Iterative methods for solving least squares

When $\boldsymbol{A}$ has full column rank, our least squares estimate is

$$\widehat{\boldsymbol{x}} = (\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}.$$

If $\boldsymbol{A}$ is $M \times N$, then constructing $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ costs $O(MN^2)$ computations, and solving the $N \times N$ system $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}$, for example, by computing the inverse of $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, costs $O(N^3)$ computations. A similar complexity is involved in computing the SVD of $\boldsymbol{A}$, so that will not be any cheaper. (Note that for $M \geq N$, the cost of constructing the matrix actually exceeds the cost to solve the system.)

This cost can be prohibitive for even moderately large M and N. But least squares 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" (3D pixels) from about 5 million measurements. In this case, the matrix $\boldsymbol{A}$, which models the mapping from the 3D image $\boldsymbol{x}$ to the set of measurements $\boldsymbol{y}$ induced by the MRI machine, is $M \times N$ where $M = 5 \cdot 10^6$ and $N = 2.1 \cdot 10^6$.

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.

To address this we can consider approaches that return to our formulation of least squares as an optimization program and then solve it by an iterative descent method. Each iteration is simple, requiring one application of $\boldsymbol{A}$ and one application of $\boldsymbol{A}^{\mathrm{T}}$.

If $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$ is "well-conditioned", then these methods can converge in very few iterations. We will be more precise about this later, but here "well-conditioned" roughly means that the ratio of the largest to the smallest singular values of $\boldsymbol{A}$ is not too big. When this works, it can make the cost of solving a least squares problem dramatically smaller — about the cost of a few hundred applications of $\boldsymbol{A}$.

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

In the MRI example above, it turns out that $\boldsymbol{A}$ has a special relationship to the Fourier transform, and because of this it takes about one second to apply $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}$, and a particular iterative method (the conjugate gradients method) converges in about 50 iterations, meaning that the problem can be solved in less than a minute.

To see how this approach works, recall that the least squares estimate is the solution to the optimization problem

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{y}\|_2^2.$$

Note that we can write this equivalently as

$$\underset{\boldsymbol{x}\in\mathbb{R}^N}{\text{minimize}} \ \boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}-2\boldsymbol{x}^{\mathrm{T}}\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}+\boldsymbol{y}^{\mathrm{T}}\boldsymbol{y}.$$

We can ignore terms that do not depend on $\boldsymbol{x}$, and can also rescale the objective function by a constant (for convenience) to obtain

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

We have previously shown that a necessary and sufficient condition for $\widehat{\boldsymbol{x}}$ to be the the minimizer of (1) is to satisfy

$$\boldsymbol{A}^{\mathrm{T}}\boldsymbol{A}\boldsymbol{x}=\boldsymbol{A}^{\mathrm{T}}\boldsymbol{y}.$$

More generally, for any $\boldsymbol{H}$ which is symmetric and positive definite and any vector $\boldsymbol{b}$, we can consider the optimization problem

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

and by the same argument we can show that $\widehat{\boldsymbol{x}}$ is the solution to (2) if and only if

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

What remains is to show how we can actually solve an optimization problem of the form (2) without directly solving the system $\boldsymbol{H}\boldsymbol{x} = \boldsymbol{b}$. Here we will describe iterative methods — most prominently **gradient descent** — that do exactly this.

Gradient 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. We will give a more formal definition later, but for now lets just go with the very informal notion that convexity corresponds to a "bowl shape". One simple way to solve this program is to simply "roll downhill". If we are sitting at a point $\boldsymbol{x}^{(0)}$, then as we mentioned previously in our review of multivariable calculus, f decreases the fastest if we move in the direction of the *negative gradient* $-\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right)$, where we recall that this notation means the gradient of f with respect to $\boldsymbol{x}$ evaluated at $\boldsymbol{x}^{(0)}$.

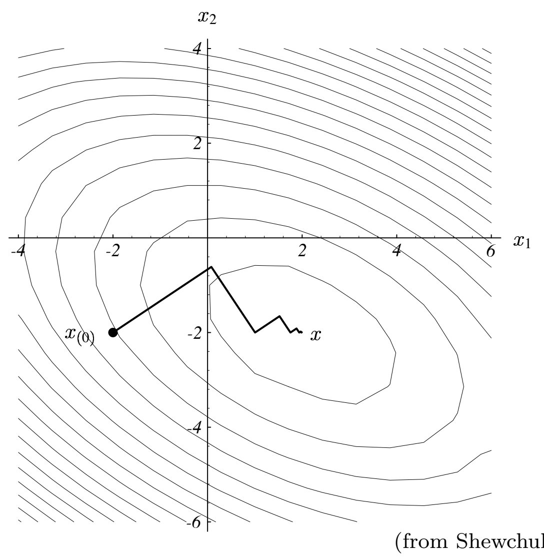
Thus, suppose that from a starting point $\boldsymbol{x}^{(0)}$, we take a step in the direction of the negative gradient, where the step size is controlled by a parameter α_0:

$$\boldsymbol{x}^{(1)} = \boldsymbol{x}^{(0)} - \alpha_0 \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(0)}\right).$$

We then repeat this process:

$$\begin{aligned}\boldsymbol{x}_2 &= \boldsymbol{x}^{(1)} - \alpha_1 \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(1)}\right)\\ &\vdots\\ \boldsymbol{x}^{(k)} &= \boldsymbol{x}^{(k-1)} - \alpha_{k-1} \nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(k-1)}\right),\end{aligned}$$

where, as before, 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 straightforward to compute (you've already done this on the homework):

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

Note that this can be interpreted as the difference between $\boldsymbol{b}$ and $\boldsymbol{H}$ applied to the current iterate $\boldsymbol{x}^{(k)}$. For this reason it is often called the **residual**, denoted by

$$\boldsymbol{r}^{(k)} := \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}^{(k)}.$$

With this notation, the core gradient descent iteration can be written as

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} + \alpha_k \, \boldsymbol{r}^{(k)}.$$

As mentioned above, in this problem there is a nifty way to choose an optimal value for the step size α_k. We want to choose α_k so that $f\left(\boldsymbol{x}^{(k+1)}\right)$ is as small as possible. It is not hard to show that if we think of $f\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{r}^{(k)}\right)$ as a function of α for $\alpha \geq 0$, then f is a (convex) quadratic function. Thus to find the α that minimizes $f\left(\boldsymbol{x}^{(k+1)}\right)$, we can choose the value of α that makes the derivative of this function zero. Specifically, we want

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} f\left(\boldsymbol{x}^{(k)} + \alpha \boldsymbol{r}^{(k)}\right) = 0.$$

By the chain rule,

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

So we need to choose α_k such that

$$\nabla_{\boldsymbol{x}} f\left(\boldsymbol{x}^{(k+1)}\right)^{\mathrm{T}} \boldsymbol{r}^{(k)} = 0,$$

or more concisely

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

So let's do this:

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

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 gradient descent algorithm performs this iteration until $\|\boldsymbol{H}\boldsymbol{x}^{(k)} - \boldsymbol{b}\|_2$ is below some tolerance ϵ:

Gradient Descent, version 1

```
  Initialize: $\boldsymbol{x}^{(0)}$ = some guess, $k = 0$, $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}^{(0)}$.
  while $\|\boldsymbol{r}^{(k)}\|_2 \geq \epsilon$ (not converged) 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

$$\boldsymbol{r}^{(k+1)} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}^{(k+1)} = \boldsymbol{b} - \boldsymbol{H}\left(\boldsymbol{x}^{(k)} + \alpha_k \boldsymbol{r}^{(k)}\right) = \boldsymbol{r}^{(k)} - \alpha_k \boldsymbol{H}\boldsymbol{r}^{(k)}.$$

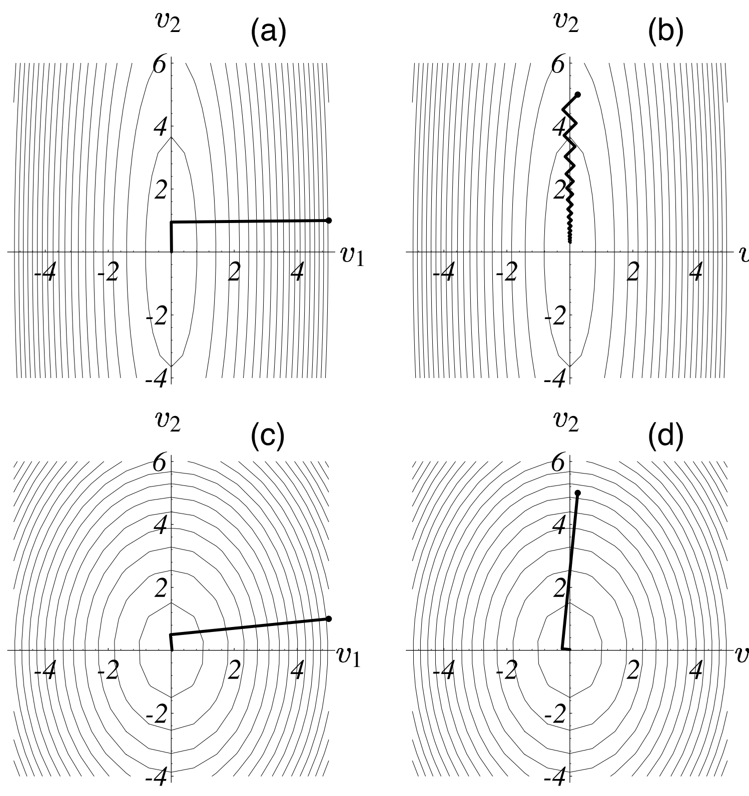
So we can save an application of $\boldsymbol{H}$ by updating the residual rather than recomputing it at each stage.

Gradient Descent, more efficient version 2

```
Initialize: $\boldsymbol{x}^{(0)}$ = some guess, $k = 0$, $\boldsymbol{r}^{(0)} = \boldsymbol{b} - \boldsymbol{H}\boldsymbol{x}^{(0)}$.
while $\|\boldsymbol{r}^{(k)}\|_2 \geq \epsilon$ (not converged) 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
```

Convergence of gradient descent

The effectiveness of gradient descent depends critically on the "conditioning" of $\boldsymbol{H}$ 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, which here corresponds to the case where the ellipses denoting the level sets of our objective function are more eccentric or "squished", and we choose a bad starting point, convergence can take many iterations even in simple cases.

We can make this a bit more precise if we define mathematically what we really mean by the conditioning of $\boldsymbol{H}$. The **condition number** of a matrix $\boldsymbol{H}$, typically denoted $\kappa(\boldsymbol{H})$ is the ratio of the largest to smallest singular values of $\boldsymbol{H}$:

$$\kappa(\boldsymbol{H}) = \frac{\sigma_{\max}(\boldsymbol{H})}{\sigma_{\min}(\boldsymbol{H})}.$$

Note that by the $\sigma_{\min}(\boldsymbol{H})$ we mean the smallest *non-zero* singular value, i.e., σ_R where R is the rank of $\boldsymbol{H}$. The condition number is a natural way of quantifying just how sensitive we are going to be to noise, but it also plays a key role in determining how computationally challenging it will be to solve the least squares problem using iterative methods.

Specifically, later in this course we will provide a more general analysis of gradient descent for general convex functions. This will provide a convergence guarantee that shows how each iteration will reduce the error between the iterates $\boldsymbol{x}^{(k)}$ and the optimal solution $\boldsymbol{x}^\star$. In the context of gradient descent for least squares, this gives us a guarantee of the form

$$\|\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^\star\|_2 \le \left(\frac{\kappa(\boldsymbol{H}) - 1}{\kappa(\boldsymbol{H}) + 1}\right) \|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2. \tag{3}$$

Let's think a bit about what this says. Note that the constant $\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}$ is always less than 1, so it says that each iteration makes some progress. If $\kappa(\boldsymbol{H}) \le 3$, then at each iteration we make *a lot* of progress – cutting the error in half with each iteration. However, if $\kappa(\boldsymbol{H})$ is very large, this constant becomes very close to 1, indicating only minor improvements.

Another way to think about this is to ask, given the above guarantee, if we wanted to ensure that

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2 \le \varepsilon \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2, \tag{4}$$

how many iterations will we need? By applying (3) iteratively, we have

$$\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2 \le \left(\frac{\kappa(\boldsymbol{H}) - 1}{\kappa(\boldsymbol{H}) + 1}\right)^k \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2.$$

To establish (4), we need to show that

$$\left(\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}\right)^k \leq \varepsilon,$$

or equivalently, that

$$k\log\left(\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}\right) \leq \log\varepsilon. \tag{5}$$

Using the bound $\log(x) \geq (x-1)/x$ (for $x > 0$), we have that

$$\log\left(\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}\right) \geq \frac{\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}-1}{\frac{\kappa(\boldsymbol{H})-1}{\kappa(\boldsymbol{H})+1}} = -\frac{2}{\kappa(\boldsymbol{H})-1}.$$

From this we have that if (5) holds, then

$$k\left(-\frac{2}{\kappa(\boldsymbol{H})-1}\right) \leq \log\varepsilon,$$

or more simply, that

$$k \geq \frac{\kappa(\boldsymbol{H})-1}{2}\log(1/\varepsilon).$$

The bottom line here is that if we want the error $\|\boldsymbol{x}^{(k)} - \boldsymbol{x}^\star\|_2$ to be a factor ε smaller than the error of our initial guess $\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^\star\|_2$, then the number of iterations scales logarithmically with $1/\varepsilon$, but linearly with the condition number $\kappa(\boldsymbol{H})$. Thus, for a fixed $\kappa(\boldsymbol{H})$, we can make ε quite small at only a modest cost in the number of iterations, but if $\kappa(\boldsymbol{H})$ grows to be very large, we may need a very large number of iterations. You will see the effect of this in some concrete examples on the next homework.