Hp} = -\mathbf{g}$$

We usually think of solving this by factoring $\mathbf{H}$ and then using forward- and back- substitution.

But if $\mathbf{H}$ is large, this might be too expensive, and we might choose to use an iterative method, like linear conjugate gradients, to solve the system.

If we do, How do we use the Hessian? All we need to do is to multiply a vector by it at each step of the algorithm.

Now Taylor series tells us that, if $\mathbf{v}$ is a vector of length 1, then

$$\mathbf{g}(\mathbf{x} + h\mathbf{v}) = \mathbf{g}(\mathbf{x}) + h\mathbf{H}(\mathbf{x})\mathbf{v} + O(h^2)\,,$$

so

$$\mathbf{H}(\mathbf{x})\mathbf{v} = \frac{\mathbf{g}(\mathbf{x} + h\mathbf{v}) - \mathbf{g}(\mathbf{x})}{h} + O(h)\,.$$

Therefore, we can get an $O(h)$ approximation of the product of the Hessian with an arbitrary vector by taking a finite difference approximation to the change in the gradient in that direction.

This is akin to the finite-difference Newton method, but much neater, because we only evaluate the finite difference in directions in which we need it.

The Truncated Newton strategy

So we'll compute an approximation to the Newton direction $\mathbf{p} = -\mathbf{H}^{-1}\mathbf{g}$ by solving the linear system $\mathbf{Hp} = -\mathbf{g}$ using the conjugate gradient method, computing approximate matrix-vector products by extra evaluations of the gradient.

We hope to obtain a superlinear convergence rate, so we need, by the theorem of Sec. 9.2.1, that

$$\frac{\|\text{our direction} - \text{Newton direction}\|}{\|\text{our direction}\|} \to 0$$

as the iteration number $\to \infty$.

We ensure this by

- taking enough iterations of conjugate gradient to get a small residual to the linear system.
- choosing h in the approximation carefully, so that the matrix- vector products are accurate enough.

Automated differentiation

Automated differentiation

The most tedious and error-prone part of nonlinear optimization: writing code for derivatives.

An alternative: let the computer do it.

Automatic differentiation is an old idea:

- The forward (bottom-up) algorithm was proposed in the 1970s.
- The backward (top-down) algorithm was proposed in the 1980s.
- Reliable software (`Adifor`, etc., by Bischof, Griewank, ...) was developed in the 1990s.

No-derivative methods

These are methods of last resort, generally used when

- derivatives are not available.
- derivatives do not exist.

This is an area of very active research currently. We'll consider three classes of methods.

No-derivative Method 1: Finite difference methods

We could do finite differences on the function to get an approximate gradient, but this is not usually a good idea, given that automatic differentiation methods exist.

No-derivative Method 2: Simplex-based methods

(Not to be confused with the simplex method for linear programming.)

The most popular of these is the Nelder-Meade algorithm, and Matlab has an implementation of this.

Idea:

- Suppose we have evaluated the function at the vertices of a simplex. (In 2-dimensions, this is a triangle, in 3, it is a tetrahedron, etc.)
- We would like to move one vertex of this simplex, reflecting it around its current position, until we have enclosed the minimizer in the simplex.
- Then we would like to shrink the size of the simplex to hone in on the minimizer.

Simplex-based algorithms have rather elaborate rules for determining when to reflect and when to shrink, and no algorithm that behaves well in practice has a good convergence proof.

For that reason, it looks as if they will fade in popularity, being supplanted by pattern search methods.

No-derivative Method 3: Pattern search methods

Idea:

- Suppose we are given an initial guess x at the solution and a set of at least $n+1$ directions v_i, $i = 1, \ldots, N$, that form a positive basis for $\mathcal{R}^n$: this means that any vector can be expressed as a linear combination of these vectors, where the coefficients in the combination are positive numbers.
- At each step, we do a line search in each of the directions to obtain $f(\mathbf{x} + \alpha_i \mathbf{v}_i)$ and replace $\mathbf{x}$ by the point with the smallest function value.

This is a remarkably simple algorithm, but works well in practice and is provably convergent!

Another desirable property is that it is easy to parallelize, and this is crucial to making a no-derivative algorithm effective when n is large.

Example: Tamara Kolda software. []

Reference: Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon, "Optimization by Direct Search: New Perspectives on Some Classical and Modern Methods," *SIAM Review* 45 (2003) 385-482.

Final words

- We have discussed a variety of algorithms for solving optimization problems.
- For least squares problems, use a specialized algorithm, as discussed in Chapter 13.
- Another class of algorithms (randomized algorithms) is discussed in Chapter 17.
- To choose among algorithms, see Section 9.6.

Some personal references

1. Dianne P. O'Leary, "Conjugate gradient algorithms in the solution of optimization problems for nonlinear elliptic partial differential equations," *Computing* 22 (1979) 59-77.
2. Dianne P. O'Leary, "A discrete Newton algorithm for minimizing a function of many variables," *Mathematical Programming* 23 (1982) 20-33. This algorithm came to be known as "truncated Newton".
3. Dianne P. O'Leary, "Why Broyden's nonsymmetric method terminates on linear equations," *SIAM Journal on Optimization*, 5 (1995) 231-235.
4. Tamara Kolda, Dianne P. O'Leary, and Larry Nazareth, "BFGS with Update Skipping and Varying Memory," *SIAM Journal on Optimization,* 8 (1998) 1060-1083. `http://epubs.siam.org/sam-bin/dbq/article/30645`
5. Daniel M. Dunlavy, Dianne P. O'Leary, Dmitri Klimov, and D. Thirumalai, "HOPE: A Homotopy Optimization Method for Protein Structure Prediction," *Journal of Computational Biology*, 12, No. 10 (2005), pp. 1275-1288.
6. Haw-ren Fang and Dianne P. O'Leary, "Modified Cholesky Algorithms: A Catalog with New Approaches," Computer Science Department Report CS-TR-4807, Institute for Advanced Computer Studies Report UMIACS-2006-??, University of Maryland, August 2006.

Part 3: Fundamentals for constrained optimization

Our approach: Always try to reduce the problem to one with a known solution.

Reference: Chapter 10

Our problem

$$\min_{\mathbf{x}} f(\mathbf{x})$$

$$\begin{aligned} c_i(\mathbf{x}) &= 0\,, \quad i \in \mathcal{E} \\ c_i(\mathbf{x}) &\geq 0\,, \quad i \in \mathcal{I} \end{aligned}$$

where f and c_i are $\mathcal{C}^2$ functions from $\mathcal{R}^n$ into $\mathcal{R}^1$.

We say that $\mathbf{x}_{opt}$ is a solution to our problem if

- $\mathbf{x}_{opt}$ satisfies all of the constraints.
- For some $\epsilon > 0$, if $\|\mathbf{y} - \mathbf{x}_{opt}\| \leq \epsilon$, and if $\mathbf{y}$ satisfies the constraints, then $f(\mathbf{y}) \geq f(\mathbf{x}_{opt})$.

In other words, $\mathbf{x}_{opt}$ is feasible and locally optimal.

Catalog of problems and algorithms

- Case 1: Linear constraints and linear f.

  This is a linear programming problem. There are two popular types of algorithm:

  - Simplex method for linear programming.
    This was the most popular up to the 1990s.
  - Interior point methods (IPMs).
    These are generally faster on large problems and remain an active area of research.

  Both of these methods are implemented in Matlab's `linprog`.

- Case 2: General .
  Note: Unless both f and the feasible region are convex, there is no guarantee of finding the global solution. You may obtain a local solution that is far from the best.

  Some algorithmic approaches:

- Idea 1: Reduced variable methods eliminate constraints by reducing the number of variables.
- Idea 2: Barrier methods and interior point methods eliminate constraints through Lagrangians.

An example of these approaches

Let

$$\begin{aligned} f(\mathbf{x}) &= x_1^2 - 2x_1x_2 + 4x_2^2 \\ c_1(\mathbf{x}) &= x_1 + x_2 - 1 = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 1 \end{aligned}$$

We'll consider two approaches to the problem.

Approach 1: The feasible direction formulation

Suppose the constraint is an equality constraint.

If $x_1 + x_2 = 1$, then all feasible points have the form

$$\mathbf{x} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ -1 \end{bmatrix}.$$

This formulation works because

$$\mathbf{A}\mathbf{x} = \begin{bmatrix} 1 & 1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \alpha \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 1$$

and all vectors $\mathbf{x}$ that satisfy the constraints have this form.

We obtain this formulation for feasible $\mathbf{x}$ by taking a particular solution

$$\begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

and adding on a linear combination of vectors that span the null space of the matrix

$$\begin{bmatrix} 1 & 1 \end{bmatrix}.$$

The null space defines the set of feasible directions, the directions in which we can step without immediately stepping outside the feasible space.

Approach 2: A barrier formulation

Suppose the constraint is an inequality constraint.

Let

$$\begin{aligned} B_\mu(\mathbf{x}) &= f(\mathbf{x}) - \mu \log c_1(\mathbf{x}) \\ &= x_1^2 - 2x_1x_2 + 4x_2^2 - \mu \log(x_1 + x_2 - 1)\,. \end{aligned}$$

If we minimize $B_\mu(\mathbf{x})$, starting from a point at which $x_1 + x_2 - 1 > 0$, and for a sufficiently small value of μ, then we will approximately solve our original problem.

End example []

The feasible direction approach

In general, if our constraints are $\mathbf{Ax} = \mathbf{b}$, to get feasible directions, we express $\mathbf{x}$ as

$$\mathbf{x} = \bar{\mathbf{x}} + \mathbf{Zv}$$

where

- $\bar{\mathbf{x}}$ is a particular solution to the equations $\mathbf{Ax} = \mathbf{b}$ (any one will do),
- the columns of $\mathbf{Z}$ form a basis for the nullspace of $\mathbf{A}$ (any basis will do),
- $\mathbf{v}$ is an arbitrary vector of dimension $(n - m) \times 1$.

Then we have succeeded in reformulating our constrained problem as an unconstrained one with a smaller number of variables:

$$\min_{\mathbf{v}} f(\bar{\mathbf{x}} + \mathbf{Zv})$$

Barrier methods

The problem:

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &\mathbf{c}(\mathbf{x}) \geq \mathbf{0} \end{aligned}$$

The Lagrangian:

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) - \boldsymbol{\lambda}^T \mathbf{c}(\mathbf{x})$$

Log Barrier function:

$$B_\mu(\mathbf{x}) = f(\mathbf{x}) - \mu \sum_{i=1}^{m} \log(c_i(\mathbf{x}))$$

The severity of the barrier is controlled by the choice of μ:

- large μ: gradual barrier
- small μ: sharp barrier

Picture

The tradeoff

- If μ is large and the minimizer is near the boundary:
  - the barrier function looks very different from f.
- If μ is small and the minimizer is near the boundary:
  - steep gradients.
  - ill-conditioning and therefore hard to solve the problem.

Because of this, we usually solve a sequence of problems:

- Start with a large μ to guide us near the solution.
- Gradually reduce μ, using our previous solution as a starting point.

Important alert

This is an important computational paradigm: we replace one hard problem (constrained minimization) by a sequence of easier problems, each of which gives a hint at the solution to the next.

We will see this again in homotopy methods for solving systems of nonlinear equations.

Final words

- We have just touched the surface of this subject. AMSC 607 / CMSC 764 expands on it.
- Barrier methods inspire a more systematic approach to the modification of μ, as in interior point methods.

References

Background: van Loan, Chapter 8.

Stephen G. Nash and Ariela Sofer, *Linear and Nonlinear Programming* by McGraw-Hill, 1996.