Iterative methods for solving least-squares

Let's return now to our fundamental (regularized) least-squares estimators. When we are using a finite-dimensional basis, we have two forms for the estimator:

$$\hat{\boldsymbol{x}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A} + \delta\mathbf{I})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y} = \boldsymbol{A}^{\mathrm{T}}(\boldsymbol{A}\boldsymbol{A}^{\mathrm{T}} + \delta\mathbf{I})^{-1}\boldsymbol{y}.$$

Likewise, when we are using a kernel, the estimate map is parameterized by

$$\hat{\boldsymbol{\alpha}} = (\boldsymbol{K} + \delta\mathbf{I})^{-1}\boldsymbol{y}.$$

The first expression above requires solving an $N \times N$ system of equations, while the second require solving an $M \times M$ system (remember that M is the number of data points). The costs involved in solving these systems can be significant; if $\boldsymbol{A}$ is $M \times N$, then constructing $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ can cost $O(MN^2)$ computations, and solving the $N \times N$ system $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$ costs $O(N^3)$ computations. (Note that for $M \geq N$, the cost of constructing the matrix can actually exceed the cost to solve the system — but if we have the "kernel trick", the cost of constructing the matrix can be greatly reduced.)

This cost can be prohibitive for even moderately large M and N. But learning and inverse problems with large M and N are common in the modern world. For example, a typical 3D MRI scan will try to reconstruct a $128 \times 128 \times 128$ cube of voxels from about 5 million non-uniformly spaced samples in the spatial Fourier domain. In this case, the matrix $\boldsymbol{A}$, which models the MRI machine, is $M = 5 \cdot 10^6$ by $N = 2.1 \cdot 10^6$.

With those values, MN^2 is huge ($\sim 10^{19}$); even storing the matrix $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ in memory would require terabytes of RAM.

In this section, we will overview two iterative methods — **steepest descent** and **conjugate gradients** — that solve symmetric positive-definite systems of equations

$$\boldsymbol{H}\boldsymbol{x} = \boldsymbol{b}$$

as an optimization program and then solve it by an iterative descent method. Each iteration is simple, requiring one or two applications of $\boldsymbol{H}$.

If $\boldsymbol{H}$ is well-conditioned, then these methods can converge in very few iterations (especially conjugate gradients). This makes the cost of solving a least-squares problem dramatically smaller — about the cost of a few hundred applications of $\boldsymbol{H}$.

Moreover, we will not need to construct $\boldsymbol{H}$ explicitly. All we need is a "black box" which takes a vector $\boldsymbol{x}$ and returns $\boldsymbol{H}\boldsymbol{x}$. This is especially useful if it takes $\ll O(N^2)$ operations to apply $\boldsymbol{H}$.

In the MRI example above, it takes about one second to apply $\boldsymbol{H} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, and the conjugate gradients method converges in about 50 iterations, meaning that the problem can be solved in less than a minute. Also, the storage requirement is on the order of $O(M+N)$, rather than $O(MN)$.

Recasting sym+def linear equations as an optimization program

We want to solve systems like

$$\underbrace{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}}_{\boldsymbol{H}}\boldsymbol{x} = \underbrace{\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}}_{\boldsymbol{b}} \quad \text{or} \quad \underbrace{(\boldsymbol{K} + \delta\mathbf{I})}_{\boldsymbol{H}}\alpha = \underbrace{\boldsymbol{y}}_{\boldsymbol{b}}.$$

Note that in both cases above, $\boldsymbol{H}$ is symmetric and positive definite (assuming in the former case that $\boldsymbol{A}$ has full column rank). This means that the following quadratic program is **convex**:

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}. \tag{1}$$

The consequence of this is that a necessary and sufficient condition for $\hat{\boldsymbol{x}}$ to be the minimizer of (1) is the gradient (evaluated at $\hat{\boldsymbol{x}}$) to be zero:

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

Since

$$\begin{aligned} \nabla_{\boldsymbol{x}} \left(\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b} \right) &= \frac{1}{2}\nabla_{\boldsymbol{x}}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}) - \nabla_{\boldsymbol{x}}(\boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}) \\ &= \boldsymbol{H}\boldsymbol{x} - \boldsymbol{b}, \end{aligned}$$

this means that $\hat{\boldsymbol{x}}$ is the solution to (1) if and only if

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

Steepest descent

Say you have an unconstrained optimization program

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \, f(\boldsymbol{x})$$

where $f(\boldsymbol{x}) : \mathbb{R}^N \to \mathbb{R}$ is convex. One simple way to solve this program is to simply "roll downhill". If we are sitting at a point

$\boldsymbol{x}_0$, then $f(\cdot)$ decreases the fastest if we move in the direction of the negative gradient $-\nabla f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0}$.

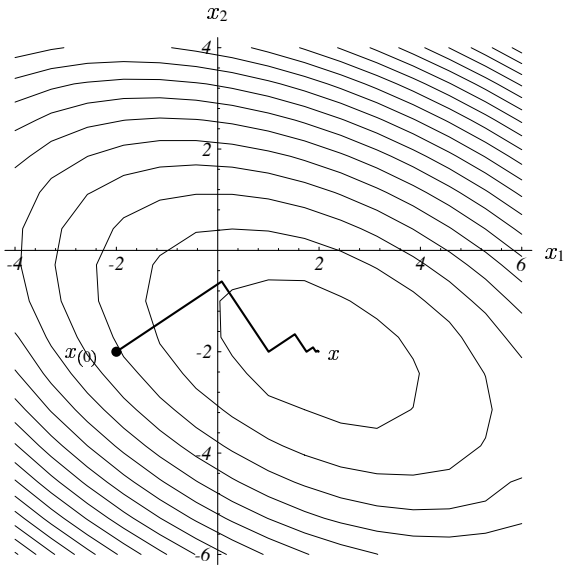
From a starting point $\boldsymbol{x}_0$, we move to

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \alpha_0 \, \nabla f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_0}$$

then to

$$\begin{aligned} \boldsymbol{x}_2 &= \boldsymbol{x}_1 - \alpha_1 \, \nabla f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_1} \\ &\vdots \\ \boldsymbol{x}_k &= \boldsymbol{x}_{k-1} - \alpha_{k-1} \, \nabla f(\boldsymbol{x})|_{\boldsymbol{x}=\boldsymbol{x}_{k-1}} , \end{aligned}$$

where the $\alpha_0, \alpha_1, \ldots$ are appropriately chosen **step sizes**.



(from Shewchuk, "... without the agonizing pain")

For our particular optimization problem

$$\underset{\boldsymbol{x}}{\text{minimize}} \ \frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x} - \boldsymbol{x}^{\mathrm{T}}\boldsymbol{b},$$

we can explicitly compute both the gradient and the best choice of step size. The (negative) gradient is what we call the **residual**, the difference between $\boldsymbol{b}$ and $\boldsymbol{H}$ applied to the current iterate:

$$-\nabla\left(\frac{1}{2}\boldsymbol{x}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{x}-\boldsymbol{x}^{\mathrm{T}}\boldsymbol{b}\right)\Big|_{\boldsymbol{x}=\boldsymbol{x}_k} \;=\; \boldsymbol{b}-\boldsymbol{H}\boldsymbol{x}_k =: \boldsymbol{r}_k.$$

The steepest descent iteration can be written as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k.$$

There is a nifty way to choose an optimal value for the step size α_k. We want to choose α_k so that $f(\boldsymbol{x}_{k+1})$ is as small as possible. It is not hard to show that along the ray $\boldsymbol{x}_k + \alpha \boldsymbol{r}_k$ for $\alpha \geq 0$,

$$f(\boldsymbol{x}_k + \alpha \boldsymbol{r}_k) \quad \text{is convex as a function of } \alpha.$$

Thus we can choose the value of α that makes the derivative of this function zero; we want

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} f(\boldsymbol{x}_k + \alpha \boldsymbol{r}_k) = 0.$$

By the chain rule,

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}\alpha} f(\boldsymbol{x}_{k+1}) &= \nabla f(\boldsymbol{x}_{k+1})^{\mathrm{T}} \frac{\mathrm{d}}{\mathrm{d}\alpha}\boldsymbol{x}_{k+1} \\ &= \nabla f(\boldsymbol{x}_{k+1})^{\mathrm{T}} \boldsymbol{r}_k.\end{aligned}$$

So we need to choose α_k such that

$$\nabla f(\boldsymbol{x}_{k+1}) \perp \boldsymbol{r}_k,$$

or more concisely

$$\boldsymbol{r}_{k+1} \perp \boldsymbol{r}_k \qquad (\boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_k = 0).$$

So let's do this

$$\begin{aligned} \boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_k &= 0 \\ \Rightarrow \quad (\boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_{k+1})^{\mathrm{T}} \boldsymbol{r}_k &= 0 \\ \Rightarrow \quad (\boldsymbol{b} - \boldsymbol{H}(\boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k))^{\mathrm{T}} \boldsymbol{r}_k &= 0 \\ \Rightarrow \quad (\boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_k)^{\mathrm{T}} \boldsymbol{r}_k - \alpha_k \, \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_k &= 0 \\ \Rightarrow \quad \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k - \alpha_k \, \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_k &= 0 \end{aligned}$$

and so the optimal step size is

$$\alpha_k = \frac{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k}{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_k}.$$

The steepest descent algorithm performs this iteration until $\|\boldsymbol{H}\boldsymbol{x}_k - \boldsymbol{b}\|_2$ is below some tolerance δ:

Steepest Descent, version 1

Initialize: $\boldsymbol{x}_0 =$ some guess, $k = 0$, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_0$.

while not converged, $\|\boldsymbol{r}_k\|_2 \geq \delta$ **do**

$\alpha_k = \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k / \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_k$

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \, \boldsymbol{r}_k$

$\boldsymbol{r}_{k+1} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_{k+1}$

$k = k + 1$

end while

There is a nice trick that can save us one of two applications of $\boldsymbol{H}$ needed in each iteration above. Notice that

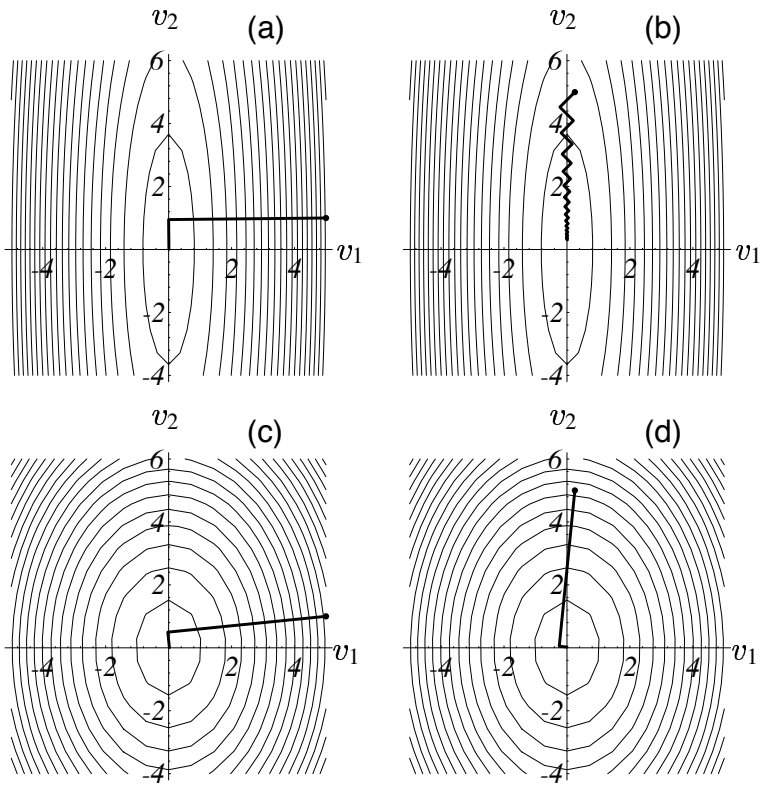
$$\begin{aligned}\boldsymbol{r}_{k+1} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_{k+1} &= \boldsymbol{b} - \boldsymbol{H}(\boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k) \\ &= \boldsymbol{r}_k - \alpha_k \boldsymbol{H}\boldsymbol{r}_k.\end{aligned}$$

So we can save an application of $\boldsymbol{H}$ by updating the residual rather than recomputing it at each stage.

Steepest Descent, more efficient version 2

Initialize: $\boldsymbol{x}_0 =$ some guess, $k = 0$, $\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_0$.
while not converged, $\|\boldsymbol{r}_k\|_2 \geq \delta$ **do**
$\boldsymbol{q} = \boldsymbol{H}\boldsymbol{r}_k$
$\alpha_k = \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{r}_k / \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{q}$
$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{r}_k$
$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{q}$
$k = k + 1$
end while

The effectiveness of SD depends critically on how $\boldsymbol{H}$ is conditioned and the starting point. Consider the two examples on the next page.



(from Shewchuk, "... without the agonizing pain")

When the conditioning of $\boldsymbol{H}$ is poor and we choose a bad starting point, convergence can take many iterations even in simple cases.

Convergence analysis for steepest descent

We can do some clean analysis of a simplified version of the steepest descent algorithm above that really bring out the role of the conditioning of $\boldsymbol{H}$. Suppose that $\boldsymbol{H}$ has eigenvalues[1]

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N > 0$$

[1]Recall that $\boldsymbol{H}$ is positive definite, so all of its eigenvalues are positive.

and take $\kappa = \lambda_1/\lambda_N$ — this is called the **condition number** of $\boldsymbol{H}$.

We will derive a rate of convergence for steepest descent with a *fixed* step size α:

$$\alpha_k = \alpha = \frac{1}{\lambda_1}.$$

This choice is not necessarily the best choice, we are simply fixing it to simplify the analysis. In practice, choosing α_k as above or using other heuristics for choosing the step size almost always are better than fixing it as above or at any other value.

We start by unrolling an iteration of SD. Say that $\boldsymbol{x}_\star$ is the unique solution to the optimization program, so that $\boldsymbol{H}\boldsymbol{x}_\star = \boldsymbol{b}$. Then

$$\begin{aligned}\boldsymbol{x}_{k+1} &= \boldsymbol{x}_k + \alpha(\boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_k) \\ &= \boldsymbol{x}_k + \alpha\boldsymbol{H}(\boldsymbol{x}_\star - \boldsymbol{x}_k),\end{aligned}$$

and so

$$\begin{aligned}\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star &= \boldsymbol{x}_k - \boldsymbol{x}_\star + \alpha\boldsymbol{H}(\boldsymbol{x}_\star - \boldsymbol{x}_k) \\ &= (\mathbf{I} - \alpha\boldsymbol{H})\,(\boldsymbol{x}_k - \boldsymbol{x}_\star).\end{aligned}$$

The distance we are at iteration k from the solution $\boldsymbol{x}_\star$ is $\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2$, and we see that

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star\|_2 \leq \|\mathbf{I} - \alpha\boldsymbol{H}\| \cdot \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2,$$

where $\|\mathbf{I} - \alpha\boldsymbol{H}\|$ is the operator norm of $\mathbf{I} - \alpha\boldsymbol{H}$, i.e. its largest magnitude eigenvalue[2] So at each iteration, the error is being multiplied by a factor of $\|\mathbf{I} - \alpha\boldsymbol{H}\|$; this is good news if this operator norm is than 1, as it means the error gets smaller at each iteration.

[2]While $\boldsymbol{H}$ is symmetric positive definite, $\mathbf{I} - \alpha\boldsymbol{H}$ will definitely have negative values.

Fortunately, our choice of α guarantees that $\|\mathbf{I} - \alpha \boldsymbol{H}\| < 1$. In fact, the eigenvalues of $\mathbf{I} - \alpha \boldsymbol{H}$ are

$$0,\ 1 - \lambda_2/\lambda_1,\ 1 - \lambda_3/\lambda_1,\ \cdots,\ 1 - \lambda_N/\lambda_1.$$

Thus

$$\|\mathbf{I} - \alpha \boldsymbol{H}\| = 1 - \lambda_N/\lambda_1 = 1 - 1/\kappa,$$

and so

$$\|\boldsymbol{x}_{k+1} - \boldsymbol{x}_\star\|_2 \leq \left(1 - \frac{1}{\kappa}\right) \cdot \|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2,$$

which means the distance from the solution at the kth iteration can be bounded in terms of the initial distance

$$\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2 \leq \left(1 - \frac{1}{\kappa}\right)^k \cdot \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2.$$

So if κ^{-1} is close to 1, the error decreases extremely quickly with k; when it is close to 0 (κ large), the error can decrease very slowly. In fact, we can guarantee that the error has decreased by a factor of δ,

$$\|\boldsymbol{x}_k - \boldsymbol{x}_\star\|_2 \leq \delta \cdot \|\boldsymbol{x}_0 - \boldsymbol{x}_\star\|_2,$$

when

$$\left(1 - \frac{1}{\kappa}\right)^k \leq \delta,$$

which happens when

$$k \log\left(1 - \frac{1}{\kappa}\right) \leq \log(\delta).$$

which happens when

$$k \geq \frac{\log(\delta)}{\log\left(1 - \frac{1}{\kappa}\right)},$$

which happens when

$$k \geq \kappa \log\left(\frac{1}{\delta}\right),$$

where the last inequality follows from the fact that $\log(1-t) \leq -t$ for $0 \leq t < 1$.

Notice how much more strongly this bound depends on κ then δ. Demanding factor of 100 more precision adds another $\log(10) \approx 4.6$ iterations, while increasing the condition number by a factor of 100 results in $100\times$ longer run time.

A different, more careful analysis is presented in the Shewchuk paper for the "optimal" (and variable) step size α_k we used in the previous section.

The method of conjugate gradients (CG)

An **excellent** resource for the material in this section is the manuscript: 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 can be inefficient because it can move in essentially the same direction many times.

CG avoids this by ensuring that each step is orthogonal (in an appropriate inner product) to all of the previous steps that have been taken. Miraculously, this can be done with very little overhead.

Suppose for a moment that we pre-determine N step directions $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{N-1}$ that are orthogonal (but not necessarily normalized), $\boldsymbol{d}_j^{\mathrm{T}} \boldsymbol{d}_i = 0$ for $i \neq j$. This means that $\{\boldsymbol{d}_k / \|\boldsymbol{d}_k\|_2,\ k = 0, \ldots, N-1\}$ is an orthobasis for $\mathbb{R}^N$. Then given a starting point $\boldsymbol{x}_0$, the initial error $\boldsymbol{e}_0 = \boldsymbol{x}_0 - \hat{\boldsymbol{x}}$, where $\hat{\boldsymbol{x}}$ is the solution that satisfies $\boldsymbol{H}\hat{\boldsymbol{x}} = \boldsymbol{b}$, can be expanded as

$$\boldsymbol{e}_0 = \sum_{\ell=0}^{N-1} c_\ell \, \frac{\boldsymbol{d}_\ell}{\|\boldsymbol{d}_\ell\|_2}, \quad \text{where} \quad c_\ell = \frac{\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{e}_0}{\|\boldsymbol{d}_\ell\|_2}. \tag{2}$$

Given step sizes $\alpha_0, \alpha_1, \ldots$, the error after the kth step is

$$\begin{aligned}
\boldsymbol{e}_k &= \boldsymbol{x}_k - \hat{\boldsymbol{x}} \\
&= \boldsymbol{x}_{k-1} + \alpha_{k-1}\boldsymbol{d}_{k-1} - \hat{\boldsymbol{x}} = \boldsymbol{e}_{k-1} + \alpha_{k-1}\boldsymbol{d}_{k-1} \\
&= \boldsymbol{e}_{k-2} + \alpha_{k-1}\boldsymbol{d}_{k-1} + \alpha_{k-2}\boldsymbol{d}_{k-2} \\
&\;\;\vdots \\
&= \boldsymbol{e}_0 + \sum_{\ell=0}^{k-1} \alpha_\ell \boldsymbol{d}_\ell.
\end{aligned}$$

Thus, if we choose the step sizes α_k carefully, then we can pick-off a component in (2) at every step. In particular, if we choose

$$\alpha_k = -\frac{c_k}{\|\boldsymbol{d}_\ell\|_2} = \frac{-\boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{e}_0}{\|\boldsymbol{d}_k\|_2^2}, \tag{3}$$

then we have

$$\boldsymbol{e}_k = \sum_{\ell=k}^{N-1} c_k \frac{\boldsymbol{d}_k}{\|\boldsymbol{d}_k\|_2}, \quad \text{and} \quad \|\boldsymbol{e}_k\|_2^2 = \sum_{\ell=k}^{N-1} |c_k|^2.$$

So we see that as k increases, there are fewer and fewer terms in the sum above, steadily decreasing the error until

$$\boldsymbol{e}_N = \boldsymbol{0}.$$

The argument above works for any set of orthogonal step directions $\{\boldsymbol{d}_k\}$. It would be beautiful, except that we **do not know** the initial error $\boldsymbol{e}_0 = \boldsymbol{x}_0 - \hat{\boldsymbol{x}}$. (If we did, we would have a solution in one step: just subtract $\boldsymbol{e}_0$ from $\boldsymbol{x}_0$!) Thus there is now way we can compute the stepsizes in (3).

But the argument above works not only for any orthobasis, but also for any orthobasis using any (valid) inner product. The key innovation in CG is to adaptively choose the step directions $\boldsymbol{d}_k$ and step sizes α_k so that the steps are orthogonal in the $\boldsymbol{H}$ inner product:

$$\langle \boldsymbol{d}_i, \boldsymbol{d}_j \rangle_{\boldsymbol{H}} = \boldsymbol{d}_j^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_i.$$

It is easy to verify that if $\boldsymbol{H}$ is sym+def, then $\langle \cdot, \cdot \rangle_{\boldsymbol{H}}$ is a valid inner product.

So again, suppose that we start at $\boldsymbol{x}_0$ with initial error $\boldsymbol{e}_0$. If $\boldsymbol{d}_0, \ldots, \boldsymbol{d}_{N-1}$ are $\boldsymbol{H}$-orthogonal vectors, then

$$\boldsymbol{e}_0 = \sum_{\ell=0}^{N-1} c_\ell \, \frac{\boldsymbol{d}_\ell}{\|\boldsymbol{d}_\ell\|_{\boldsymbol{H}}},$$

where $\|\boldsymbol{d}_\ell\|_{\boldsymbol{H}}^2 = \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_\ell$, and

$$c_\ell = \frac{\langle \boldsymbol{e}_0, \boldsymbol{d}_\ell \rangle_{\boldsymbol{H}}}{\|\boldsymbol{d}_\ell\|_{\boldsymbol{H}}} = \frac{\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{e}_0}{\|\boldsymbol{d}_\ell\|_{\boldsymbol{H}}}.$$

As we will show below, the iterations below produce a set of $\boldsymbol{H}$-orthogonal step directions $\{\boldsymbol{d}_k\}$ with step sizes $\alpha_k = -c_k / \|\boldsymbol{d}_k\|_{\boldsymbol{H}}$.

Conjugate Gradients

Initialize: $\boldsymbol{x}_0 = $ some guess

$\boldsymbol{r}_0 = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_0$

$\boldsymbol{d}_0 = \boldsymbol{r}_0$

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

$\alpha_k = \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{r}_k / \boldsymbol{d}_k^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}_k$

$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{d}_k$

$\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k \boldsymbol{H}\boldsymbol{d}_k$

$\beta_{k+1} = \boldsymbol{r}_{k+1}^{\mathrm{T}}\boldsymbol{r}_{k+1} / \boldsymbol{r}_k^{\mathrm{T}}\boldsymbol{r}_k$

$\boldsymbol{d}_{k+1} = \boldsymbol{r}_{k+1} + \beta_{k+1}\boldsymbol{d}_k$

end for

We will start our analysis of this iteration by establishing the following two facts:

Fact 1: $\langle \boldsymbol{r}_{k+1}, \boldsymbol{r}_\ell \rangle = \boldsymbol{r}_\ell^{\mathrm{T}}\boldsymbol{r}_{k+1} = 0$ for $\ell = 0, \ldots, k$.
That is, the **residual is orthogonal** to all previous residuals.

Fact 2: $\langle \boldsymbol{d}_{k+1}, \boldsymbol{d}_\ell \rangle_{\boldsymbol{H}} = \boldsymbol{d}_\ell^{\mathrm{T}}\boldsymbol{H}\boldsymbol{d}_{k+1} = 0$ for $\ell = 0, \ldots, k$.
That is, the **direction is $\boldsymbol{H}$-orthogonal** to all previous directions.

We establish these two facts by induction. We start at $k = 1$:

1. $\langle \boldsymbol{r}_1, \boldsymbol{r}_0 \rangle = \boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_1 = 0$, since

$$\begin{aligned} \boldsymbol{r}_1 &= \boldsymbol{r}_0 - \frac{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_0} \boldsymbol{H} \boldsymbol{r}_0 \\ \Rightarrow \quad \boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_1 &= \boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0 - \boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0 \frac{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_0}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_0} \\ &= 0. \end{aligned}$$

2. $\langle \boldsymbol{d}_1, \boldsymbol{d}_0 \rangle_{\boldsymbol{H}} = \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_1 = 0$, since

$$\begin{aligned} \boldsymbol{r}_1 &= \boldsymbol{r}_0 - \alpha_0 \boldsymbol{H} \boldsymbol{d}_0 \\ \Rightarrow \quad \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1 &= \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_0 - \alpha_0 \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0 \\ \Rightarrow \quad \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0 &= -\frac{1}{\alpha_0} \boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1, \end{aligned}$$

since $\boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_0 = 0$. Also,

$$\begin{aligned} \boldsymbol{d}_1 &= \boldsymbol{r}_1 + \frac{\boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0} \boldsymbol{d}_0 \\ \Rightarrow \quad \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_1 &= \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_1 + \frac{\boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0} \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0 \\ &= \frac{\boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0} \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0 - \frac{\boldsymbol{r}_1^{\mathrm{T}} \boldsymbol{r}_1}{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0} \boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0 \\ &= 0. \end{aligned}$$

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

$$\begin{aligned} \langle \boldsymbol{r}_j, \boldsymbol{r}_\ell \rangle &= \boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{r}_j = 0 \quad \forall j, \ell \leq k, \\ \langle \boldsymbol{d}_j, \boldsymbol{d}_\ell \rangle_{\boldsymbol{H}} &= \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_j = 0 \quad \forall j, \ell \leq k. \end{aligned}$$

Then we will also have the following:

1. $\langle \boldsymbol{r}_{k+1}, \boldsymbol{r}_\ell \rangle = \boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{r}_{k+1} = 0$ for all $\ell \leq k$.

   To see this, notice that

$$\boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k = (\boldsymbol{d}_\ell - \beta_\ell \boldsymbol{d}_{\ell-1})^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k \tag{4}$$

$$= \begin{cases} \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k & \ell = k \\ 0 & \ell < k, \end{cases} \tag{5}$$

   where the second step follows directly from the fact that $\langle \boldsymbol{d}_k, \boldsymbol{d}_\ell \rangle_{\boldsymbol{H}} = 0$ for $\ell < k$. As a result

$$\boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{r}_{k+1} = \boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{r}_k - \frac{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k}{\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k} \boldsymbol{r}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k = 0 \quad \text{for all } \ell \leq k. \tag{6}$$

2. $\langle \boldsymbol{d}_{k+1}, \boldsymbol{d}_\ell \rangle_{\boldsymbol{H}} = \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_{k+1} = 0$ for all $\ell \leq k$.

   This follows from the expansion

$$\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_{k+1} = \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k.$$

   Notice that

$$\boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_{k+1} = \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{r}_k - \alpha_k \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k$$

$$\Rightarrow \quad \boldsymbol{r}_i^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k = \begin{cases} \frac{1}{\alpha_k} \boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k & i = k \\ -\frac{1}{\alpha_k} \boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_{k+1} & i = k+1 \\ 0 & i < k. \end{cases} \tag{7}$$

   Then for $\ell = k$

$$\begin{aligned} \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_{k+1} &= -\frac{1}{\alpha_k} \boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k \\ &= \frac{-\boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k} \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k + \frac{\boldsymbol{r}_{k+1}^{\mathrm{T}} \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k} \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k \\ &= 0. \end{aligned}$$

For $\ell < k$,

$$\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_{k+1} = \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_{k+1} + \beta_{k+1} \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k.$$

For the first term

$$\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{r}_{k+1} = 0,$$

since $\boldsymbol{H}\boldsymbol{d}_\ell = \alpha_\ell^{-1}(\boldsymbol{r}_\ell - \boldsymbol{r}_{\ell+1})$ and we have (6); for the second term

$$\beta_{k+1} \boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k = 0,$$

since the $\boldsymbol{d}_0, \boldsymbol{d}_1, \ldots, \boldsymbol{d}_k$ are $\boldsymbol{H}$-orthogonal already.

We have established that the direction $\boldsymbol{d}_k$ that CG moves on iteration k is $\boldsymbol{H}$-orthogonal to all previous directions. Now let's look at the step sizes, where we want to establish that $\alpha_k = -c_k / \|\boldsymbol{d}_k\|_{\boldsymbol{H}} = -\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{e}_0 / \|\boldsymbol{d}_k\|_{\boldsymbol{H}}^2$. Start by noting (5) above, and recall that

$$\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}_k = \boldsymbol{H}(\hat{\boldsymbol{x}} - \boldsymbol{x}_k) = -\boldsymbol{H}\boldsymbol{e}_k.$$

At the first step, we have $\boldsymbol{d}_0 = \boldsymbol{r}_0$, and so

$$\alpha_0 = \frac{\boldsymbol{r}_0^{\mathrm{T}} \boldsymbol{r}_0}{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0} = \frac{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{r}_0}{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0} = \frac{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} (\hat{\boldsymbol{x}} - \boldsymbol{x}_0)}{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0} = \frac{-\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{e}_0}{\boldsymbol{d}_0^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_0}.$$

At subsequent steps, since

$$\boldsymbol{d}_k = \boldsymbol{r}_k + \sum_{i=0}^{k-1} \gamma_i \boldsymbol{r}_i \quad \text{for some} \quad \gamma_i \in \mathbb{R},$$

by Fact 1, we have

$$\boldsymbol{r}_k^{\mathrm{T}} \boldsymbol{r}_k = \boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{r}_k,$$

and so

$$\alpha_k = \frac{\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{r}_k}{\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k} = \frac{-\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \left(\boldsymbol{e}_0 + \sum_{\ell=0}^{k-1} \alpha_\ell \boldsymbol{d}_\ell \right)}{\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k} = \frac{-\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{e}_0}{\boldsymbol{d}_k^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_k}.$$

So finally, this means that for the method of conjugate gradients,

$$\boldsymbol{e}_k = \sum_{\ell=k}^{N-1} \left(\frac{\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{r}_\ell}{\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_\ell} \right) \boldsymbol{d}_\ell, \quad \|\boldsymbol{e}_k\|_{\boldsymbol{H}}^2 = \sum_{\ell=k}^{N-1} \frac{|\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{r}_\ell|^2}{\boldsymbol{d}_\ell^{\mathrm{T}} \boldsymbol{H} \boldsymbol{d}_\ell}.$$

As k increases, the number of (positive) terms in the sum above gets smaller and smaller, until finally

$$\boldsymbol{e}_N = \boldsymbol{0}.$$

Thus **CG is guaranteed to converge exactly in N steps**.

Since each iteration of CG involves a vector-matrix multiply, each of which are $O(N^2)$, and we converge in $O(N)$ iterations, CG solves $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{b}$ in $O(N^3)$ computations in general, the same as other solvers.

But there are two important things to realize:

1. If $\boldsymbol{H}$ 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 $\boldsymbol{H}$.
2. It is often the case that $\|\boldsymbol{e}_k\|_{\boldsymbol{H}}^2$ is acceptably small for relatively modest values of k. This is particularly true if $\boldsymbol{H}$ is well-conditioned. Each iteration (application of $\boldsymbol{H}$) gets us closer, in a measurable way, to the solution.

CG can get an approximate (but still potentially very good) solution using much less computation than solving the system directly.

It also significantly outperforms steepest descent.

Convergence Guarantees

We can actually talk intelligently about how many iterations we need for steepest descent and CG to converge to within a certain precision. Here we present (but do not prove; see the Shewchuk paper for a complete analysis) two "worst case" bounds that depend on the condition number κ of $\boldsymbol{H}$:

$$\kappa = \frac{\lambda_{\max}(\boldsymbol{H})}{\lambda_{\min}(\boldsymbol{H})} = \frac{\text{max eigenvalue}}{\text{min eigenvalue}}.$$

For steepest descent, we will have

$$\|\boldsymbol{e}_k\|_{\boldsymbol{H}} \leq \delta \|\boldsymbol{e}_0\|_{\boldsymbol{H}}$$

in at most[3]

$$k \leq \left\lceil \frac{1}{2} \kappa \log\left(\frac{1}{\delta}\right) \right\rceil$$

iterations.

For CG, we need at most

$$k \leq \left\lceil \frac{1}{2} \sqrt{\kappa} \log\left(\frac{2}{\delta}\right) \right\rceil.$$

[3]These are natural logarithms.

There are nice derivations for both of these bounds in the Shewchuk manuscript mentioned at the beginning of these notes.

Example:

Say the condition number of $\boldsymbol{H}$ 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 in both cases is typically better.