Chapter 10

Iterative Linear Solvers

In the previous two chapters, we developed strategies for solving a new class of problems involving minimizing a function $f(\vec{x})$ with or without constraints on $\vec{x}$. In doing so, we relaxed our viewpoint from numerical linear algebra and in particular Gaussian elimination that we must find an *exact* solution to a system of equations and instead turned to iterative schemes that are guaranteed to approximate the minimum of a function better and better as they iterate more and more. Even if we never find the minimum exactly, we know that eventually we will find an $\vec{x}_0$ with $f(\vec{x}_0) \approx \vec{0}$ with arbitrary levels quality, depending on the number of iterations we run.

We now have a reprise of our favorite problem from numerical linear algebra, solving $A\vec{x} = \vec{b}$ for $\vec{x}$, but apply an *iterative* approach rather than expecting to find a solution in closed form. This strategy reveals a new class of linear system solvers that can find reliable approximations of $\vec{x}$ in amazingly few iterations. We already have suggested how to approach this in our discussion of linear algebra, by suggesting that solutions to linear systems are minima of the energy $\|A\vec{x} - \vec{b}\|_2^2$, among others.

Why bother deriving yet another class of linear system solvers? So far, most of our direct approaches require us to represent A as a full $n \times n$ matrix, and algorithms such as LU, QR, or Cholesky factorization all take around $O(n^3)$ time. There are two cases to keep in mind for potential reasons to try iterative schemes:

1. When A is sparse, methods like Gaussian elimination tend to induce *fill*, meaning that even if A contains $O(n)$ nonzero values, intermediate steps of elimination may introduce $O(n^2)$ nonzero values. This property rapidly can cause linear algebra systems to run out of memory. Contrastingly, the algorithms in this chapter require only that you can *apply* A to vectors, which can be done in time proportional to the number of nonzero values in a matrix.

2. We may wish to defeat the $O(n^3)$ runtime of standard matrix factorization techniques. In particular, if an iterative scheme can uncover a fairly accurate solution to $A\vec{x} = \vec{b}$ in a few iterations, runtimes can be decreased considerably.

Also, notice that many of the nonlinear optimization methods we have discussed, in particular those depending on a Newton-like step, require solving a linear system in each iteration! Thus, formulating the fastest possible solver can make a considerable difference when implementing large-scale optimization methods that require one or more linear solves per iteration. In fact, in this case an inaccurate but fast solve to a linear system might be acceptable, since it feeds into a larger iterative technique anyway.

Please note that much of our discussion is due to CITE, although our development can be somewhat shorter given the development in previous chapters.

10.1 Gradient Descent

We will focus our discussion on solving $A\vec{x} = \vec{b}$ where A has three properties:

1. $A \in \mathbb{R}^{n \times n}$ is square
2. A is symmetric, that is, $A^\top = A$
3. A is positive definite, that is, for all $\vec{x} \neq \vec{0}$, $\vec{x}^\top A\vec{x} > 0$

Toward the end of this chapter we will relax these assumptions. For now, notice that we can replace $A\vec{x} = \vec{b}$ with the normal equations $A^\top A\vec{x} = A^\top \vec{b}$ to satisfy these criteria, although as we have discussed this substitution can create numerical conditioning issues.

10.1.1 Deriving the Iterative Scheme

In this case, it is easy to check that solutions of $A\vec{x} = \vec{b}$ are minima of the function $f(\vec{x})$ given by the *quadratic form*

$$f(\vec{x}) \equiv \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top \vec{x} + c$$

for any $c \in \mathbb{R}$. In particular, taking the derivative of f shows

$$\nabla f(\vec{x}) = A\vec{x} - \vec{b},$$

and setting $\nabla f(\vec{x}) = \vec{0}$ yields the desired result.

Rather than solving $\nabla f(\vec{x}) = \vec{0}$ directly as we have done in the past, suppose we apply the gradient descent strategy to this minimization. Recall the basic gradient descent algorithm:

1. Compute the search direction $\vec{d_k} \equiv -\nabla f(\vec{x}_{k-1}) = \vec{b} - A\vec{x}_{k-1}$.
2. Define $\vec{x}_k \equiv \vec{x}_{k-1} + \alpha_k \vec{d_k}$, where α_k is chosen such that $f(\vec{x}_k) < f(\vec{x}_{k-1})$

For a generic function f, deciding on the value of α_k can be a difficult one-dimensional "line search" problem, boiling down to minimizing $f(\vec{x}_{k-1} + \alpha_k \vec{d_k})$ as a function of a single variable $\alpha_k \geq 0$. For our particular choice of the quadratic form $f(\vec{x}) = \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top \vec{x} + c$, however, we can do line search in closed form. In particular, define

$$\begin{aligned}
g(\alpha) &\equiv f(\vec{x} + \alpha\vec{d}) \\
&= \frac{1}{2}(\vec{x} + \alpha\vec{d})^\top A(\vec{x} + \alpha\vec{d}) - \vec{b}^\top(\vec{x} + \alpha\vec{d}) + c \\
&= \frac{1}{2}(\vec{x}^\top A\vec{x} + 2\alpha\vec{x}^\top A\vec{d} + \alpha^2\vec{d}^\top A\vec{d}) - \vec{b}^\top\vec{x} - \alpha\vec{b}^\top\vec{d} + c \text{ by symmetry of } A \\
&= \frac{1}{2}\alpha^2\vec{d}^\top A\vec{d} + \alpha(\vec{x}^\top A\vec{d} - \vec{b}^\top\vec{d}) + \text{const.} \\
\implies \frac{dg}{d\alpha}(\alpha) &= \alpha\vec{d}^\top A\vec{d} + \vec{d}^\top(A\vec{x} - \vec{b})
\end{aligned}$$

Thus, if we wish to minimize g with respect to α, we simply choose

$$\alpha = \frac{\vec{d}^\top(\vec{b} - A\vec{x})}{\vec{d}^\top A\vec{d}}$$

In particular, for gradient descent we chose $\vec{d}_k = \vec{b} - A\vec{x}_k$, so in fact α_k takes a nice form:

$$\alpha_k = \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k}$$

In the end, our formula for line search yields the following iterative gradient descent scheme for solving $A\vec{x} = \vec{b}$ in the symmetric positive definite case:

$$\begin{aligned}
\vec{d}_k &= \vec{b} - A\vec{x}_{k-1} \\
\alpha_k &= \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k} \\
\vec{x}_k &= \vec{x}_{k-1} + \alpha_k \vec{d}_k
\end{aligned}$$

10.1.2 Convergence

By construction our strategy for gradient descent decreases $f(\vec{x}_k)$ as $k \to \infty$. Even so, we have not shown that the algorithm reaches the minimal possible value of f, and we have not been able to characterize how many iterations we should run to reach a reasonable level of confidence that $A\vec{x}_k \approx \vec{b}$.

One simple strategy for understanding the convergence of the gradient descent algorithm for our choice of f is to examine the change in backward error from iteration to iteration.[1] Suppose $\vec{x}^*$ is the solution that we are seeking, that is, $A\vec{x}^* = \vec{b}$. Then, we can study the ratio of backward error from iteration to iteration:

$$R_k \equiv \frac{f(\vec{x}_k) - f(\vec{x}^*)}{f(\vec{x}_{k-1}) - f(\vec{x}^*)}$$

Obviously bounding $R_k < \beta < 1$ for some β shows that gradient descent converges.

For convenience, we can expand $f(\vec{x}_k)$:

$$\begin{aligned}
f(\vec{x}_k) &= f(\vec{x}_{k-1} + \alpha_k \vec{d}_k) \text{ by our iterative scheme} \\
&= \frac{1}{2}(\vec{x}_{k-1} + \alpha_k \vec{d}_k)^\top A(\vec{x}_{k-1} + \alpha_k \vec{d}_k) - \vec{b}^\top(\vec{x}_{k-1} + \alpha_k \vec{d}_k) + c \\
&= f(\vec{x}_{k-1}) + \alpha_k \vec{d}_k^\top A\vec{x}_{k-1} + \frac{1}{2}\alpha_k^2 \vec{d}_k^\top A\vec{d}_k - \alpha_k \vec{b}^\top \vec{d}_k \text{ by definition of } f \\
&= f(\vec{x}_{k-1}) + \alpha_k \vec{d}_k^\top(\vec{b} - \vec{d}_k) + \frac{1}{2}\alpha_k^2 \vec{d}_k^\top A\vec{d}_k - \alpha_k \vec{b}^\top \vec{d}_k \text{ since } \vec{d}_k = \vec{b} - A\vec{x}_{k-1} \\
&= f(\vec{x}_{k-1}) - \alpha_k \vec{d}_k^\top \vec{d}_k + \frac{1}{2}\alpha_k^2 \vec{d}_k^\top A\vec{d}_k
\end{aligned}$$

[1] This argument is presented e.g. in `http://www-personal.umich.edu/~mepelman/teaching/IOE511/Handouts/511notes07-7.pdf`.

$$= f(\vec{x}_{k-1}) - \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k}\vec{d}_k^\top \vec{d}_k + \frac{1}{2}\left(\frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k}\right)^2 \vec{d}_k^\top A\vec{d}_k \text{ by definition of } \alpha_k$$
$$= f(\vec{x}_{k-1}) - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{\vec{d}_k^\top A\vec{d}_k} + \frac{1}{2}\frac{(\vec{d}_k^\top \vec{d}_k)^2}{\vec{d}_k^\top A\vec{d}_k} = f(\vec{x}_{k-1}) - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k}$$

Thus, we can return to our fraction:

$$R_k = \frac{f(\vec{x}_{k-1}) - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k} - f(\vec{x}^*)}{f(\vec{x}_{k-1}) - f(\vec{x}^*)} \text{ by our formula for } f(\vec{x}_k)$$
$$= 1 - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k(f(\vec{x}_{k-1}) - f(\vec{x}^*))}$$

Notice that $A\vec{x}^* = \vec{b}$, so we can write:

$$f(\vec{x}_{k-1}) - f(\vec{x}^*) = \left[\frac{1}{2}\vec{x}_{k-1}^\top A\vec{x}_{k-1} - \vec{b}^\top \vec{x}_{k-1} + c\right] - \left[\frac{1}{2}(\vec{x}^*)^\top \vec{b} - \vec{b}^\top \vec{x}^* + c\right]$$
$$= \frac{1}{2}\vec{x}_{k-1}^\top A\vec{x}_{k-1} - \vec{b}^\top \vec{x}_{k-1} - \frac{1}{2}\vec{b}^\top A^{-1}\vec{b}$$
$$= \frac{1}{2}(A\vec{x}_{k-1} - \vec{b})^\top A^{-1}(A\vec{x}_{k-1} - \vec{b}) \text{ by symmetry of } A$$
$$= \frac{1}{2}\vec{d}_k^\top A^{-1}\vec{d}_k \text{ by definition of } \vec{d}_k$$

Thus,

$$R_k = 1 - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k(f(\vec{x}_{k-1}) - f(\vec{x}^*))}$$
$$= 1 - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{\vec{d}_k^\top A\vec{d}_k \cdot \vec{d}_k^\top A^{-1}\vec{d}_k} \text{ by our latest simplification}$$
$$= 1 - \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k} \cdot \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A^{-1}\vec{d}_k}$$
$$\leq 1 - \left(\min_{\|\vec{d}\|=1} \frac{1}{\vec{d}^\top A\vec{d}}\right)\left(\min_{\|\vec{d}\|=1} \frac{1}{\vec{d}^\top A^{-1}\vec{d}}\right) \text{ since this makes the second term smaller}$$
$$= 1 - \left(\max_{\|\vec{d}\|=1} \vec{d}^\top A\vec{d}\right)^{-1}\left(\max_{\|\vec{d}\|=1} \vec{d}^\top A^{-1}\vec{d}\right)^{-1}$$
$$= 1 - \frac{\sigma_{\min}}{\sigma_{\max}} \text{ where } \sigma_{\min} \text{ and } \sigma_{\max} \text{ are the minimum and maximum singular values of } A$$
$$= 1 - \frac{1}{\operatorname{cond} A}$$

It took a considerable amount of algebra, but we proved an important fact:

Convergence of gradient descent on f depends on the conditioning of A.

That is, the better conditioned A is, the faster gradient descent will converge. Additionally, since $\text{cond}\, A \geq 1$, we know that our gradient descent strategy above converges *unconditionally* to $\vec{x}^*$, although convergence can be slow when A is poorly-conditioned.

Figure NUMBER illustrates behavior of gradient descent for well- and poorly-conditioned matrices. As you can see, gradient descent can struggle to find the minimum of our quadratic function f when the eigenvalues of A have a wide spread.

10.2 Conjugate Gradients

Recall that solving $A\vec{x} = \vec{b}$ for $A \in \mathbb{R}^{n \times n}$ took $O(n^3)$ time. Reexamining the gradient descent strategy above, we see each iteration takes $O(n^2)$ time, since we must compute matrix-vector products between A, $\vec{x}_{k-1}$ and $\vec{d}_k$. Thus, if gradient descent takes more than n iterations, we might as well have applied Gaussian elimination, which will recover the *exact* solution in the same amount of time. Unfortunately, we are not able to show that gradient descent has to take a finite number of iterations, and in fact in poorly-conditioned cases it can take a huge number of iterations to find the minimum.

For this reason, we will design an algorithm that is *guaranteed* to converge in at most n steps, preserving the $O(n^3)$ worst-case timing for solving linear systems. Along the way, we will find that this algorithm in fact exhibits better convergence properties overall, making it a reasonable choice even if we do not run it to completion.

10.2.1 Motivation

Our derivation of the conjugate gradients algorithm is motivated by a fairly straightforward observation. Suppose we knew the solution $\vec{x}^*$ to $A\vec{x}^* = \vec{b}$. Then, we can write our quadratic form f in a different way:

$$\begin{aligned}
f(\vec{x}) &= \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top\vec{x} + c \text{ by definition} \\
&= \frac{1}{2}(\vec{x}-\vec{x}^*)^\top A(\vec{x}-\vec{x}^*) + \vec{x}^\top A\vec{x}^* - \frac{1}{2}(\vec{x}^*)^\top A\vec{x}^* - \vec{b}^\top\vec{x} + c \\
&\qquad \text{by adding and subtracting the same terms} \\
&= \frac{1}{2}(\vec{x}-\vec{x}^*)^\top A(\vec{x}-\vec{x}^*) + \vec{x}^\top\vec{b} - \frac{1}{2}(\vec{x}^*)^\top\vec{b} - \vec{b}^\top\vec{x} + c \text{ since } A\vec{x}^* = \vec{b} \\
&= \frac{1}{2}(\vec{x}-\vec{x}^*)^\top A(\vec{x}-\vec{x}^*) + \text{const. since the } \vec{x}^\top\vec{b} \text{ terms cancel}
\end{aligned}$$

Thus, up to a constant shift f is the same as the product $\frac{1}{2}(\vec{x}-\vec{x}^*)^\top A(\vec{x}-\vec{x}^*)$. Of course, we do not know $\vec{x}^*$, but this observation shows us the nature of f; it is simply measuring the distance from $\vec{x}$ to $\vec{x}^*$ with respect to the "A-norm" $\|\vec{v}\|_A^2 \equiv \vec{v}^\top A\vec{v}$.

In fact, since A is symmetric and positive definite, even if it might be slow to carry out in practice, we know that it can be factorized using the Cholesky strategy as $A = LL^\top$. With this factorization in hand, f takes an even nicer form:

$$f(\vec{x}) = \frac{1}{2}\|L^\top(\vec{x}-\vec{x}^*)\|_2^2 + \text{const.}$$

Since $L^\top$ is an invertible matrix, this norm truly is a distance measure between $\vec{x}$ and $\vec{x}^*$.

Define $\vec{y} \equiv L^\top \vec{x}$ and $\vec{y}^* \equiv L^\top \vec{x}^*$. Then, from this new standpoint, we are minimizing $\bar{f}(\vec{y}) = \|\vec{y} - \vec{y}^*\|_2^2$. Of course, if we truly could get to this point via Cholesky factorization, optimizing $\bar{f}$ would be exceedingly easy, but to derive a scheme for this minimization without L we consider the possibility of minimizing $\bar{f}$ using only line searches derived in §10.1.1.

We make a simple observation about minimizing our simplified function $\bar{f}$ using such a strategy, illustrated in Figure NUMBER:

Proposition 10.1. *Suppose $\{\vec{w}_1, \ldots, \vec{w}_n\}$ are orthogonal in $\mathbb{R}^n$. Then, $\bar{f}$ is minimized in at most n steps by line searching in direction $\vec{w}_1$, then direction $\vec{w}_2$, and so on.*

Proof. Take the columns of $Q \in \mathbb{R}^{n \times n}$ to be the vectors $\vec{w}_i$; Q is an orthogonal matrix. Since Q is orthogonal, we can write $\bar{f}(\vec{y}) = \|\vec{y} - \vec{y}^*\|_2^2 = \|Q^\top \vec{y} - Q^\top \vec{y}^*\|_2^2$; in other words, we rotate so that $\vec{w}_1$ is the first standard basis vector, $\vec{w}_2$ is the second, and so on. If we write $\vec{z} \equiv Q^\top \vec{y}$ and $\vec{z}^* \equiv Q^\top \vec{y}^*$, then clearly after the first iteration we must have $z_1 = z_1^*$, after the second iteration $z_2 = z_2^*$, and so on. After n steps we reach $z_n = z_n^*$, yielding the desired result. □

So, optimizing $\bar{f}$ can always be accomplished using n line searches so long as those searches are in *orthogonal* directions.

All we did to pass from f to $\bar{f}$ is rotate coordinates using $L^\top$. Such a linear transformation takes straight lines to straight lines, so doing a line search on $\bar{f}$ along some vector $\vec{w}$ is equivalent to doing a line search along $(L^\top)^{-1}\vec{w}$ on our original quadratic function f. Conversely, if we do n line searches on f on directions $\vec{v}_i$ such that $L^\top \vec{v}_i \equiv \vec{w}_i$ are orthogonal, then by Proposition 10.1 we must have found $\vec{x}^*$. Notice that asking $\vec{w}_i \cdot \vec{w}_j = 0$ is the same as asking

$$0 = \vec{w}_i \cdot \vec{w}_j = (L^\top \vec{v}_i)^\top (L^\top \vec{v}_j) = \vec{v}_i^\top (LL^\top)\vec{v}_j = \vec{v}_i^\top A \vec{v}_j.$$

We have just argued an important corollary to Proposition 10.1. Define *conjugate* vectors as follows:

Definition 10.1 (A-conjugate vectors)**.** *Two vectors $\vec{v}, \vec{w}$ are* A-conjugate *if $\vec{v}^\top A \vec{w} = 0$.*

Then, based on our discussion we have shown:

Proposition 10.2. *Suppose $\{\vec{v}_1, \ldots, \vec{v}_n\}$ are A-conjugate. Then, f is minimized in at most n steps by line searching in direction $\vec{v}_1$, then direction $\vec{v}_2$, and so on.*

At a high level, the conjugate gradients algorithm simply applies this proposition, generating and searching along A-conjugate directions rather than moving along $-\nabla f$. Notice that this result might appear somewhat counterintuitive: We do *not* necessarily move along the steepest descent direction, but rather ask that our *set* of search directions satisfies a global criterion to make sure we do not repeat work. This setup guarantees convergence in a finite number of iterations and acknowledges the structure of f in terms of $\bar{f}$ discussed above.

Recall that we motivated A-conjugate directions by noting that they are orthogonal after applying $L^\top$ from the factorization $A = LL^\top$. From this standpoint, we are dealing with two dot products: $\vec{x}_i \cdot \vec{x}_j$ and $\vec{y}_i \cdot \vec{y}_j \equiv (L^\top \vec{x}_i) \cdot (L^\top \vec{x}_j) = x_i^\top LL^\top \vec{x}_j = \vec{x}_i^\top A \vec{x}_j$. These two products will figure into our subsequent discussion in equal amounts, so we denote the "A-inner product" as

$$\langle \vec{u}, \vec{v} \rangle_A \equiv (L^\top \vec{u}) \cdot (L^\top \vec{v}) = \vec{u}^\top A \vec{v}.$$

10.2.2 Suboptimality of Gradient Descent

So far, we know that if we can find n A-conjugate search directions, we can solve $A\vec{x} = \vec{b}$ in n steps via line searches along these directions. What remains is to uncover a strategy for finding these directions as efficiently as possible. To do so, we will examine one more property of the gradient descent algorithm that will inspire a more refined approach.

Suppose we are at $\vec{x}_k$ during an iterative line search method on $f(\vec{x})$; we will call the direction of steepest descent of f at $\vec{x}_k$ the *residual* $\vec{r}_k \equiv \vec{b} - A\vec{x}_k$. We may not decide to do a line search along $\vec{r}_k$ as in gradient descent, since the gradient directions are not necessarily A-conjugate. So, generalizing slightly, we will find $\vec{x}_{k+1}$ via line search along a yet-undetermined direction $\vec{v}_{k+1}$.

From our derivation of gradient descent, we should choose $\vec{x}_{k+1} = \vec{x}_k + \alpha_{k+1}\vec{v}_{k+1}$, where α_{k+1} is given by

$$\alpha_{k+1} = \frac{\vec{v}_{k+1}^\top \vec{r}_k}{\vec{v}_{k+1}^\top A\vec{v}_{k+1}}.$$

Applying this expansion of $\vec{x}_{k+1}$, we can write an alternative update formula for the residual:

$$\begin{aligned}
\vec{r}_{k+1} &= \vec{b} - A\vec{x}_{k+1} \\
&= \vec{b} - A(\vec{x}_k + \alpha_{k+1}\vec{v}_{k+1}) \text{ by definition of } \vec{x}_{k+1} \\
&= (\vec{b} - A\vec{x}_k) - \alpha_{k+1}A\vec{v}_{k+1} \\
&= \vec{r}_k - \alpha_{k+1}A\vec{v}_{k+1} \text{ by definition of } \vec{r}_k
\end{aligned}$$

This formula holds regardless of our choice of $\vec{v}_{k+1}$ and can be applied to any iterative line search method.

In the case of gradient descent, however, we chose $\vec{v}_{k+1} \equiv \vec{r}_k$. This choice gives a recurrence relation:

$$\vec{r}_{k+1} = \vec{r}_k - \alpha_{k+1}A\vec{r}_k.$$

This simple formula leads to an instructive proposition:

Proposition 10.3. *When performing gradient descent on f, $\text{span}\{\vec{r}_0, \ldots, \vec{r}_k\} = \text{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^k\vec{r}_0\}$.*

Proof. This statement follows inductively from our formula for $\vec{r}_{k+1}$ above. □

The structure we are uncovering is beginning to look a lot like the Krylov subspace methods mentioned in Chapter 5: This is not a mistake!

Gradient descent gets to $\vec{x}_k$ by moving along $\vec{r}_0$, then $\vec{r}_1$, and so on through $\vec{r}_k$. Thus, in the end we know that the iterate $\vec{x}_k$ of gradient descent on f lies somewhere in the plane $\vec{x}_0 + \text{span}\{\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{k-1}\} = \vec{x}_0 + \text{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, by Proposition 10.3. Unfortunately, it is *not* true that if we run gradient descent, the iterate $\vec{x}_k$ is optimal in this subspace. In other words, in general it can be the case that

$$\vec{x}_k - \vec{x}_0 \neq \underset{\vec{v} \in \text{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}}{\arg\min} f(\vec{x}_0 + \vec{v})$$

Ideally, switching this inequality to an equality would make sure that generating $\vec{x}_{k+1}$ from $\vec{x}_k$ does not "cancel out" any work done during iterations 1 to $k-1$.

If we reexamine our proof of Proposition 10.1 with this fact in mind, we can make an observation suggesting how we might use conjugacy to improve gradient descent. In particular, once z_i switches to z_i^*, it never changes value in a future iteration. In other words, after rotating from $\vec{z}$ to $\vec{x}$ the following proposition holds:

Proposition 10.4. *Take $\vec{x}_k$ to be the k-th iterate of the process from Proposition 10.1 after searching along $\vec{v}_k$. Then,*

$$\vec{x}_k - \vec{x}_0 = \underset{\vec{v} \in span\,\{\vec{v}_1,\ldots,\vec{v}_k\}}{\arg\min} f(\vec{x}_0 + \vec{v})$$

Thus, in the best of all possible worlds, in an attempt to outdo gradient descent we might hope to find A-conjugate directions $\{\vec{v}_1, \ldots, \vec{v}_n\}$ such that $\text{span}\,\{\vec{v}_1, \ldots, \vec{v}_k\} = \text{span}\,\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$ for each k; then our iterative scheme is guaranteed to do no worse than gradient descent during any given iteration. But, greedily we wish to do so without orthogonalization or storing more than a finite number of vectors at a time.

10.2.3 Generating A-Conjugate Directions

Of course, given any set of directions, we can make them A-orthogonal using a method like Gram-Schmidt orthogonalization. Unfortunately, orthogonalizing $\{\vec{r}_0, A\vec{r}_0, \ldots\}$ to find the set of search directions is expensive and would require us to maintain a complete list of directions $\vec{v}_k$; this construction likely would exceed the time and memory requirements even of Gaussian elimination. We will reveal one final observation *about* Gram-Schmidt that makes conjugate gradients tractable by generating conjugate directions without an expensive orthogonalization process.

Ignoring these issues, we might write a "method of conjugate directions" as follows:

$$\textbf{Update search direction (bad Gram-Schmidt step):}\ \vec{v}_k = A^{k-1}\vec{r}_0 - \sum_{i<k} \frac{\langle A^{k-1}\vec{r}_0, \vec{v}_i\rangle_A}{\langle \vec{v}_i, \vec{v}_i\rangle_A}\vec{v}_i$$

$$\textbf{Line search:}\ \alpha_k = \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A\vec{v}_k}$$

$$\textbf{Update estimate:}\ \vec{x}_k = \vec{x}_{k-1} + \alpha_k\vec{v}_k$$

$$\textbf{Update residual:}\ \vec{r}_k = \vec{r}_{k-1} - \alpha_k A\vec{v}_k$$

Here, we compute the k-th search direction $\vec{v}_k$ simply by projecting $\vec{v}_1, \ldots, \vec{v}_{k-1}$ out of the vector $A^{k-1}\vec{r}_0$. This algorithm obviously has the property $\text{span}\,\{\vec{v}_1, \ldots, \vec{v}_k\} = \text{span}\,\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$ suggested in §10.2.2, but has two issues:

1. Similar to power iteration for eigenvectors, the power $A^{k-1}\vec{r}_0$ is likely to look mostly like the first eigenvector of A, making the projection more and more poorly conditioned
2. We have to keep $\vec{v}_1, \ldots, \vec{v}_{k-1}$ around to compute $\vec{v}_k$; thus, each iteration of this algorithm needs more memory and time than the last.

We can fix the first issue in a relatively straightforward manner. In particular, right now we project the previous search directions out of $A^{k-1}\vec{r}_0$, but in reality we can project out previous directions from *any* vector $\vec{w}$ so long as

$$\vec{w} \in \text{span}\,\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\} \backslash \text{span}\,\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-2}\vec{r}_0\},$$

that is, as long as $\vec{w}$ has some component in the new part of the space.

An alternative choice of $\vec{w}$ with this property is the residual $\vec{r}_{k-1}$. This property follows from the residual update $\vec{r}_k = \vec{r}_{k-1} - \alpha_k A\vec{v}_k$; in this expression, we multiply $\vec{v}_k$ by A, introducing the new power of A that we need. This choice also more closely mimics the gradient descent algorithm, which took $\vec{v}_k = \vec{r}_{k-1}$. Thus we can update our algorithm a bit:

$$\textbf{Update search direction (bad Gram-Schmidt on residual):}\ \vec{v}_k = \vec{r}_{k-1} - \sum_{i<k} \frac{\langle \vec{r}_{k-1}, \vec{v}_i \rangle_A}{\langle \vec{v}_i, \vec{v}_i \rangle_A} \vec{v}_i$$

$$\textbf{Line search:}\ \alpha_k = \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A\vec{v}_k}$$

$$\textbf{Update estimate:}\ \vec{x}_k = \vec{x}_{k-1} + \alpha_k \vec{v}_k$$

$$\textbf{Update residual:}\ \vec{r}_k = \vec{r}_{k-1} - \alpha_k A\vec{v}_k$$

Now we do not do arithmetic involving the poorly-conditioned vector $A^{k-1}\vec{r}_0$ but still have the "memory" problem above.

In fact, the surprising observation about the orthogonalizing step above is that most terms in the sum are exactly zero! This amazing observation allows each iteration of conjugate gradients to happen without increasing memory usage. We memorialize this result in a proposition:

Proposition 10.5. *In the "conjugate direction" method above, $\langle \vec{r}_k, \vec{v}_\ell \rangle_A = 0$ for all $\ell < k$.*

Proof. We proceed inductively. There is nothing to prove for the base case $k = 1$, so assume $k > 1$ and that the result holds for all $k' < k$. By the residual update formula, we know:

$$\langle \vec{r}_k, \vec{v}_\ell \rangle_A = \langle \vec{r}_{k-1}, \vec{v}_\ell \rangle_A - \alpha_k \langle A\vec{v}_k, \vec{v}_\ell \rangle_A = \langle \vec{r}_{k-1}, \vec{v}_\ell \rangle_A - \alpha_k \langle \vec{v}_k, A\vec{v}_\ell \rangle_A,$$

where the second equality follows from symmetry of A.

First, suppose $\ell < k-1$. Then the first term of the difference above is zero by induction. Furthermore, by construction $A\vec{v}_\ell \in \text{span}\{\vec{v}_1, \ldots, \vec{v}_{\ell+1}\}$, so since we have constructed our search directions to be A-conjugate we know the second term must be zero as well.

To conclude the proof, we consider the case $\ell = k-1$. Using the residual update formula, we know:

$$A\vec{v}_{k-1} = \frac{1}{\alpha_{k-1}}(\vec{r}_{k-2} - \vec{r}_{k-1})$$

Premultiplying by $\vec{r}_k$ shows:

$$\langle \vec{r}_k, \vec{v}_{k-1} \rangle_A = \frac{1}{\alpha_{k-1}} \vec{r}_k^\top (\vec{r}_{k-2} - \vec{r}_{k-1})$$

The difference $\vec{r}_{k-2} - \vec{r}_{k-1}$ lives in $\text{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, by the residual update formula. Proposition 10.4 shows that $\vec{x}_k$ is optimal in this subspace. Since $\vec{r}_k = -\nabla f(\vec{x}_k)$, this implies that we must have $\vec{r}_k \perp \text{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, since otherwise there would exist a direction in the subspace to move from $\vec{x}_k$ to decrease f. In particular, this shows the inner product above $\langle \vec{r}_k, \vec{v}_{k-1} \rangle_A = 0$, as desired. □

Thus, our proof above shows that we can find a new direction $\vec{v}_k$ as follows:

$$\begin{aligned}\vec{v}_k &= \vec{r}_{k-1} - \sum_{i<k} \frac{\langle \vec{r}_{k-1}, \vec{v}_i \rangle_A}{\langle \vec{v}_i, \vec{v}_i \rangle_A} \vec{v}_i \text{ by the Gram-Schmidt formula} \\ &= \vec{r}_{k-1} - \frac{\langle \vec{r}_{k-1}, \vec{v}_{k-1} \rangle_A}{\langle \vec{v}_{k-1}, \vec{v}_{k-1} \rangle_A} \vec{v}_{k-1} \text{ because the remaining terms vanish}\end{aligned}$$

Since the summation over i disappears, the cost of computing $\vec{v}_k$ has no dependence on k.

10.2.4 Formulating the Conjugate Gradients Algorithm

Now that we have a strategy that yields A-conjugate search directions with relatively little computational effort, we simply apply this strategy to formulate the conjugate gradients algorithm. In particular, suppose $\vec{x}_0$ is an initial guess of the solution to $A\vec{x} = \vec{b}$, and take $\vec{r}_0 \equiv \vec{b} - A\vec{x}_0$. For convenience take $\vec{v}_0 \equiv \vec{0}$. Then, we iteratively update $\vec{x}_{k-1}$ to $\vec{x}_k$ using a series of steps for $k = 1, 2, \ldots$:

$$\begin{aligned}\textbf{Update search direction: } \vec{v}_k &= \vec{r}_{k-1} - \frac{\langle \vec{r}_{k-1}, \vec{v}_{k-1} \rangle_A}{\langle \vec{v}_{k-1}, \vec{v}_{k-1} \rangle_A} \vec{v}_{k-1} \\ \textbf{Line search: } \alpha_k &= \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A \vec{v}_k} \\ \textbf{Update estimate: } \vec{x}_k &= \vec{x}_{k-1} + \alpha_k \vec{v}_k \\ \textbf{Update residual: } \vec{r}_k &= \vec{r}_{k-1} - \alpha_k A \vec{v}_k\end{aligned}$$

This iterative scheme is only a minor adjustment to the gradient descent algorithm but has many desirable properties by construction:

- $f(\vec{x}_k)$ is upper-bounded by that of the k-th iterate of gradient descent
- The algorithm converges to $\vec{x}^*$ in n steps
- At each step, the iterate $\vec{x}_k$ is optimal in the subspace spanned by the first k search directions

In the interests of squeezing maximal numerical quality out of conjugate gradients, we can try to simplify the numerics of the expressions above. For instance, if we plug search direction update into the formula for α_k, by orthogonality we can write:

$$\alpha_k = \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{v}_k^\top A \vec{v}_k}$$

The numerator of this fraction now is guaranteed to be nonnegative without numerical precision issues.

Similarly, we can define a constant β_k to split the search direction update into two steps:

$$\begin{aligned}\beta_k &\equiv -\frac{\langle \vec{r}_{k-1}, \vec{v}_{k-1} \rangle_A}{\langle \vec{v}_{k-1}, \vec{v}_{k-1} \rangle_A} \\ \vec{v}_k &= \vec{r}_{k-1} + \beta_k \vec{v}_{k-1}\end{aligned}$$

We can simplify our formula for β_k:

$$\begin{aligned}
\beta_k &\equiv -\frac{\langle \vec{r}_{k-1}, \vec{v}_{k-1} \rangle_A}{\langle \vec{v}_{k-1}, \vec{v}_{k-1} \rangle_A} \\
&= -\frac{\vec{r}_{k-1} A \vec{v}_{k-1}}{\vec{v}_{k-1}^\top A \vec{v}_{k-1}} \text{ by definition of } \langle \cdot, \cdot \rangle_A \\
&= -\frac{\vec{r}_{k-1}^\top (\vec{r}_{k-2} - \vec{r}_{k-1})}{\alpha_{k-1} \vec{v}_{k-1}^\top A \vec{v}_{k-1}} \text{ since } \vec{r}_k = \vec{r}_{k-1} - \alpha_k A \vec{v}_k \\
&= \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\alpha_{k-1} \vec{v}_{k-1}^\top A \vec{v}_{k-1}} \text{ by a calculation below} \\
&= \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{r}_{k-2}^\top \vec{r}_{k-2}} \text{ by our last formula for } \alpha_k
\end{aligned}$$

This expression reveals that $\beta_k \geq 0$, a property which might not have held after numerical precision issues. We do have one remaining calculation below:

$$\begin{aligned}
\vec{r}_{k-2}^\top \vec{r}_{k-1} &= \vec{r}_{k-2}^\top (\vec{r}_{k-2} - \alpha_{k-1} A \vec{v}_{k-1}) \text{ by our residual update formula} \\
&= \vec{r}_{k-2}^\top \vec{r}_{k-2} - \frac{\vec{r}_{k-2}^\top \vec{r}_{k-2}}{\vec{v}_{k-1}^\top A \vec{v}_{k-1}} \vec{r}_{k-2}^\top A \vec{v}_{k-1} \text{ by our formula for } \alpha_k \\
&= \vec{r}_{k-2}^\top \vec{r}_{k-2} - \frac{\vec{r}_{k-2}^\top \vec{r}_{k-2}}{\vec{v}_{k-1}^\top A \vec{v}_{k-1}} \vec{v}_{k-1}^\top A \vec{v}_{k-1} \text{ by the update for } \vec{v}_k \text{ and } A\text{-conjugacy of the } \vec{v}_k\text{'s} \\
&= 0, \text{ as needed.}
\end{aligned}$$

With these simplifications, we have an alternative version of conjugate gradients:

$$\textbf{Update search direction: } \beta_k = \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{r}_{k-2}^\top \vec{r}_{k-2}}$$

$$\vec{v}_k = \vec{r}_{k-1} + \beta_k \vec{v}_{k-1}$$

$$\textbf{Line search: } \alpha_k = \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{v}_k^\top A \vec{v}_k}$$

$$\textbf{Update estimate: } \vec{x}_k = \vec{x}_{k-1} + \alpha_k \vec{v}_k$$

$$\textbf{Update residual: } \vec{r}_k = \vec{r}_{k-1} - \alpha_k A \vec{v}_k$$

For numerical reasons, occasionally rather than using the update formula for $\vec{r}_k$ it is advisable to use the residual formula $\vec{r}_k = \vec{b} - A\vec{x}_k$. This formula requires an extra matrix-vector multiply but repairs numerical "drift" caused by finite-precision rounding. Notice that there is no need to store a long list of previous residuals or search directions: Conjugate gradients takes a constant amount of space from iteration to iteration.

10.2.5 Convergence and Stopping Conditions

By construction the conjugate gradients (CG) algorithm is guaranteed to converge no more slowly than gradient descent on f, while being no harder to implement and having a number of other

positive properties. A detailed discussion of CG convergence is out of the scope of our discussion, but in general the algorithm behaves best on matrices with evenly-distributed eigenvalues over a small range. One rough estimate paralleling our estimate in §10.1.2 shows that the CG algorithm satisfies:

$$\frac{f(\vec{x}_k) - f(\vec{x}^*)}{f(\vec{x}_0) - f(\vec{x}^*)} \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$

where $\kappa \equiv \text{cond}\, A$. More generally, the number of iterations needed for conjugate gradient to reach a given error value usually can be bounded by a function of $\sqrt{\kappa}$, whereas bounds for convergence of gradient descent are proportional to κ.

We know that conjugate gradients is guaranteed to converge to $\vec{x}^*$ exactly in n steps, but when n is large it may be preferable to stop earlier than that. In fact, the formula for β_k will divide by zero when the residual gets very short, which can cause numerical precision issues near the minimum of f. Thus, in practice CG usually his halted when the ration $\|\vec{r}_k\|/\|\vec{r}_0\|$ is sufficiently small.

10.3 Preconditioning

We now have two powerful iterative schemes for finding solutions to $A\vec{x} = \vec{b}$ when A is symmetric and positive definite: gradient descent and conjugate gradients. Both strategies converge *unconditionally*, meaning that regardless of the initial guess $\vec{x}_0$ with enough iterations we will get arbitrarily close to the true solution $\vec{x}^*$; in fact, conjugate gradients guarantees we will reach $\vec{x}^*$ exactly in a finite number of iterations. Of course, the time taken to reach a solution of $A\vec{x} = \vec{b}$ for both of these methods is directly proportional to the number of iterations needed to reach $\vec{x}^*$ within an acceptable tolerance. Thus, it makes sense to tune a strategy as much as possible to minimize the number of iterations for convergence.

To this end, we notice that we are able to characterize the convergence rates of both algorithms and many more related iterative techniques in terms of the condition number cond A. That is, the small the value of cond A, the less time it should take to solve $A\vec{x} = \vec{b}$. Notice that this situation is somewhat different for Gaussian elimination, which takes the same amount of steps regardless of A; in other words, the conditioning of A affects not only the quality of the output of iterative methods but also the speed at which $\vec{x}^*$ is approached.

Of course, for any invertible matrix P, it is the case that solving $PA\vec{x} = P\vec{b}$ is equivalent to solving $A\vec{x} = \vec{b}$. The trick, however, is that the condition number of PA does not need to be the same as that of A; in the (unachievable) extreme, of course if we took $P = A^{-1}$ we would remove conditioning issues altogether! More generally, suppose $P \approx A^{-1}$. Then, we expect cond $PA \ll$ cond A, and thus it may be advisable to apply P before solving the linear system. In this case, we will call P a *preconditioner*.

While the idea of preconditioning appears attractive, two issues remain:

1. While A may be symmetric and positive definite, the product PA in general will not enjoy these properties.

2. We need to find $P \approx A^{-1}$ that is easier to compute than A^{-1} itself.

We address these issues in the sections below.

10.3.1 CG with Preconditioning

We will focus our discussion on conjugate gradients since it has better convergence properties, although most of our constructions will apply fairly easily to gradient descent as well. From this standpoint, if we look over our constructions in §10.2.1 it is clear that our construction of CG depends strongly on both the symmetry and positive definiteness of A, so running CG on PA usually will not converge out of the box.

Suppose, however, that the preconditioner P is itself symmetric and positive definite. This is a reasonable assumption since A^{-1} must satisfy these properties. Then, we again can write a Cholesky factorization of the inverse $P^{-1} = EE^\top$. We make the following observation:

Proposition 10.6. *The condition number of PA is the same as that of $E^{-1}AE^{-\top}$.*

Proof. We show that PA and $E^{-1}AE^{-\top}$ have the same singular values; the condition number is the ratio of the maximum to the minimum singular value, so this statement is more than sufficient. In particular, it is clear that $E^{-1}AE^{-\top}$ is symmetric and positive definite, so its eigenvectors are its singular values. Thus, suppose $E^{-1}AE^{-\top}\vec{x} = \lambda\vec{x}$. We know $P^{-1} = EE^\top$, so $P = E^{-\top}E^{-1}$., Thus, if we pre-multiply both sides of our eigenvector expression by $E^{-\top}$ we find $PAE^{-\top}\vec{x} = \lambda E^{-\top}\vec{x}$. Defining $\vec{y} \equiv E^{-\top}\vec{x}$ shows $PA\vec{y} = \lambda\vec{y}$. Thus, PA and $E^{-1}AE^{-\top}$ both have full eigenspaces and identical eigenvalues. □

This proposition implies that if we do CG on the symmetric positive definite matrix $E^{-1}AE^{-\top}$, we will receive the same conditioning benefits we would have if we could iterate on PA. As in our proof of Proposition 10.6, we could accomplish our new solve for $\vec{y} = E^\top\vec{x}$ in two steps:

1. Solve $E^{-1}AE^{-\top}\vec{y} = E^{-1}\vec{b}$.
2. Solve $\vec{x} = E^{-\top}\vec{y}$.

Finding E would be integral to this strategy but likely is difficult, but we will prove shortly that it is unnecessary.

Ignoring the computation of E, we could accomplish step 1 using CG as follows:

$$\textbf{Update search direction: } \beta_k = \frac{\vec{r}_{k-1}^\top\vec{r}_{k-1}}{\vec{r}_{k-2}^\top\vec{r}_{k-2}}$$

$$\vec{v}_k = \vec{r}_{k-1} + \beta_k\vec{v}_{k-1}$$

$$\textbf{Line search: } \alpha_k = \frac{\vec{r}_{k-1}^\top\vec{r}_{k-1}}{\vec{v}_k^\top E^{-1}AE^{-\top}\vec{v}_k}$$

$$\textbf{Update estimate: } \vec{y}_k = \vec{y}_{k-1} + \alpha_k\vec{v}_k$$

$$\textbf{Update residual: } \vec{r}_k = \vec{r}_{k-1} - \alpha_k E^{-1}AE^{-\top}\vec{v}_k$$

This iterative scheme will converge according to the conditioning of our matrix $E^{-1}AE^{-\top}$.

Define $\tilde{r}_k \equiv E\vec{r}_k$, $\tilde{v}_k \equiv E^{-\top}\vec{v}_k$, and $\vec{x}_k \equiv E\vec{y}_k$. If we recall the relationship $P = E^{-\top}E^{-1}$, we can rewrite our preconditioned conjugate gradients iteration using these new variables:

$$\textbf{Update search direction: } \beta_k = \frac{\tilde{r}_{k-1}^\top P\tilde{r}_{k-1}}{\tilde{r}_{k-2}^\top P\tilde{r}_{k-2}}$$

$$\tilde{v}_k = P\tilde{r}_{k-1} + \beta_k \tilde{v}_{k-1}$$
$$\textbf{Line search: } \alpha_k = \frac{\vec{r}_{k-1}^\top P \vec{r}_{k-1}}{\tilde{v}_k^\top A \tilde{v}_k}$$
$$\textbf{Update estimate: } \vec{x}_k = \vec{x}_{k-1} + \alpha_k \tilde{v}_k$$
$$\textbf{Update residual: } \tilde{r}_k = \tilde{r}_{k-1} - \alpha_k A \tilde{v}_k$$

This iteration does not depend on the Cholesky factorization of P^{-1} at all, but instead can be carried out using solely applications of P and A. It is easy to see that $\vec{x}_k \to \vec{x}^*$, so in fact this scheme enjoys the benefits of preconditioning without the need to factor the preconditioner.

As a side note, even more effective preconditioning can be carried out by replacing A with PAQ for a second matrix Q, although this second matrix will require additional computations to apply. This example represents a common trade-off: If a preconditioner itself takes too long to apply in a single iteration of CG or another method, it may not be worth the reduced number of iterations.

10.3.2 Common Preconditioners

Finding good preconditioners in practice is as much an art as it is a science. Finding the best approximation P of A^{-1} depends on the structure of A, the particular application at hand, and so on. Even rough approximations, however, can help convergence considerably, so rarely do applications of CG appear that do *not* use a preconditioner.

The best strategy for formulating P often is application-specific, and an interesting engineering approximation problem involves designing and testing assorted P's for the best preconditioner. Two common strategies are below:

- A *diagonal* (or "*Jacobi*") preconditioner simply takes P to be the matrix obtained by inverting diagonal elements of A; that is, P is the diagonal matrix with entries $1/a_{ii}$. This strategy can alleviate nonuniform scaling from row to row, which is a common cause of poor conditioning.

- The *sparse approximate inverse* preconditioner is formulated by solving a subproblem $\min_{P \in S} \|AP - I\|_{\text{Fro}}$, where P is restricted to be in a set S of matrices over which it is less difficult to optimize such an objective. For instance, a common constraint is to prescribe a sparsity pattern for P, e.g. only nonzeros on the diagonal or where A has nonzeros.

- The *incomplete Cholesky* precondtioner factors $A \approx L_* L_*^\top$ and then approximates A^{-1} by solving the appropriate forward- and back-substitution problems. For instance, a popular strategy involves going through the steps of Cholesky factorization but only saving the output in positions (i, j) where $a_{ij} \neq 0$.

- The nonzero values in A can be considered a graph, and removing edges in the graph or grouping nodes may disconnect assorted components; the resulting system is block-diagonal after permuting rows and columns and thus can be solved using a sequence of smaller solves. Such a *domain decomposition* strategy can be effective for linear systems arising from differential equations such as those considered in Chapter NUMBER.

Some preconditioners come with bounds describing changes to the conditioning of A after replacing it with PA, but for the most part these are heuristic strategies that should be tested and refined.

10.4 Other Iterative Schemes

The algorithms we have developed in detail this chapter apply for solving $A\vec{x} = \vec{b}$ when A is square, symmetric, and positive definite. We have focused on this case because it appears so often in practice, but there are cases when A is asymmetric, indefinite, or even rectangular. It is out of the scope of our discussion to derive iterative algorithms in each case, since many require some specialized analysis or advanced development, but we summarize some techniques here from a high-level (CITE EACH):

- *Splitting* methods decompose $A = M - N$ and note that $A\vec{x} = \vec{b}$ is equivalent to $M\vec{x} = N\vec{x} + \vec{b}$. If M is easy to invert, then a fixed-point scheme can be derived by writing $M\vec{x}_k = N\vec{x}_{k-1} + \vec{b}$ (CITE); these techniques are easy to implement but have convergence depending on the spectrum of the matrix $G = M^{-1}N$ and in particular can diverge when the spectral radius of G is greater than one. One popular choice of M is the diagonal of A. Methods such as *successive over-relaxation* (SOR) weight these two terms for better convergence.

- The *conjugate gradient normal equation residual* (CGNR) method simply applies the CG algorithm to the normal equations $A^\top A\vec{x} = A^\top\vec{b}$. This method is simple to implement and guaranteed to converge so long as A is full-rank, but convergence can be slow thanks to poor conditioning of $A^\top A$ as discussed in Chapter NUMBER.

- The *conjugate gradient normal equation error* (CGNE) method similarly solves $AA^\top\vec{y} = \vec{b}$; then the solution of $A\vec{x} = \vec{b}$ is simply $A^\top\vec{y}$.

- Methods such as MINRES and SYMMLQ apply to symmetric but not necessarily positive definite matrices A by replacing our quadratic form $f(\vec{x})$ with $g(\vec{x}) \equiv \|\vec{b} - A\vec{x}\|_2$; this function g is minimized at solutions to $A\vec{x} = \vec{b}$ regardless of the definiteness of A.

- Given the poor conditioning of CGNR and CGNE, the LSQR and LSMR algorithms also minimize $g(\vec{x})$ with fewer assumptions on A, in particular allowing for solution of least-squares systems..

- Generalized methods including GMRES, QMR, BiCG, CGS, and BiCGStab solve $A\vec{x} = \vec{b}$ with the only caveat that A is square and invertible. They optimize similar energies but often have to store more information about previous iterations and may have to factor intermediate matrices to guarantee convergence with such generality.

- Finally, the *Fletcher-Reeves*, *Polak-Ribière*, and other methods return to the more general problem of minimizing a non-quadratic function f, applying conjugate gradient steps to finding new line search directions. Functions f that are well-approximated by quadratics can be minimized very effectively using these strategies, although they do not necessarily make use of the Hessian; for instance, the Fletcher-Reeves method simply replaces the residual in CG iterations with the negative gradient $-\nabla f$. It is possible to characterize convergence of these methods when they are accompanied with sufficiently effective line search strategies.

Many of these algorithms are nearly as easy to implement as CG or gradient descent, and many implementations exist that simply require inputting A and $\vec{b}$. Many of the algorithms listed above require application of both A and $A^\top$, which can be a technical challenge in some cases. As a rule of thumb, the more generalized a method is—that is, the fewer the assumptions a method makes on the structure of the matrix A—the more iterations it is likely to take to compensate for this lack of assumptions. This said, there are no hard-and-fast rules simply by looking at A most successful iterative scheme, although limited theoretical discussion exists comparing the advantages and disadvantages of each of these methods (CITE).

10.5 Problems

- Derive CGNR and/or CGNE
- Derive MINRES
- Derive Fletcher-Reeves
- Slide 13 of `http://math.ntnu.edu.tw/~min/matrix_computation/Ch4_Slide4_CG_2011.pdf`