CS 6241: Numerics for Data Science

Sparse least squares and iterations

David Bindel

2025-01-30

Direct methods for large least squares

Our brief discussion of methods for least squares has so far focused on *dense* factorization methods (normal equations with Cholesky, QR, and SVD). For $A \in \mathbb{R}^{m \times n}$, these methods all require $O(mn^2)$ time to set up the factorization and $O(mn)$ time to solve the system for a particular right hand sice. They also all require $O(mn)$ space. What happens when either m or n is so large as to make this awkward? We consider a few scenarios:

- If m is large but n is not too large (say on the order of a few hundreds, or even 1000-2000), we might still use a standard factorization, but arranged to be efficient on parallel machines. The *tall skinny QR* (TSQR) approach involves breaking the observations into groups and doing a factorization on each group, then recursively combining the factorizations.
- In many cases with large n, the matrix A is often *sparse*: that is, most of the entries of A might be zero. Sometimes, we have that A is *data-sparse*: that is, it has special structure that can be described with far fewer than mn parameters.

In the sparse case, we are sometimes able to use *sparse direct* methods. That is, if A is sparse (most of the elements are zero), we might be able to write an economy QR factorization $A = QR$ where R is also sparse (the matrix Q is usually dense and therefore not saved, though we might be able to store in compressed form as a product of simpler othogonal transformations). Sparse QR isn't always practical, as the R factor is sometimes significantly denser than A. The order of the columns in A can make a huge difference in the sparsity of R, and so we typically would seek a factorization $A\Pi = QR$ where Π is a permutation matrix that reorders the columns.

Frequently, though, sparse direct methods are simply impractical. In this case, we turn to *iterative* methods.

Iterative methods

We started with a discussion of *direct* methods for solving least squares problems based on matrix factorizations. These methods have a well-understood running time, and they produce a solution that is accurate except for roundoff effects. For larger or more complicated problems, though, we turn to *iterative* methods that produce a series of approximation solutions.

We will turn now to iterative methods: gradient and stochastic gradient approaches, Newton and Gauss-Newton, and (block) coordinate descent. We will see additional solver ideas as we move through the class, but these are nicely prototypical examples that illustrate two running themes in the design of numerical methods for optimization.

Fixed point iterations

All our nonlinear solvers (and some of our linear solvers) will be *iterative*. We can write most as *fixed point iterations*

$$x^{k+1} = G(x^k),$$

which we hope will converge to a fixed point, i.e. $x^* = G(x^*)$. We often approach convergence analysis through the *error iteration* relating the error $e^k = x^k - x^*$ at successive steps:

$$e^{k+1} = G(x^* + e^k) - G(x^*).$$

Model-based methods

Most nonquadratic problems are too hard to solve directly. On the other hand, we can *model* hard nonquadratic problems by simpler (possibly linear) problems as a way of building iterative solvers. The most common tactic — but not the only one! — is to approximate the nonlinear function by a linear or quadratic function and apply all the things we know about linear algebra. We will return to this idea in when we discuss Newton-type methods for optimization.

Gradient descent

One very simple iteration is *steepest descent* or *gradient descent*:

$$x^{k+1} = x^k - \alpha_k \nabla \phi(x^k)$$

where α_k is the *step size*, chosen adaptively or with some fixed schedule.

To understand the convergence of this method, consider gradient descent with a fixed step size α for the quadratic model problem

$$\phi(x) = \frac{1}{2} x^T A x + b^T x + c$$

where A is symmetric positive definite. We have computed the gradient for a quadratic before:

$$\nabla\phi(x) = Ax + b,$$

which gives us the iteration equation

$$x_{k+1} = x_k - \alpha(Ax_k + b).$$

Subtracting the fixed point equation

$$x_* = x_* - \alpha(Ax_* + b)$$

yields the error iteration

$$e_{k+1} = (I - \alpha A)e_k.$$

If $\{\lambda_j\}$ are the eigenvalues of A, then the eigenvalues of $I - \alpha A$ are $\{1 - \alpha\lambda_j\}$. The spectral radius of the iteration matrix is thus

$$\max\{|1 - \alpha\lambda_j|\}_j = \max\left(|1 - \alpha\lambda_{\min}|, |1 - \alpha\lambda_{\max}|\right).$$

The iteration converges provided $\alpha < 2/\lambda_{\max}$, and the optimal α is

$$\alpha_* = \frac{2}{\lambda_{\min} + \lambda_{\max}},$$

which leads to the spectral radius

$$1 - \frac{2\lambda_{\min}}{\lambda_{\min} + \lambda_{\max}} = 1 - \frac{2}{1 + \kappa(A)}$$

where $\kappa(A) = \lambda_{\max}/\lambda_{\min}$ is the condition number for the (symmetric positive definite) matrix A. If A is ill-conditioned, then, we are forced to take very small steps to guarantee convergence, and convergence may be heart breakingly slow. We will get to the minimum in the long run — but, then again, in the long run we all die.

The Benefits of Slow Convergence

How steepest descent behaves on a quadratic model is how it behaves generally: if x_* is a strong local minimizer of some general nonlinear ϕ, then gradient descent with a small enough step size will converge locally to x_*. But if $H_\phi(x_*)$ is ill-conditioned, then one has to take small steps, and convergence can be quite slow.

Somewhat surprisingly, sometimes we *want* this slow convergence. To illustrate why, consider the *Landweber iteration*, which is steepest descent iteration applied to linear least squares problems:

$$x^{k+1} = x^k - \alpha_k A^T(Ax^k - b).$$

If we start from the initial guess $x^0 = 0$ and let the step size be a fixed value $\alpha_k = \alpha$, we have the subsequent steps

$$\begin{aligned} x^1 &= \alpha A^T b \\ x^2 &= (I - \alpha A^T A)\alpha A^T b + \alpha A^T b \\ x^3 &= (I - \alpha A^T A)^2 \alpha A^T b + (I - \alpha A^T A)\alpha A^T b + \alpha A^T b \end{aligned}$$

and so forth. That is, each step is a partial sum of a *Neumann series*, which is the matrix generalization of the geometric series

$$\sum_{j=0}^{k} z^j = (1 - z^{k+1})(1 - z)^{-1} \to (1 - z)^{-1} \text{ as } k \to \infty \text{ for } |z| < 1.$$

Using the more concise expression for the partial sums of the Neumann series expansion, we have

$$\begin{aligned} x^{k+1} &= \sum_{j=0}^{k} (I - \alpha A^T A)^j \alpha A^T b \\ &= (I - (I - \alpha A^T A)^{k+1})(\alpha A^T A)^{-1} \alpha A^T b \\ &= (I - (I - \alpha A^T A)^{k+1}) A^\dagger b. \end{aligned}$$

Alternately, we can write the iterates in terms of the singular value decomposition with a filter for regularization:

$$x^{k+1} = V \tilde{\Sigma}^{-1} U^T b, \quad \tilde{\sigma}_j^{-1} = (1 - (1 - \alpha \sigma_j^2)^{k+1}))\sigma_j^{-1}.$$

Hence, rather than running the Landweber iteration to convergence, we typically stop when k is large enough so that the filter is nearly the identity for large singular values, but is small enough so that the influence of the small singular values is suppressed.

Preconditioning stationary iterations

While the slow convergence of iterations like Landweber has some surprising advantages, sometimes it is just a pain. However, we can speed up these transformations by *preconditioning* the problem. That is, rather than applying the Landweber iteration to the problem

$$\min_x \|Ax - b\|^2$$

we instead consider

$$\min_{x = R^{-1} y} \|A\tilde{R}^{-1} y - b\|^2$$

where $\tilde{R}^{-1}$ is easy to apply (e.g. because $\tilde{R}$ might be chosen to be upper triangular) and $A\tilde{R}^{-1}$ has a much smaller condition number than A. In the extreme case where $\tilde{R}$ is the R factor

in a QR factorization of A, we would be able to solve the resulting problem by one step of Landweber with step length one. But we can often do pretty well even when far from the case where $A\tilde{R}^{-1}$ has orthonormal columns. This type of re-scaling of the problem to encourage fast convergence is often called *preconditioning*.

Krylov subspace iterations

We now consider a more general class of iterative methods that *accelerates* the convergence of methods like Landweber. There are many ways to derive these accelerated methods (Krylov subspace methods). We deliberately choose a somewhat unorthodox description that highlights the connections to other accelerated solvers we will encounter later in the class, as well as to our final unit on learning dynamical systems from data.

Let's momentarily consider the case of solving a linear system $Ax = b$, keeping the special case of the normal equations in the back of our minds. A *stationary iteration* has the form

$$Mx_{k+1} = Kx_k + b$$

where $A = M - K$ is sometimes called a *splitting*. The typical way that we analyze such iterations is to subtract the fixed point equation from the iteration, yielding

$$M(x_{k+1} - x_*) = K(x_k - x_*)$$

or $e_{k+1} = (M^{-1}K)e_k = (M^{-1}K)^k e_0$ where $e_k = x_k - x_*$ is the error at step k. The Landweber iteration with fixed step size is an example of a stationary iteration.

Stationary iterations are an example of a linear time-invariant (LTI) dynamical system in discrete time. The dynamics can be described entirely by the eigenvalue decomposition of the iteration matrix $R = M^{-1}K$. Even when the error is guaranteed to decay, general it may decay quickly in some directions (associated with eigenvalues of small magnitude) and slowly in others (associated with eigenvalues with magnitude near 1). We can get rid of the slowly-decaying directions (also called modes) of the error by *filtering* them from the sequence. Unfortunately, the simplest way to construct such a filter in advance involves knowing where the eigenvalues of the iteration matrix lie, at least approximately, and that's often tricky.

An alternative approach is to "learn" the filter from the data by considering all possible filtered sequences, i.e. we consider

$$\tilde{x}_k = \sum_{j=0}^{k} \beta_{jk} x_k$$

for some to-be-determined set of coefficients β. Simplifying slightly by taking $x_0 = 0$, we would have that $\tilde{x}_k$ lies in the $k+1$-dimensional *Krylov subspace*

$$\mathcal{K}_{k+1}(R, b) = \mathrm{sp}\{b, Rb, R^2b, \ldots, R^kb\} = \{p(R)b : p \in \mathcal{P}_k\}$$

where $\mathcal{P}_k$ is the space of polynomials of degree at most k.

It turns out that Krylov subspaces often contain very good approximations to the solution to a linear system. Different Krylov subspace methods choose the "best" approximate solution to $Ax = b$ in a Krylov subspace using different criteria. When A is symmetric and positive definite, we might minimize a quadratic form $\phi(z) = z^T Az/2 - z^T b$ over the subspace; this gives us the *method of conjugate gradients* (CG). Or we might minimize the residual $\|Az - b\|$ over all z in the subspace; this gives us the minimum residual method (MINRES) in the symmetric case, or the generalized minimal residual method (GMRES) in the nonsymmetric case.

Applying CG and MINRES to the normal equations gives an algorithms that, while effective in principle, are not as numerically stable as one might like. One can rearrange these algorithms to get more stable versions specifically for least squares problems; CG in this setting is the basis for LSQR, and MINRES is the basis for LMRES.

Gradient descent with errors

Before we turn to stochastic gradient descent, let us instead look at how to analyze *gradient descent with errors*. In particular, consider the iteration

$$x^{k+1} = x^k - \alpha_k p^k$$

where

$$p^k = \nabla\phi(x^k) + u^k$$

for some error u^k that is "small." As before, let's keep things simple by looking how this iteration behaves for a quadratic model problem with a fixed step size, i.e.

$$x^{k+1} = x^k - \alpha(Ax^k + b + u^k).$$

Subtracting x^* from both sides gives the error iteration

$$e^{k+1} = (I - \alpha A)e^k - \alpha u^k.$$

A little mumbling over the iteration gives us

$$e^{k+1} = (I - \alpha A)^{k+1} e^0 - \alpha \sum_{j=0}^{k} (I - \alpha A)^{k-j} u^j.$$

In order to analyze the second term in this iteration, we need some additional sort of control. In the simplest case, that control might be deterministic. For example, if we can guarantee that $\|u^k\| \leq C\gamma^{-k}$, then we have the bound

$$\left\| \sum_{j=0}^{k} (I - \alpha A)^{k-j} u_j \right\| \leq C\gamma^{-k} \sum_{j=0}^{k} \left(\gamma \|(I - \alpha A)\|\right)^{k-j} \leq \frac{C\gamma^{-k-1}}{1 - \gamma\|I - \alpha A\|}.$$

Hence, we can make the iteration converge with inaccurate gradients, as long as the accuracy improves sufficiently quickly with time.

We will pick up this iteration again next time under the assumption that the errors are random, which is what happens in the *stochastic gradient* method.