# Iterative Methods for Solving Least-squares

When $A$ has full column rank, our L.S. estimate is
$$\hat{x} = (A^TA)^{-1}A^Ty$$

If $A$ is $M \times N$, then constructing $A^TA$ costs $O(MN^2)$ computations, and inverting $A^TA$ costs $O(N^3)$ computations. (Note that for $M > N$, the cost of constructing the matrix is actually larger than inverting it.)

This is a problem for even moderately large $M$ and $N$. But inverse problems with large $M$ & $N$ are common in the modern world.

Typical 3D MRI scan:

Reconstruct a $128 \times 128 \times 128$ cube of voxels from $\approx 5 \cdot 10^6$ Fourier-domain samples

$N \approx 2.1$ million, $M \approx 5$ million

It takes about .25 seconds for me to construct $A^TA$ and solve $A^TAx = A^Ty$ when

$M = 5000, \quad N = 1000$

How long would it take (approx) for the example above? ($M = 5{,}000{,}000$ $N = 2{,}000{,}000$)

How much memory would you need to hold $A$ (assume double precision = 8 bytes per entry)?

In this section, we will overview two iterative methods – steepest descent & conjugate gradients – that reformulate

$$A^T A x = A^T y$$

as an optimization program. Each iteration is simple, and requires one application of $A$ & one application of $A^T$. If $A^T A$ is well conditioned, they can converge in very few iterations (especially CG). This makes the cost of solving this type of L.S. problem dramatically smaller – about the cost of a few to a couple hundred applications of $A$.

Moreover, we don't need to construct $A^T A$ or even $A$ explicitly, all we need is a "black box" which takes a vector $x$ and returns $Ax$. This is especially useful if it takes $\ll O(MN)$ time to apply $A$ or $A^T$

In the MRI example above, it takes about 30 seconds to apply $A$ or $A^T$ using a un-equispaced FFT. CG (which we will learn about soon) converges in about 50 iterations $\Rightarrow$ the problem is solved in $\approx 50$ minutes. Also, the storage requirements are $O(M+N)$.

## Recasting as an optimization program

We want to solve

$$\underbrace{A^T A}_{H} x = \underbrace{A^T y}_{b}$$

or

$$H x = b \qquad H : N \times N$$

Note $H$ is symmetric + def (if $A$ has full column rank).

Since $H$ is sym+def, then the solution $\hat{x}$ to

$$\min_{x \in \mathbb{R}^N} \frac{1}{2} x^T H x - x^T b \qquad (QP)$$

will satisfy $H\hat{x} = b$.

Why? Well, if $H$ is sym+def, then (QP) is convex (strictly convex, actually). Thus a necessary and sufficient condition for $\hat{x}$ to be the minimizer is

$$\nabla_x \left( \frac{1}{2} x^T H x - x^T b \right) \Big|_{x=\hat{x}} = 0$$

gradient w.r.t. $x$

Since

$$\nabla_x \left( \frac{1}{2} x^T H x - x^T b \right) = \frac{1}{2} \nabla_x (x^T H x) - \nabla_x (x^T b)$$

$$= H x - b$$

the solution to (QP) must have

$$H \hat{x} = b.$$

## Steepest Descent

Say you have an unconstrained optimization program

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

where $f(x): \mathbb{R}^N \to \mathbb{R}$ is convex. One way to solve this program is to simply "roll downhill". That is, from a starting point $x_0$ we move to

$$x_1 = x_0 - \alpha_0 \cdot \nabla f(x)\Big|_{x=x_0}$$

then to

$$x_2 = x_1 - \alpha_1 \nabla f(x)\Big|_{x=x_1}$$

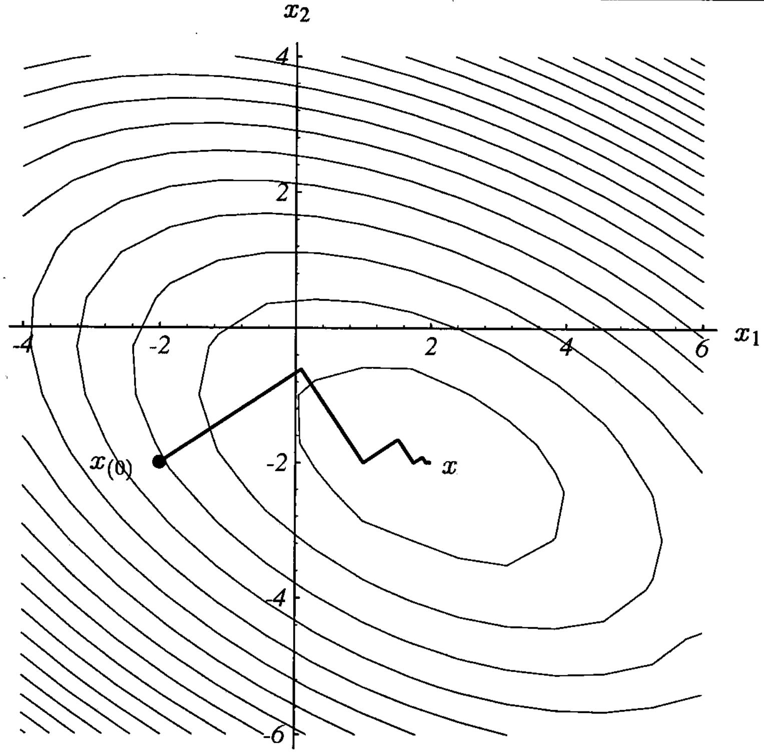
$$\vdots$$

$$x_k = x_{k-1} - \alpha_{k-1} \nabla f(x)\Big|_{x=x_{k-1}}$$

for some appropriate $\alpha_1, \alpha_2, \ldots$ ; $\alpha_k > 0$.

At each iteration we are moving in the direction $-\nabla f(x_k)$ ; this is the direction of "steepest descent".

from Jonathan Richard Shewchuk "Intro to CG"




For our problem

$$\min_x \; x^T H x - x^T b$$

the gradient is simply the residual

$$-\nabla\left(x^T H x - x^T b\right)\Big|_{x=x_k} = b - Hx_k =: r_k$$

and so the iteration becomes

$$x_{k+1} = x_k + \alpha_k r_k$$

There is a nifty way to choose an optimal value for the $\alpha_k$. We want to choose $\alpha_k$ so that $f(x_{k+1})$ is as small as possible. Along

indexed by $\alpha_k$ the ray $x_k + \alpha_k r_k$, $f(x_k + \alpha_k r_k)$ is again convex (in $\alpha_k$) so we want

$$\frac{d}{d\alpha_k} f(x_k + \alpha_k r_k) = 0$$

By the chain rule

$$\frac{d}{d\alpha_k} f(x_{k+1}) = \nabla f(x_{k+1})^T \frac{d}{d\alpha} x_{k+1}$$

$$= \nabla f(x_{k+1})^T r_k$$

so we need to choose $\alpha_k$ such that

$$\nabla f(x_{k+1}) \perp r_k$$

or more concisely

$$r_{k+1} \perp r_k \qquad (r_{k+1}^T r_k = 0)$$

So let's do this:

$$r_{k+1}^T r_k = 0$$

$$\Rightarrow \quad (b - Hx_{k+1})^T r_k = 0$$

$$\Rightarrow \quad (b - H(x_k + \alpha_k r_k))^T r_k = 0$$

$$\Rightarrow \quad (b - Hx_k)^T r_k - \alpha_k r_k^T H r_k = 0$$

$$\Rightarrow \quad r_k^T r_k - \alpha_k r_k^T H r_k = 0$$

$$\Rightarrow \quad \alpha_k = \frac{r_k^T r_k}{r_k^T H r_k}$$

So the steepest descent algorithm is

- Initialize $x_k$ = some guess, $k=0$
- while not coverged

$$\left[\begin{array}{l} r_k = b - Hx_k \\ \alpha_k = r_k^T r_k / r_k^T H r_k \\ x_{k+1} = x_k + \alpha_k r_k \\ k = k+1 \end{array}\right.$$

Notice that

$$r_{k+1} = b - Hx_{k+1} = b - H(x_k + \alpha_k r_k)$$

$$= r_k - \alpha_k H r_k$$

So we can save an application of $H$ using

Initialize $k = 0$

$x_0 =$ some guess

$r_0 = b - Hx_0$

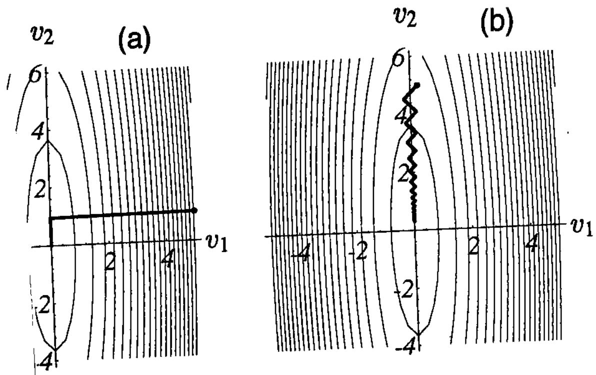
While not converged

$$\left[\begin{aligned} & q = H r_k \\ & \alpha_k = r_k^T r_k / r_k^T q \\ & x_{k+1} = x_k + \alpha_k r_k \\ & r_{k+1} = r_k - \alpha_k q \end{aligned}\right.$$

30

The effectiveness of S.D. depends critically on how $H$ is conditioned and the starting point

From Shewchuk:



## The conjugate gradient (CG) method

An excellent resource for the material in this section is

J. Shewchuk: "An introduction to the conjugate gradient method without the agonizing pain"

We can see from the example on the last page that steepest descent is sometimes inefficient because it can move in essentially the same direction many times.

CG avoids this by ensuring that each step is $\perp$ (in an appropriate inner product) to all of the previous steps that have been taken.

Consider the general iteration

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

direction to move @ $k^{th}$ iteration

What we'd like is to choose $\alpha_K$ so that

$$e_{K+1} = x_{K+1} - x$$

the error at iteration $K+1$ is orthogonal to the direction $d_K$. This would mean (thanks to the $\perp$-principle) that we are optimally aligned with $x$ (the true solution) along the direction $d_K$. Then if $d_0, d_1, \ldots, d_{N-1}$ is a set of $\perp$ directions we would only need to move in each exactly once.

We would like $\alpha_K$ s.t.

$$d_K \perp e_{K+1} \quad \text{i.e.} \quad \langle d_K, e_{K+1}\rangle = d_K^T e_{K+1} = 0$$

so we would need

$$d_K^T(x_K + \alpha_K d_K - x) = 0$$

$$\Rightarrow d_K^T(e_K + \alpha_K d_K) = 0$$

$$\Rightarrow \alpha_K = -\frac{d_K^T e_K}{d_K^T d_K}$$

This is fine, except we have no idea what $e_K$ is (if we did, we could solve the entire problem instantly).

What we can do is choose the $\alpha_k$ such that $d_k$ is $H$-$\perp$ to $e_{k+1}$. That is,

$$\langle d_k, e_{k+1} \rangle_H = d_k^T H e_{k+1} = 0$$

(Recall that if $H$ is an $N \times N$ sym+def matrix, then $\langle x, y \rangle_H = x^T H y$ is a valid inner product on $\mathbb{R}^N$.)

We need

$$d_k^T H (x_k + \alpha_k d_k - x) = 0$$

$$\Rightarrow \quad d_k^T H e_k + \alpha_k d_k^T H d_k = 0$$

$$\Rightarrow \quad \alpha_k = \frac{-d_k^T H e_k}{d_k^T H d_k}$$

Notice that

residual

$$H e_k = H(x_k - x) = H x_k - b = -r_k$$

which we definitely know, so we could take

$$\alpha_k = \frac{d_k^T r_k}{d_k^T H d_k}$$

So given a set of $H$-$\perp$ search directions $d_0, d_1, \ldots, d_{N-1}$, $\langle d_i, d_j \rangle_H = 0$ $i \neq j$

We could implement the following

Initialize $x_0$ = some guess

for $k = 0$ to $N-1$

$$\left[\begin{array}{l} r_k = b - Hx_k \\ \alpha_k = d_k^T r_k / d_k^T H d_k \\ \\ x_{k+1} = x_k + \alpha_k d_k \end{array}\right.$$

This procedure would converge to the <u>exact</u> solution after the $N^{th}$ step (more on this later).

All we need now is a set of $H$-$\perp$ direction vectors $d_j$. The beauty of CG is that it generates these directions "on the fly" by running what is essentially Gram-Schmidt.

Here is the CG algorithm:

Initialize $x_0$ = some guess
$r_0 = b - Hx_0$
$d_0 = r_0$

for $k = 0$ up to $N-1$

$$\left[\begin{array}{l} \alpha_k = r_k^T r_k / d_k^T H d_k \\ \\ x_{k+1} = x_k + \alpha_k d_k \\ r_{k+1} = r_k - \alpha_k H d_k \\ \beta_{k+1} = r_{k+1}^T r_{k+1} / r_k^T r_k \\ d_{k+1} = r_{k+1} + \beta_{k+1} d_k \end{array}\right.$$

(We will show below that $r_k^T r_k = r_k^T d_k$, so the $\alpha_k$ really is the same as before.)

These choices of $\alpha_k$ and $\beta_{k+1}$ are ensuring two important things:

① $\langle r_\ell, r_{k+1}\rangle = 0$ for $\ell = 0, \ldots, j$ "the residuals are $\perp$"

② $\langle d_\ell, d_{k+1}\rangle_H = 0$ for $\ell = 0, \ldots, k$ "the directions are $H$-$\perp$"

We can establish these two facts by induction. We start with the following:

① $\langle r_0, r_1 \rangle = r_0^T r_1 = 0$

since
$$r_1 = r_0 - \frac{r_0^T r_0}{r_0^T H r_0} \cdot H r_0$$

$$\Rightarrow r_0^T r_1 = r_0^T r_0 - r_0^T r_0 \frac{r_0^T H r_0}{r_0^T H r_0}$$
$$= 0$$

② $\langle d_0, d_1 \rangle_H = d_0^T H d_1 = 0$

since
$$r_1 = r_0 - \alpha_0 H d_0$$
$$\Rightarrow r_1^T r_1 = r_1^T r_0 - \alpha_0 r_1^T H d_0 \qquad (\text{since } r_1^T r_0 = 0)$$
$$\Rightarrow r_1^T H d_0 = \frac{-1}{\alpha_0} \cdot r_1^T r_1$$

and also
$$d_1 = r_1 + \frac{r_1^T r_1}{r_0^T r_0} \cdot d_0$$
$$\Rightarrow d_0^T H d_1 = d_0^T H r_1 + \frac{r_1^T r_1}{r_0^T r_0} \cdot d_0^T H d_0$$
$$= \frac{-r_1^T r_1}{r_0^T r_0} \cdot d_0^T H d_0 + \frac{r_1^T r_1}{r_0^T r_0} \cdot d_0^T H d_0 = 0.$$

Now at step $k+1$, suppose we have

$$\langle r_\ell, r_j \rangle = r_\ell^T r_j = 0 \quad \forall j, \ell \le k$$

$$\langle d_\ell, d_j \rangle_H = d_\ell^T H d_j = 0 \quad \forall j, \ell \le k$$

Then we will also have the following:

① $$\langle r_\ell, r_{k+1} \rangle = r_\ell^T r_{k+1} = 0 \quad \forall \ell \le k$$

To see this, first note that

$$r_\ell^T H d_k = (d_\ell - \beta_\ell d_{\ell-1})^T H d_k$$

$$= \begin{cases} d_k^T H d_k & \ell = k \\ 0 & \ell < k \end{cases} \quad \text{since } \langle d_\ell, d_k \rangle_H = 0 \text{ for } \ell < k$$

As a result

$$r_\ell^T r_{k+1} = r_\ell^T r_k - \frac{r_k^T r_k}{d_k^T H d_k} r_\ell^T H d_k = 0 \quad \forall \ell \le k$$

② $\langle d_\ell, d_{k+1} \rangle_H = d_\ell^T H d_{k+1} = 0 \quad \forall \ell \le k$

This follows from the expansion

$$d_\ell^T H d_{k+1} = d_\ell^T H r_{k+1} + \beta_{k+1} d_\ell^T H d_k$$

Notice that

$$r_i^T r_{k+1} = r_i^T r_k - \alpha_k r_i^T H d_k$$

$$\Rightarrow \quad r_i^T H d_k = \begin{cases} \frac{1}{\alpha_k} r_k^T r_k & i = k \\ \frac{-1}{\alpha_k} r_{k+1}^T r_{k+1} & i = k+1 \\ 0 & i < k \end{cases} \qquad (*)$$

Then for $\ell = k$

$$\begin{aligned} d_k^T H d_{k+1} &= \frac{-1}{\alpha_k} r_{k+1}^T r_{k+1} + \beta_{k+1} d_k^T H d_k \\ &= \frac{-r_{k+1}^T r_{k+1}}{r_k^T r_k} \cdot d_k^T H d_k + \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \cdot d_k^T H d_k \\ &= 0 \end{aligned}$$

and for $l < k$

$$d_l^T H d_{k+1} = \underbrace{d_l^T H r_{k+1}}_{\substack{=0 \text{ by } (*) \\ \text{on last page}}} + \underbrace{\beta_{k+1} d_l^T H d_k}_{\substack{=0 \text{ since the } d_0, d_1, \ldots, d_k \\ \text{are } H\perp \text{ already}}}$$

So we have established that the direction $d_k$ that CG moves on iteration $k$ is $H$-$\perp$ to all previous directions.

Now we will establish that this means that CG will converge <u>exactly</u> in at most $N$ iterations.

Let the error at the $k^{\text{th}}$ iteration be

$$e_k = x_k - x$$

Notice that since $x_{k+1} = x_k + \alpha_k d_k$

$$\Rightarrow \quad e_{k+1} = x_{k+1} - x$$
$$= e_k + \alpha_k d_k$$

Unrolling this expression, we get

$$e_k = e_0 + \sum_{j=0}^{k-1} \alpha_j d_j$$

Now remember that $d_0, \ldots, d_{N-1}$ are all $H$-$\perp$. Thus

$$\frac{d_0}{\|d_0\|_H}, \frac{d_1}{\|d_1\|_H}, \ldots, \frac{d_{N-1}}{\|d_{N-1}\|_H}$$ is an $H$-orthobasis for $\mathbb{R}^N$

As such, we can write

$$e_0 = \sum_{j=0}^{N-1} \left\langle e_0, \frac{d_j}{\|d_j\|_H} \right\rangle_H \cdot d_j / \|d_j\|_H$$

$$= \sum_{j=0}^{N-1} \frac{\langle e_0, d_j \rangle_H}{\|d_j\|_H^2} \cdot d_j$$

(since $d_j$ are $H$-$\perp$)

$$= \sum_{j=0}^{N-1} \frac{\langle e_0 + \sum_{i=0}^{j-1} \alpha_i d_i, d_j \rangle_H}{\|d_j\|_H^2} \cdot d_j$$

$$= \sum_{j=0}^{N-1} \frac{d_j^T H e_j}{d_j^T H d_j} \cdot d_j$$

$$= \sum_{j=0}^{N-1} \frac{-d_j^T r_j}{d_j^T H d_j} \cdot d_j \quad \text{since } He_j = -r_j$$

This seems like the appropriate place to mention that $d_j^T r_j = r_j^T r_j$, since

$$
\begin{aligned}
d_j &= r_j + \beta_j d_{j-1} \\
&= r_j + \beta_j r_{j-1} + \beta_j \beta_{j-1} d_{j-2} \\
&= r_j + \beta_j r_{j-1} + \beta_j \beta_{j-1} r_{j-2} + \beta_j \beta_{j-1} \beta_{j-2} d_{j-3} \\
&\qquad \text{etc} \\
&= r_j + \sum_{i=0}^{j-1} \gamma_i r_i \quad \text{for some scalars } \gamma_i
\end{aligned}
$$

$$
\Rightarrow r_j^T d_j = r_j^T r_j + \cancel{\sum_{i \le j-1} \gamma_i r_j^T r_i}^{\,0}
$$

Thus

$$
e_0 = \sum_{j=0}^{N-1} -\alpha_j \cdot d_j
$$

and

$$
\begin{aligned}
e_K &= e_0 + \sum_{j=0}^{K-1} \alpha_j d_j \\
&= -\sum_{j=0}^{N-1} \alpha_j d_j + \sum_{j=0}^{K-1} \alpha_j d_j \\
&= -\sum_{j=K}^{N-1} \alpha_j d_j = -\sum_{j=K}^{N-1} \alpha_j \cdot \|d_j\|_H \cdot \frac{d_j}{\|d_j\|_H}
\end{aligned}
$$

By Parseval (for $\langle \cdot, \cdot \rangle_H$), we have

$$\|e_k\|_H^2 = \sum_{j=k}^{N-1} \alpha_j^2 / \|d_j\|_H^2$$

This is obviously monotonically decreasing with $k$, and

$$\|e_N\|_H^2 = 0$$

$$\Rightarrow \quad \boxed{x_N = x = \text{true solution !}}$$

Since each iteration of CG is a matrix-vector multiply — which is $O(N^2)$ — and we converge in $N$ iterations, CG solves $Hx = b$ in $O(N^3)$ computations in general, the same as with other solvers.

BUT, if $H$ is specially structured so that it takes $\ll O(N^2)$ computations to apply, then CG takes advantage of this. The real cost is $N$ applications of $H$.

In addition, it is often the case that $\|e_k\|_H^2$ is acceptably small for relatively modest values of $k$. This is particularly true if $H$ is well-conditioned.

Moral: CG can get an approximate (but still potentially very good) solution using <u>much</u> less computation than solving the system directly.

It also significantly outperforms steepest descent.

## Convergence Guarantees

How many iterations do we need for steepest descent or CG to coverge within a certain precision? There are "worst case" bounds that depend on the condition number $\mathbb{K}$ of $H$

$$\mathbb{K} = \frac{\lambda_{max}(H)}{\lambda_{min}(H)} = \frac{\text{max eigenvalue}}{\text{min eigenvalue}}$$

For steepest descent, we will have

$$\|e_k\|_H \leq \delta \cdot \|e_0\|_H$$

in at most

$$k \leq \left\lceil \frac{1}{2} \mathbb{K} \cdot \log(1/\delta) \right\rceil$$

iterations.

For CG, we need at most

$$k \leq \left\lceil \frac{1}{2} \sqrt{\mathbb{K}} \cdot \log(2/\delta) \right\rceil$$

(these are "natural logs")

See Shewchuk for a nice derivation of these.

Say the condition number is $\kappa = 100$. How many iterations do you need to get 6 digits of precision ($\delta = 10^{-6}$)?

$$\text{SD: } \left\lceil \frac{1}{2} \cdot 100 \cdot \log(10^6) \right\rceil = 691$$

$$\text{CG: } \left\lceil \frac{1}{2} \cdot 10 \cdot \log(2 \cdot 10^6) \right\rceil = 73$$

Again, these are worst-case bounds, and performance is typically better.