Introduction to non-linear optimization

Ross A. Lippert

D. E. Shaw Research

February 25, 2008

Optimization problems

problem: Let $f : \mathbb{R}^n \to (-\infty, \infty]$,

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

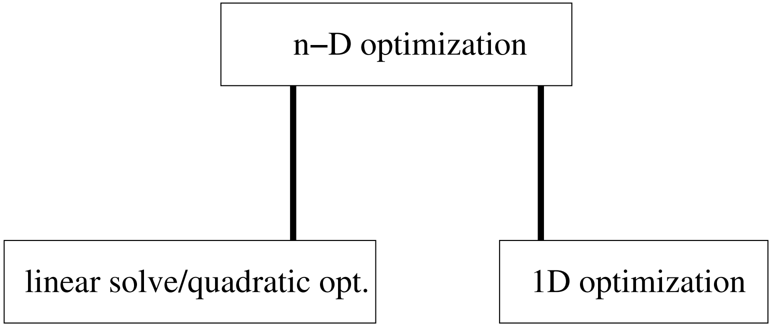
$$\text{find} \quad x_* \text{ s.t. } f(x_*) = \min_{x \in \mathbb{R}^n} \{f(x)\}$$

Quite general, but some cases, like f convex, are fairly solvable.
Today's problem: How about $f : \mathbb{R}^n \to \mathbb{R}$, smooth?

$$\text{find} \quad x_* \text{ s.t. } \nabla f(x_*) = 0$$

We have a reasonable shot at this if f is *twice differentiable*.

Two pillars of smooth multivariate optimization



The simplest example we can get

Quadratic optimization: $f(x) = c - x^t b + \frac{1}{2} x^t A x$.

- very common (actually universal, more later)

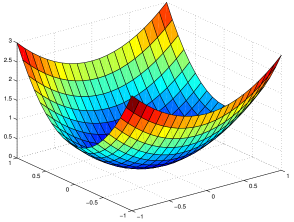
Finding $\nabla f(x) = 0$

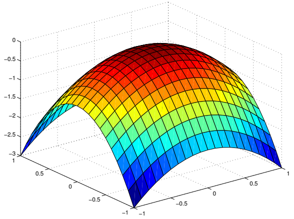
$$\begin{aligned} \nabla f(x) = b - Ax &= 0 \\ x_* &= A^{-1} b \end{aligned}$$

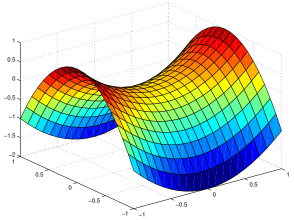
A has to be invertible (really, b in range of A).
Is this all we need?

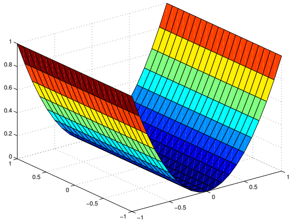
Max, min, saddle, or what?

Require *A* be positive definite, why?









Universality of linear algebra in optimization

$$f(x) = c - x^t b + \frac{1}{2} x^t A x$$

Linear solve: $x_* = A^{-1} b$.
Even for non-linear problems: if optimal x_* near our x

$$\begin{aligned} f(x_*) &\sim f(x) + (x_* - x)^t \nabla f(x) + \frac{1}{2} (x_* - x)^t \nabla\nabla f(x) (x_* - x) + \cdots \\ \Delta x &= x_* - x \sim -(\nabla\nabla f(x))^{-1} \nabla f(x) \end{aligned}$$

Optimization $\leftrightarrow$ Linear solve

Linear solve

$$x = A^{-1}b$$

But really we just want to solve

$$Ax = b$$

Don't form A^{-1} if you can avoid it.
(Don't form A if you can avoid that!)

For a general A, there are three important special cases,

- diagonal: $A = \begin{pmatrix} a_1 & 0 & 0 \\ 0 & a_2 & 0 \\ 0 & 0 & a_3 \end{pmatrix}$ thus $x_i = \frac{1}{a_i} b_i$
- orthogonal $A^t A = I$, thus $A^{-1} = A^t$ and $x = A^t b$
- triangular: $A = \begin{pmatrix} a_{11} & 0 & 0 \\ a_{21} & a_{22} & 0 \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$, $x_i = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j \right)$

Direct methods

A is symmetric positive definite.

Cholesky factorization:

$$A = LL^t,$$

where L lower triangular. So $x = L^{-t}\left(L^{-1}b\right)$ by

$$\begin{aligned} Lz &= b, \quad z_i = \frac{1}{L_{ii}}\left(b_i - \sum_{j<i} L_{ij} z_j\right) \\ L^t x &= z, \quad x_i = \frac{1}{L_{ii}}\left(z_i - \sum_{j>i} L_{ij} x_j\right) \end{aligned}$$

Direct methods

A is symmetric positive definite.

Eigenvalue factorization:

$$A = QDQ^t,$$

where Q is orthogonal and D is diagonal. Then

$$x = Q\left(D^{-1}\left(Q^t b\right)\right).$$

More expensive than Choesky

Direct methods are usually quite expensive ($O(n^3)$ work).

Iterative method basics

What's an iterative method?

Definition (Informal definition)

An *iterative method* is an algorithm $\mathcal{A}$ which takes what you have, x_i, and gives you a new x_{i+1} which is *less bad* such that $x_1, x_2, x_3, \ldots$ converges to some x_* with badness$= 0$.

A notion of *badness* could come from

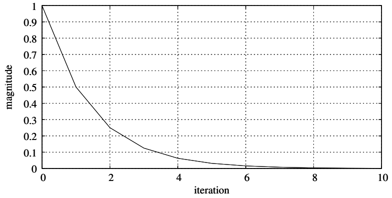
1. distance from x_i to our problem solution
2. value of some objective function above its minimum
3. size of the gradient at x_i

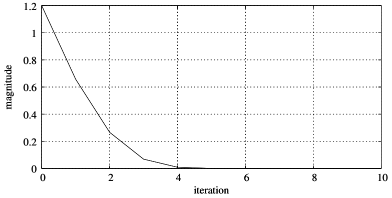
e.g. If x is supposed to satisfy $Ax = b$, we could take $||b - Ax||$ to be the measure of badness.

Iterative method considerations

How expensive is one $x_i \rightarrow x_{i+1}$ step?
How quickly does the badness decrease per step?
A thousand and one years of experience yields two cases

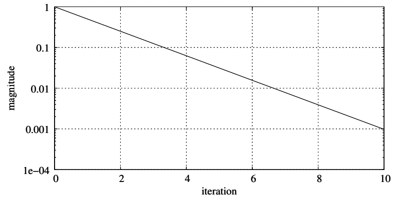
1. $B_i \propto \rho^i$ for some $\rho \in (0, 1)$ (**linear**)
2. $B_i \propto \rho^{(\alpha^i)}$ for $\rho \in (0, 1), \alpha > 1$ (**superlinear**)

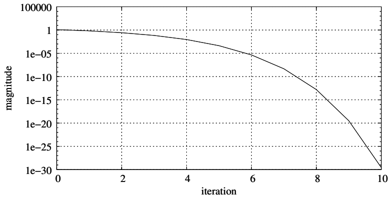




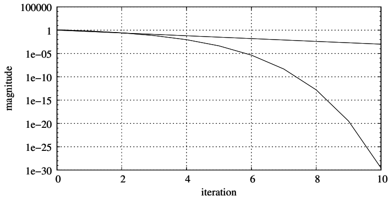
Can you tell the difference?

Convergence





Now can you tell the difference?



When evaluating an iterative method against manufacturer's claims, be sure to do semilog plots.

Iterative methods

Motivation: directly optimize $f(x) = c - x^t b + \frac{1}{2} x^t A x$.

gradient descent:

1. Search direction: $r_i = -\nabla f = b - Ax_i$
2. Search step: $x_{i+1} = x_i + \alpha_i r_i$
3. Pick alpha: $\alpha_i = \frac{r_i^t r_i}{r_i^t A r_i}$ minimizes $f(x + \alpha r_i)$

$$\begin{aligned} f(x_i + \alpha r_i) &= c - x_i^t b + \frac{1}{2} x_i^t A x_i + \alpha r_i^t (Ax_i - b) + \frac{1}{2} \alpha^2 r_i^t A r_i \\ &= f(x_i) - \alpha r_i^t r_i + \frac{1}{2} \alpha^2 r_i^t A r_i \end{aligned}$$

(Cost of a step $= 1$ A-multiply.)

Iterative methods

Optimize $f(x) = c - x^t b + \frac{1}{2} x^t A x$.

conjugate gradient descent:

1. Search direction: $d_i = r_i + \beta_i d_{i-1}$, with $r_i = b - Ax_i$.
2. Pick $\beta_i = -\frac{d_{i-1}^t A r_i}{d_{i-1}^t A d_{i-1}}$, ensures $d_{i-1}^t A d_i = 0$.
3. Search step: $x_{i+1} = x_i + \alpha_i d_i$
4. Pick $\alpha_i = \frac{d_i^t r_i}{d_i^t A d_i}$: minimizes $f(x_i + \alpha d_i)$

$$f(x_i + \alpha d_i) \quad = \quad c - x_i^t b + \frac{1}{2} x_i^t A x_i - \alpha d_i^t r_i + \frac{1}{2} \alpha^2 d_i^t A d_i$$

(also means that $r_{i+1}^t d_i = 0$)

Avoid extra A-multiply: using $Ad_{i-1} \propto r_{i-1} - r_i$

$$\beta_i = -\frac{(r_{i-1} - r_i)^t r_i}{(r_{i-1} - r_i)^t d_{i-1}} = -\frac{(r_{i-1} - r_i)^t r_i}{r_{i-1}^t d_{i-1}} = \frac{(r_i - r_{i-1})^t r_i}{r_{i-1}^t r_{i-1}}$$

A cute result

conjugate gradient descent:

1. $r_i = b - Ax_i$
2. Search direction: $d_i = r_i + \beta_i d_{i-1}$ (β s.t. $d_i A d_{i-1} = 0$)
3. Search step: $x_{i+1} = x_i + \alpha_i d_i$ (α minimizes).

Cute result (not that useful in practice)

Theorem (sub-optimality of CG)

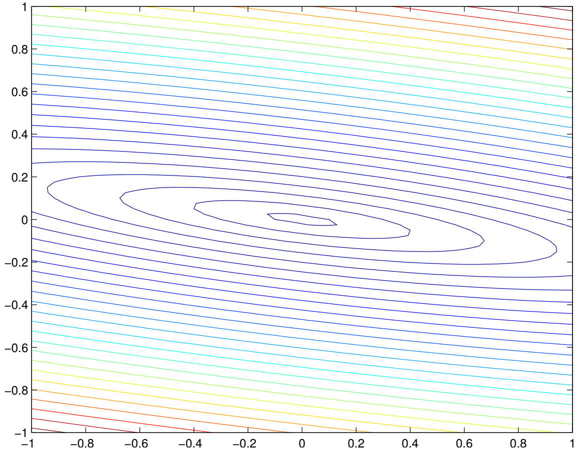
(Assuming $x_0 = 0$) at the end of step k, the solution x_k is the optimal linear combination of $b, Ab, A^2b, \ldots A^kb$ for minimizing

$$c - b^t x + \frac{1}{2} x^t A x.$$

(computer arithmetic errors make this less than perfect)
Very little extra effort. Much better convergence.

Slow convergence: Conditioning

The eccentricity of the quadratic is a big factor in convergence



Convergence and eccentricity

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

For gradient descent,

$$||r_i|| \sim \left|\frac{\kappa - 1}{\kappa + 1}\right|^i$$

For CG,

$$||r_i|| \sim \left|\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right|^i$$

useless CG fact: in exact arithmetic $r_i = 0$ when $i > n$ (A is $n \times n$).

The truth about descent methods

Very slow unless κ can be controlled.
How do we control κ?

$$Ax = b \rightarrow (PAP^t)y = Pb, \quad x = P^t y$$

where P is a *pre-conditioner* you pick.
How to make $\kappa(PAP^t)$ small?

- perfect answer, $P = L^{-1}$ where $L^t L = A$ (Cholesky factorization).
- imperfect answer, $P \sim L^{-1}$

Variations on the theme of *incomplete factorization*:

- $P^{-1} = D^{\frac{1}{2}}$ where $D = \operatorname{diag}(a_{11}, \ldots, a_{nn})$
- more generally, incomplete Cholesky decomposition
- some easy nearby solution or simple approximate A (requiring **domain knowledge**)

Class project?

One idea for a preconditioner is by a *block diagonal* matrix

$$P^{-1} = \begin{pmatrix} L_{11} & 0 & 0 \\ 0 & L_{22} & 0 \\ 0 & 0 & L_{33} \end{pmatrix}$$

where $L_{ii}^t L_{ii} = A_{ii}$ a diagonal block of A.
In what sense does good clustering give good preconditioners?

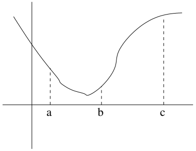
End of solvers: there are a few other iterative solvers out there I haven't discussed.

Second pillar: 1D optimization

1D optimization gives important insights into non-linearity.

$$\min_{s \in \mathbb{R}} f(s), \quad f \text{ continuous.}$$

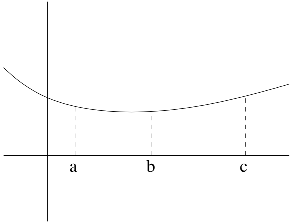
A **derivative-free** option:
A bracket is (a, b, c) s.t. $a < b < c$ and $f(a) > f(b) < f(c)$ then $f(x)$ has a local min for $a < x < b$



Golden search based on picking $a < b' < b < c$ and either $(a < b' < b)$ or $(b' < b < c)$ is a new bracket... continue
Linearly convergent, $e_i \propto G^i$, golden ratio G.

1D optimization

Fundamentally limited accuracy of derivative-free argmin:



Derivative-based methods, $f'(s) = 0$, for accurate argmin

- bracketed: (a, b) s.t. $f'(a), f'(b)$ opposite sign
  1. bisection (linearly convergent)
  2. modified regula falsi & Brent's method (superlinear)
- unbracketed:
  1. secant method (superlinear)
  2. Newton's method (superlinear; requires another derivative)

From quadratic to non-linear optimizations

What can happen when far from the optimum?

- $-\nabla f(x)$ always points in a direction of decrease
- $\nabla\nabla f(x)$ may not be positive definite

For *convex* problems $\nabla\nabla f$ is always positive semi-definite and for strictly convex it is positive definite.
What do we want?

- find a convex neighborhood of x_* (be robust against mistakes)
- apply a quadratic approximation (do linear solve)

Fact: $\forall$ non-linear optimization algorithms, $\exists f$ which fools it.

Naïve Newton's method

Newton's method finding x s.t. $\nabla f(x) = 0$

$$\begin{aligned} \Delta x_i &= -\left(\nabla\nabla f(x_i)\right)^{-1} \nabla f(x_i) \\ x_{i+1} &= x_i + \Delta x_i \end{aligned}$$

Asymptotic convergence, $e_i = x_i - x_*$

$$\begin{aligned} \nabla f(x_i) &= \nabla\nabla f(x_*) e_i + O(||e_i||^2) \\ \nabla\nabla f(x_i) &= \nabla\nabla f(x_*) + O(||e_i||) \\ e_{i+1} &= e_i - \left(\nabla\nabla f_i\right)^{-1} \nabla f_i = O(||e_i||^2) \end{aligned}$$

"squares the error" at every step (exactly eliminates the linear error).

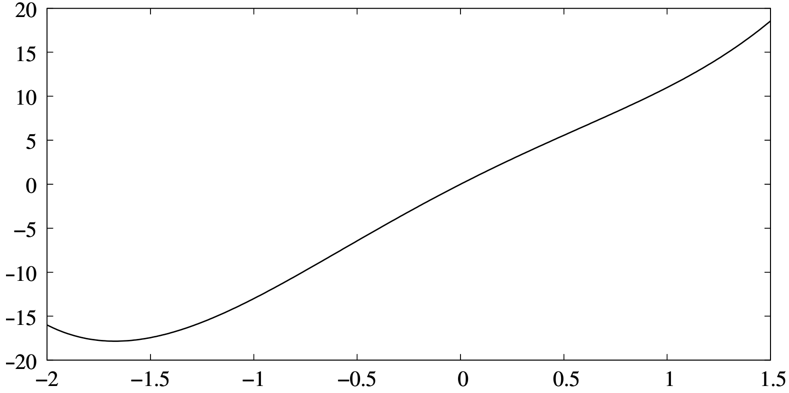
Naïve Newton's method

Sources of trouble

1. if $\nabla\nabla f(x_i)$ not posdef, $\Delta x_i = x_{i+1} - x_i$ might be in an *increasing direction*.
2. if $\nabla\nabla f(x_i)$ posdef, $(\nabla f(x_i))^t \Delta x_i < 0$ so Δx_i is a direction of decrease (could overshoot)
3. even if f is convex, $f(x_{i+1}) \leq f(x_i)$ not assured. $(f(x) = 1 + e^x + \log(1 + e^{-x})$ starting from $x = -2)$.
4. if all goes well, **superlinear convergence**!

1D example of Newton trouble

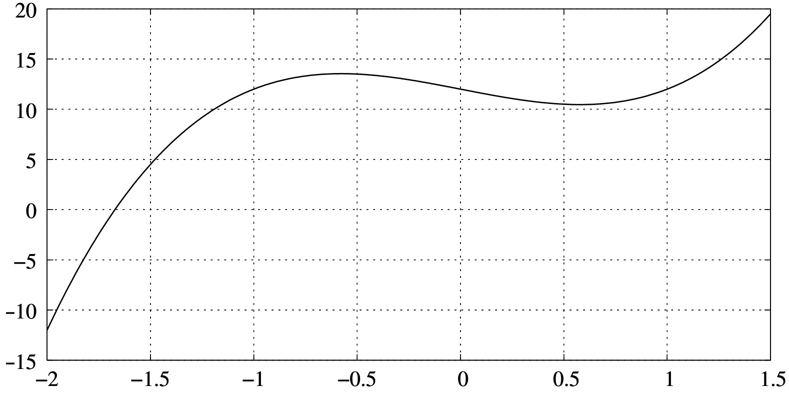
1D example of trouble: $f(x) = x^4 - 2x^2 + 12x$



- Has one local minimum
- Is not convex (note the concavity near x=0)

1D example of Newton trouble

derivative of trouble: $f'(x) = 4x^3 - 4x + 12$



the negative f'' region around $x = 0$ repels the iterates:

$0 \to 3 \to 1.96154 \to 1.14718 \to 0.00658 \to 3.00039 \to 1.96182 \to 1.14743 \to 0.00726 \to 3.00047 \to 1.96188 \to 1.14749 \to \cdots$

Non-linear Newton

Try to enforce $f(x_{i+1}) \leq f(x_i)$

$$\begin{aligned} \Delta x_i &= -(\lambda I + \nabla\nabla f(x_i))^{-1} \nabla f(x_i) \\ x_{i+1} &= x_i + \alpha_i \Delta x_i \end{aligned}$$

Set $\lambda > 0$ to keep Δx_i in a direction of decrease (many heuristics).
Pick $\alpha_i > 0$ such that $f(x_i + \alpha_i \Delta x_i) \leq f(x_i)$. If Δx_i is a direction of decrease, some α_i exists.

- **1D-minimization** do 1D optimization problem,

$$\min_{\alpha_i \in (0,\beta]} f(x_i + \alpha_i \Delta x_i)$$

- **Armijo-search** use this rule: $\alpha_i = \rho\mu^n$ some n

$$f(x_i + s\Delta x_i) - f(x_i) \leq \nu s (\Delta x_i)^t \nabla f(x_i)$$

with ρ, μ, ν fixed (e.g. $\rho = 2, \mu = \nu = \frac{1}{2}$).

Line searching

1D-minimization looks like less of a hack than Armijo. For Newton, asymptotic convergence is not strongly affected, and function evaluations can be expensive.

- far from x_* their only value is ensuring decrease
- near x_* the methods will return $\alpha_i \sim 1$.

If you have a Newton step, accurate line-searching adds little value.

Practicality

Direct (non-iterative, non-structured) solves are expensive!
$\nabla\nabla f$ information is often expensive!

Iterative methods

gradient descent:

1. Search direction: $r_i = -\nabla f(x_i)$
2. Search step: $x_{i+1} = x_i + \alpha_i r_i$
3. Pick alpha: (depends on what's cheap)
   1. linearized $\alpha_i = \frac{r_i^t(\nabla\nabla f)r_i}{r_i^t r_i}$
   2. minimization $f(x_i + \alpha r_i)$ (danger: low quality)
   3. zero-finding $r_i^t \nabla f(x_i + \alpha r_i) = 0$

Iterative methods

conjugate gradient descent:

1. Search direction: $d_i = -r_i + \beta_i d_{i-1}$, with $r_i = -\nabla f(x_i)$.
2. Pick β_i without $\nabla\nabla f$
    1. $\beta_i = \frac{(r_i - r_{i-1})^t r_{i-1}}{(r_i - r_{i-1})^t r_i}$ (Polak-Ribiere)
    2. can also use $\beta_i = \frac{r_i^t r_i}{r_{i-1}^t r_{i-1}}$ (Fletcher-Reeves)
3. Search step: $x_{i+1} = x_i + \alpha_i d_i$
    1. linearized $\alpha_i = \frac{d_i^t (\nabla\nabla f) d_i}{r_i^t d_i}$
    2. 1D minimization $f(x_i + \alpha d_i)$ (danger: low quality)
    3. zero-finding $d_i^t \nabla f(x_i + \alpha d_i) = 0$

Don't forget the truth about iterative methods

To get good convergence you must precondition!

$$B \sim (\nabla\nabla f(x_*))^{-1}$$

Without pre-conditioner

1. Search direction: $d_i = -r_i + \beta_i d_{i-1}$, with $r_i = -P^t \nabla f(x_i)$.
2. Pick $\beta_i = \frac{(r_i - r_{i-1})^t r_{i-1}}{(r_i - r_{i-1})^t r_i}$ (Polak-Ribiere)
3. Search step: $x_{i+1} = x_i + \alpha_i d_i$
4. zero-finding $d_i^t \nabla f(x_i + \alpha d_i) = 0$

with $B = PP^t$ *change of metric*

1. Search direction: $d_i = -r_i + \beta_i d_{i-1}$, with $r_i = -\nabla f(x_i)$.
2. Pick $\beta_i = \frac{(r_i - r_{i-1})^t B r_{i-1}}{(r_i - r_{i-1})^t r_i}$
3. Search step: $x_{i+1} = x_i + \alpha_i d_i$
4. zero-finding $d_i^t B \nabla f(x_i + \alpha d_i) = 0$

What else?

Remember this cute property?

Theorem (sub-optimality of CG)

(Assuming $x_0 = 0$) at the end of step k, the solution x_k is the optimal linear combination of $b, Ab, A^2b, \dots A^kb$ for minimizing

$$c - b^t x + \frac{1}{2} x^t A x.$$

In a sense, CG *learns* about A from the history of $b - Ax_i$. Noting,

1. computer arithmetic errors ruin this nice property quickly
2. non-linearity ruins this property quickly

Quasi-Newton

Quasi-Newton has much popularity/hype. What if we approximate $(\nabla\nabla f(x_*))^{-1}$ from the data we have

$$\begin{aligned} (\nabla\nabla f(x_*))(x_i - x_{k-1}) &\sim \nabla f(x_i) - \nabla f(x_{k-1}) \\ x_i - x_{k-1} &\sim (\nabla\nabla f(x_*))^{-1}(\nabla f(x_i) - \nabla f(x_{k-1})) \end{aligned}$$

over some fixed-finite history.
Data: $y_i = \nabla f(x_i) - \nabla f(x_{k-1})$, $s_i = x_i - x_{k-1}$ with $1 \le i \le k$
Problem: Find symmetric positive def H_k s.t.

$$H_k y_i = s_i$$

Multiple solutions, but BFGS works best in most situations.

BFGS update

$$H_k = \left(I - \frac{s_k y_k^t}{y_k^t s_k}\right) H_{k-1} \left(I - \frac{y_k s_k^t}{y_k^t s_k}\right) + \frac{s_k s_k^t}{y_k^t s_k}$$

Lemma

The BFGS update minimizes $\min_H ||H^{-1} - H_{k-1}^{-1}||_F^2$ *such that* $Hy_k = s_k$.

Forming H_k not necessary, e.g. $H_k v$ can be recursively computed.

Quasi-Newton

Typically keep about 5 data points in the history.

initialize Set $H_0 = I$, $r_0 = -\nabla f(x_0)$, $d_0 = r_0$ goto 3

1. Compute $r_k = -\nabla f(x_k)$, $y_k = r_{k-1} - r_k$
2. Compute $d_k = H_k r_k$
3. Search step: $x_{k+1} = x_k + \alpha_k d_k$ (line-search)

Asymptotically identical to CG (with $\alpha_i = \frac{d_i^t(\nabla\nabla f)d_i}{r_i^t d_i}$)

Armijo line searching has good theoretical properties. Typically used.

Quasi-Newton ideas generalize beyond optimization (e.g. fixed-point iterations)

Summary

- All multi-variate optimizations relate to posdef linear solves
- Simple iterative methods **require** pre-conditioning to be effective in high dimensions.
- Line searching strategies are highly variable
- Timing and storage of $f, \nabla f, \nabla\nabla f$ are all critical in selecting your method.

f	∇f	concerns	method
fast	fast	2,5	quasi-N (zero-search)
fast	fast	5	CG (zero-search)
fast	slow	1,2,3	derivative-free methods
fast	slow	2,5	quasi-N (min-search)
fast	slow	3,5	CG (min-search)
fast/slow	slow	2,4,5	quasi-N with Armijo
fast/slow	slow	4,5	CG (linearized α)

1=time 2=space 3=accuracy
4=robust vs. nonlinearity 5=precondition

Don't take this table too seriously. . .