CHAPTER 11

Iterative Linear Solvers

CONTENTS

11.1 Gradient Descent 208
11.1.1 Gradient Descent for Linear Systems 208
11.1.2 Convergence 209
11.2 Conjugate Gradients 211
11.2.1 Motivation 212
11.2.2 Suboptimality of Gradient Descent 214
11.2.3 Generating A-Conjugate Directions 215
11.2.4 Formulating the Conjugate Gradients Algorithm 217
11.2.5 Convergence and Stopping Conditions 219
11.3 Preconditioning 219
11.3.1 CG with Preconditioning 220
11.3.2 Common Preconditioners 221
11.4 Other Iterative Algorithms 222

IN the previous two chapters, we developed general algorithms for minimizing a function $f(\vec{x})$ with or without constraints on $\vec{x}$. In doing so, we relaxed our viewpoint from numerical linear algebra that we must find an *exact* solution to a system of equations and instead designed iterative methods that successively produce better approximations of the minimizer. Even if we never find the position $\vec{x}^*$ of a local minimum exactly, such methods generate $\vec{x}_k$ with smaller and smaller $f(\vec{x}_k)$, in many cases getting arbitrarily close to the desired optimum.

We now revisit our favorite problem from numerical linear algebra, solving $A\vec{x} = \vec{b}$ for $\vec{x}$, but apply an *iterative* approach rather than seeking a solution in closed form. This adjustment reveals a new class of linear solvers that can find reliable approximations of $\vec{x}$ in remarkably few iterations. To formulate these methods, we will view solving $A\vec{x} = \vec{b}$ not as a system of equations but rather as a minimization problem, e.g., on energies like $\|A\vec{x}-\vec{b}\|_2^2$.

Why bother deriving yet another class of linear solvers? So far, most of our direct solvers require us to represent A as a full $n \times n$ matrix, and algorithms such as LU, QR, and Cholesky factorization all take around $O(n^3)$ time. Two cases motivate the need for iterative methods:

- When A is sparse, Gaussian elimination tends to induce *fill*, meaning that even if A contains $O(n)$ nonzero values, intermediate steps of elimination may fill in the remaining $O(n^2)$ empty positions. Storing a matrix in sparse format dramatically reduces the space it takes in memory, but fill during elimination can rapidly undo these savings. Contrastingly, the algorithms in this chapter require only *application* A to vectors (that is, computation of the product $A\vec{v}$ for any $\vec{v}$), which does not induce fill and can be carried out in time proportional to the number of nonzeros.

- We may wish to defeat the $O(n^3)$ runtime of standard matrix factorization techniques. If an iterative scheme can uncover a fairly, if not completely, accurate solution to $A\vec{x} = \vec{b}$ in a few steps, we may halt the method early in favor of speed over accuracy of the output.

Newton's method and other nonlinear optimization algorithms solve a linear system in each iteration. Formulating the fastest possible solver can make a huge difference in efficiency when implementing these methods for large-scale problems. An inaccurate but fast linear solve may be sufficient, since it feeds into a larger iterative technique anyway.

Although our discussion in this chapter benefits from intuition and formalism developed in previous chapters, our approach to deriving iterative linear methods owes much to the classic extended treatment in [109].

11.1 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. Of course, we always can replace $A\vec{x} = \vec{b}$—at least when A is invertible or overdetermined—with the normal equations $A^\top A\vec{x} = A^\top \vec{b}$ to satisfy these criteria, although as discussed in §5.1, this substitution can create conditioning issues.

11.1.1 Gradient Descent for Linear Systems

Under the restrictions above, 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 $c \in \mathbb{R}$. To see this connection, when A is symmetric, 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.

Solving $\nabla f(\vec{x}) = \vec{0}$ directly amounts to performing Gaussian elimination on A. Instead, suppose we apply gradient descent to this minimization problem. 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 $f(\vec{x}_k) < f(\vec{x}_{k-1})$.
3. Repeat.

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

```
function LINEAR-GRADIENT-DESCENT($A, \vec{b}$)
  $\vec{x} \leftarrow \vec{0}$
  for $k \leftarrow 1, 2, 3, \ldots$
    $\vec{d} \leftarrow \vec{b} - A\vec{x}$                ▷ Search direction is residual
    $\alpha \leftarrow \frac{\|\vec{d}\|_2^2}{\vec{d}^\top A\vec{d}}$               ▷ Line search formula
    $\vec{x} \leftarrow \vec{x} + \alpha\vec{d}$              ▷ Update solution vector $\vec{x}$
```

Figure 11.1 Gradient descent algorithm for solving $A\vec{x} = \vec{b}$ for symmetric and positive definite A, by iteratively decreasing the energy $f(\vec{x}) = \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top\vec{x} + c$.

variable $\alpha_k \geq 0$. For the quadratic form $f(\vec{x}) = \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top\vec{x} + c$, however, we can choose α_k optimally using a closed-form formula. To do so, define

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

With this simplification, to minimize g with respect to α, we solve ${dg}/{d\alpha} = 0$ to find

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

For gradient descent, we chose $\vec{d}_k = \vec{b} - A\vec{x}_k$, so α_k takes the form

$$\alpha_k = \frac{\|\vec{d}_k\|_2^2}{\vec{d}_k^\top A\vec{d}_k}.$$

Since A is positive definite, $\alpha_k > 0$ by definition. This formula leads to the iterative gradient descent algorithm for solving $A\vec{x} = \vec{b}$ shown in Figure 11.1. Unlike generic line search, for this problem the choice of α in each iteration is optimal.

11.1.2 Convergence

By construction, gradient descent decreases $f(\vec{x}_k)$ in each step. Even so, we have not shown that the algorithm approaches the minimum possible $f(\vec{x}_k)$, nor we have 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 way to understand the convergence of the gradient descent algorithm for our choice of f is to examine the change in backward error from iteration to iteration; we will follow the argument in [38] and elsewhere.

Suppose $\vec{x}^*$ satisfies $A\vec{x}^* = \vec{b}$ exactly. Then, the change in backward error in iteration k is given by

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

Bounding $R_k < \beta < 1$ for some fixed β (possibly depending on A) would imply $f(\vec{x}_k) - f(\vec{x}^*) \to 0$ as $k \to \infty$, showing that the gradient descent algorithm converges.

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

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

We can use this formula to find an alternative expression for the backward error R_k:

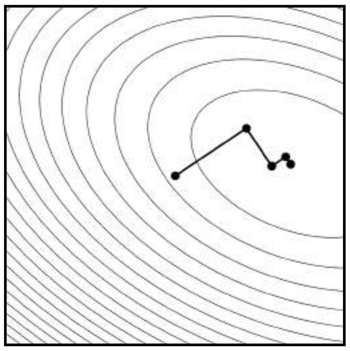
$$\begin{aligned}
R_k &= \frac{f(\vec{x}_{k-1}) - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k} - f(\vec{x}^*)}{f(\vec{x}_{k-1}) - f(\vec{x}^*)} \text{ by the expansion of } f(\vec{x}_k)\\
&= 1 - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{2\vec{d}_k^\top A\vec{d}_k(f(\vec{x}_{k-1}) - f(\vec{x}^*))}.
\end{aligned}$$

To simplify the difference in the denominator, we can use $\vec{x}^* = A^{-1}\vec{b}$ to write:

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

Plugging this expression into our simplified formula for R_k shows:

$$\begin{aligned}
R_k &= 1 - \frac{(\vec{d}_k^\top \vec{d}_k)^2}{\vec{d}_k^\top A\vec{d}_k \cdot \vec{d}_k^\top A^{-1}\vec{d}_k}\\
&= 1 - \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A\vec{d}_k} \cdot \frac{\vec{d}_k^\top \vec{d}_k}{\vec{d}_k^\top A^{-1}\vec{d}_k}
\end{aligned}$$


Well conditioned A

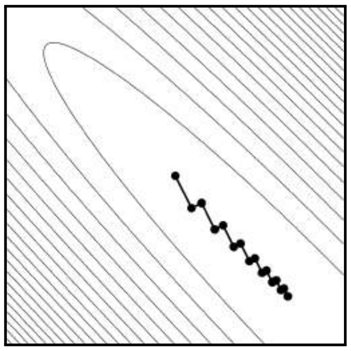

Poorly conditioned A

Figure 11.2 Gradient descent starting from the origin $\vec{0}$ (at the center) on $f(\vec{x}) = \frac{1}{2}\vec{x}^\top A\vec{x} - \vec{b}^\top\vec{x} + c$ for two choices of A. Each figure shows level sets of $f(\vec{x})$ as well as iterates of gradient descent connected by line segments.

$$\begin{aligned}
&\leq 1 - \left(\min_{\|\vec{d}\|=1} \frac{1}{\vec{d}^\top A\vec{d}}\right)\left(\min_{\|\vec{d}\|=1} \frac{1}{\vec{d}^\top A^{-1}\vec{d}}\right) \text{ since this makes the second term smaller}\\
&= 1 - \left(\max_{\|\vec{d}\|=1} \vec{d}^\top A\vec{d}\right)^{-1}\left(\max_{\|\vec{d}\|=1} \vec{d}^\top A^{-1}\vec{d}\right)^{-1}\\
&= 1 - \frac{\sigma_{\min}}{\sigma_{\max}} \text{ where } \sigma_{\min}, \sigma_{\max} \text{ are the minimum/maximum singular values of } A\\
&= 1 - \frac{1}{\operatorname{cond} A}.
\end{aligned}$$

Here, we assume the condition number $\operatorname{cond} A$ is computed with respect to the two-norm of 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 $\operatorname{cond} A \geq 1$, we know that gradient descent converges *unconditionally* to $\vec{x}^*$, although convergence can be slow when A is poorly conditioned.

Figure 11.2 illustrates the behavior of gradient descent for well and poorly conditioned matrices A. When the eigenvalues of A have a wide spread, A is poorly conditioned and gradient descent struggles to find the minimum of our quadratic function f, zig-zagging along the energy landscape.

11.2 CONJUGATE GRADIENTS

Solving $A\vec{x} = \vec{b}$ for dense $A \in \mathbb{R}^{n\times n}$ takes $O(n^3)$ time using Gaussian elimination. Reexamining gradient descent from §11.1.1 above, we see that in the dense case each iteration takes $O(n^2)$ time, since we must compute matrix-vector products between A and $\vec{x}_{k-1}, \vec{d}_k$. So, if gradient descent takes more than n iterations, from a timing standpoint we might as well have used Gaussian elimination, which would have recovered the *exact* solution in the same amount of time. Unfortunately, gradient descent may never reach the exact solution $\vec{x}^*$ in a finite number of iterations, and in poorly conditioned cases it can take a huge number of iterations to approximate $\vec{x}^*$ well.

For this reason, we will design the *conjugate gradients* (CG) algorithm, which is *guaranteed* to converge in at most n steps, preserving $O(n^3)$ worst-case timing for solving linear systems. We also will find that this algorithm exhibits better convergence properties overall, often making it preferable to gradient descent even if we do not run it to completion.

11.2.1 Motivation

Our derivation of the conjugate gradients algorithm is motivated by writing the energy functional $f(\vec{x})$ in an alternative form. Suppose we knew the solution $\vec{x}^*$ to $A\vec{x}^* = \vec{b}$. Then, we could write:

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

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

Since A is symmetric and positive definite, even if it might be slow to compute algorithmically, we know from §4.2.1 that A admits a Cholesky factorization $A = LL^\top$. With this factorization, f takes a nicer form:

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

From this form of $f(\vec{x})$, we now know that the A-norm truly measures a distance between $\vec{x}$ and $\vec{x}^*$.

Define $\vec{y} \equiv L^\top\vec{x}$ and $\vec{y}^* \equiv L^\top\vec{x}^*$. After this change of variables, we are minimizing $\bar{f}(\vec{y}) \equiv \|\vec{y}-\vec{y}^*\|_2^2$. Optimizing $\bar{f}$ would be easy if we knew L and $\vec{y}^*$ (take $\vec{y} = \vec{y}^*$), but to eventually remove the need for L we consider the possibility of minimizing $\bar{f}$ using only line searches derived in §11.1.1; from this point on, we will assume that we use the optimal step α for this search rather than any other procedure.

We make an observation about minimizing our simplified function $\bar{f}$ using line searches, illustrated in Figure 11.3:

Proposition 11.1. 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 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. □

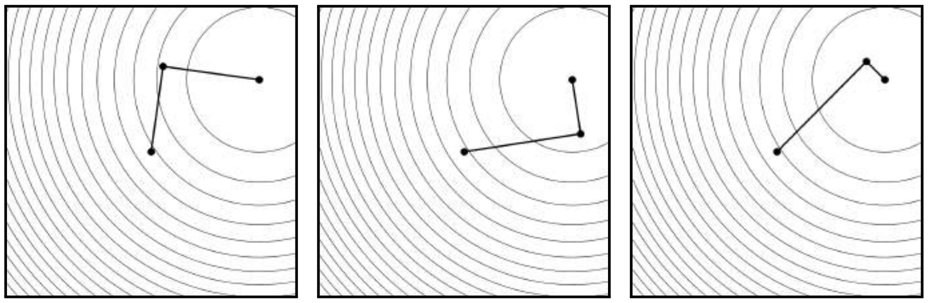


Figure 11.3 Searching along *any* two orthogonal directions minimizes $\bar{f}(\vec{y}) = \|\vec{y} - \vec{y}^*\|_2^2$ over $\vec{y} \in \mathbb{R}^2$. Each example in this figure has the same starting point but searches along a different pair of orthogonal directions; in the end they all reach the same optimal point.

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

All we did to pass from f to $\bar{f}$ is change coordinates using $L^\top$. Linear transformations take straight lines to straight lines, so line search on $\bar{f}$ along some vector $\vec{w}$ is equivalent to line search along $(L^\top)^{-1}\vec{w}$ on the original quadratic function f. Conversely, if we do n line searches on f in directions $\vec{v}_i$ such that $L^\top \vec{v}_i \equiv \vec{w}_i$ are orthogonal, then by Proposition 11.1 we must have found $\vec{x}^*$. The condition $\vec{w}_i \cdot \vec{w}_j = 0$ can be simplified:

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

We have just argued a corollary to Proposition 11.1. Define *conjugate* vectors as follows:

Definition 11.1 (A-conjugate vectors)**.** Two vectors $\vec{v}, \vec{w}$ are A-*conjugate* if $\vec{v}^\top A \vec{w} = 0$.

Then, we have shown how to use Proposition 11.1 to optimize f rather than $\bar{f}$:

Proposition 11.2. Suppose $\{\vec{v}_1, \ldots, \vec{v}_n\}$ are A-conjugate. Then, f is minimized in at most n steps by line search in direction $\vec{v}_1$, then direction $\vec{v}_2$, and so on.

Inspired by this proposition, the conjugate gradients algorithm generates and searches along A-conjugate directions rather than moving along $-\nabla f$. This change might appear somewhat counterintuitive: Conjugate gradients does not necessarily move along the steepest descent direction in each iteration, but rather constructs a *set* of search directions satisfying a global criterion to avoid repeating work. This setup guarantees convergence in a finite number of iterations and acknowledges the structure of f in terms of $\bar{f}$ discussed above.

We motivated the use of A-conjugate directions by their orthogonality 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, so for clarity we will denote the "A-inner product" as

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

11.2.2 Suboptimality of Gradient Descent

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

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 in §11.1.1, even if $\vec{v}_{k+1} \neq \vec{r}_k$, we should choose $\vec{x}_{k+1} = \vec{x}_k + \alpha_{k+1}\vec{v}_{k+1}$, where

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

Applying this expansion of $\vec{x}_{k+1}$, we can write an 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 on f.

In the case of gradient descent, we chose $\vec{v}_{k+1} \equiv \vec{r}_k$, giving a recurrence relation $\vec{r}_{k+1} = \vec{r}_k - \alpha_{k+1}A\vec{r}_k$. This formula inspires an instructive proposition:

Proposition 11.3. When performing gradient descent on f, $\operatorname{span}\{\vec{r}_0, \ldots, \vec{r}_k\} = \operatorname{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^k\vec{r}_0\}$.

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 6: This is not a coincidence!

Gradient descent gets to $\vec{x}_k$ by moving along $\vec{r}_0$, then $\vec{r}_1$, and so on through $\vec{r}_k$. In the end we know that the iterate $\vec{x}_k$ of gradient descent on f lies somewhere in the plane $\vec{x}_0 + \operatorname{span}\{\vec{r}_0, \vec{r}_1, \ldots, \vec{r}_{k-1}\} = \vec{x}_0 + \operatorname{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, by Proposition 11.3. Unfortunately, it is *not* true that if we run gradient descent, the iterate $\vec{x}_k$ is optimal in this subspace. In other words, it can be the case that

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

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 11.1 from this perspective, we can make an observation suggesting how we might use conjugacy to improve gradient descent. Once z_i switches to z_i^*, it never changes in a future iteration. After rotating back from $\vec{z}$ to $\vec{x}$ the following proposition holds:

Proposition 11.4. Take $\vec{x}_k$ to be the k-th iterate of the process from Proposition 11.1 after searching along $\vec{v}_k$. Then,

$$\vec{x}_k - \vec{x}_0 = \operatorname*{arg\,min}_{\vec{v} \in \text{span}\,\{\vec{v}_1, \ldots, \vec{v}_k\}} f(\vec{x}_0 + \vec{v}).$$

In the best of all possible worlds and 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. By the previous two propositions, the resulting iterative scheme would be guaranteed to do no worse than gradient descent even if it is halted early. But, we wish to do so without incurring significant memory demand or computation time. Amazingly, the conjugate gradient algorithm satisfies all these criteria.

11.2.3 Generating A-Conjugate Directions

Given any set of directions spanning $\mathbb{R}^n$, we can make them A-orthogonal using Gram-Schmidt orthogonalization. Explicitly orthogonalizing $\{\vec{r}_0, A\vec{r}_0, A^2\vec{r}_0, \ldots\}$ to find the set of search directions, however, is expensive and would require us to maintain a complete list of directions in memory; this construction likely would exceed the time and memory requirements even of Gaussian elimination. Alternatively, we will reveal one final observation *about* Gram-Schmidt that makes conjugate gradients tractable by generating conjugate directions without an expensive orthogonalization process.

To start, we might write a "method of conjugate directions" using the following iterations:

```
$\vec{v}_k \leftarrow A^{k-1}\vec{r}_0 - \sum_{i<k} \frac{\langle A^{k-1}\vec{r}_0, \vec{v}_i\rangle_A}{\langle \vec{v}_i, \vec{v}_i\rangle_A}\vec{v}_i$    ▷ Explicit Gram-Schmidt
$\alpha_k \leftarrow \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A\vec{v}_k}$    ▷ Line search
$\vec{x}_k \leftarrow \vec{x}_{k-1} + \alpha_k\vec{v}_k$    ▷ Update estimate
$\vec{r}_k \leftarrow \vec{r}_{k-1} - \alpha_k A\vec{v}_k$    ▷ Update residual
```

Here, we compute the k-th search direction $\vec{v}_k$ by projecting $\vec{v}_1, \ldots, \vec{v}_{k-1}$ out of the vector $A^{k-1}\vec{r}_0$ using the Gram-Schmidt algorithm. This algorithm 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 §11.2.2, but it 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 projection poorly conditioned when k is large.

2. We have to store $\vec{v}_1, \ldots, \vec{v}_{k-1}$ to compute $\vec{v}_k$, so each iteration needs more memory and time than the last.

We can fix the first issue in a relatively straightforward manner. 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 $\vec{w}$ in this span is the residual $\vec{r}_{k-1}$. We can check this using 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 $\vec{v}_k = \vec{r}_{k-1}$. We can update our algorithm to use this improved choice:

```
$\vec{v}_k \leftarrow \vec{r}_{k-1} - \sum_{i<k} \frac{\langle \vec{r}_{k-1}, \vec{v}_i \rangle_A}{\langle \vec{v}_i, \vec{v}_i \rangle_A} \vec{v}_i$    ▷ Gram-Schmidt on residual
$\alpha_k \leftarrow \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A \vec{v}_k}$    ▷ Line search
$\vec{x}_k \leftarrow \vec{x}_{k-1} + \alpha_k \vec{v}_k$    ▷ Update estimate
$\vec{r}_k \leftarrow \vec{r}_{k-1} - \alpha_k A \vec{v}_k$    ▷ Update residual
```

Now we do not do arithmetic with the poorly conditioned vector $A^{k-1}\vec{r}_0$ but still have the "memory" problem above since the sum in the first step is over $k-1$ vectors.

A surprising observation about the residual Gram-Schmidt projection above is that most terms in the sum are exactly zero! This observation allows each iteration of conjugate gradients to be carried out without increasing memory requirements. We memorialize this result in a proposition:

Proposition 11.5. In the second "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,

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

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 $A\vec{v}_\ell \in \operatorname{span}\{\vec{v}_1, \ldots, \vec{v}_{\ell+1}\}$, so since we have constructed our search directions to be A-conjugate, the second term must be zero as well.

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

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

Pre-multiplying by $\vec{r}_k^\top$ 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 $\vec{r}_{k-2} - \vec{r}_{k-1}$ is in the subspace $\operatorname{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, by the residual update formula. Proposition 11.4 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 $\vec{r}_k \perp \operatorname{span}\{\vec{r}_0, A\vec{r}_0, \ldots, A^{k-1}\vec{r}_0\}$, 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. □

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.

11.2.4 Formulating the Conjugate Gradients Algorithm

Now that we can obtain A-conjugate search directions with relatively little computational effort, we apply this strategy to formulate the conjugate gradients algorithm, with full pseudocode in Figure 11.4(a):

```
$\vec{v}_k \leftarrow \vec{r}_{k-1} - \frac{\langle \vec{r}_{k-1}, \vec{v}_{k-1}\rangle_A}{\langle \vec{v}_{k-1}, \vec{v}_{k-1}\rangle_A}\vec{v}_{k-1}$    ▷ Update search direction
$\alpha_k \leftarrow \frac{\vec{v}_k^\top \vec{r}_{k-1}}{\vec{v}_k^\top A\vec{v}_k}$    ▷ Line search
$\vec{x}_k \leftarrow \vec{x}_{k-1} + \alpha_k \vec{v}_k$    ▷ Update estimate
$\vec{r}_k \leftarrow \vec{r}_{k-1} - \alpha_k A\vec{v}_k$    ▷ Update residual
```

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 at most n steps, as illustrated in Figure 11.5.
- 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 simplify the numerics of the formulation in Figure 11.4(a). For instance, if we plug the search direction update into the formula for α_k, by orthogonality we know

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

The numerator of this fraction now is guaranteed to be nonnegative even when using finite-precision arithmetic.

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 the formula for β_k:

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

This expression guarantees that $\beta_k \geq 0$, a property that might not have held after rounding using the original formula. We have one remaining calculation below:

$$\vec{r}_{k-2}^\top \vec{r}_{k-1} = \vec{r}_{k-2}^\top(\vec{r}_{k-2} - \alpha_{k-1}A\vec{v}_{k-1}) \text{ by the residual update formula}$$

```
function CONJUGATE-GRAD-1($A, \vec{b}, \vec{x}_0$)
    $\vec{x} \leftarrow \vec{x}_0$
    $\vec{r} \leftarrow \vec{b} - A\vec{x}$
    $\vec{v} \leftarrow \vec{r}$
    for $k \leftarrow 1, 2, 3, \ldots$
        $\alpha \leftarrow \frac{\vec{v}^\top \vec{r}}{\vec{v}^\top A\vec{v}}$          ▷ Line search
        $\vec{x} \leftarrow \vec{x} + \alpha\vec{v}$          ▷ Update estimate
        $\vec{r} \leftarrow \vec{r} - \alpha A\vec{v}$          ▷ Update residual
        if $\|\vec{r}\|_2^2 < \varepsilon\|\vec{r}_0\|_2^2$ then
            return $x^* = \vec{x}$
        $\vec{v} \leftarrow \vec{r} - \frac{\langle \vec{r}, \vec{v}\rangle_A}{\langle \vec{v}, \vec{v}\rangle_A}\vec{v}$          ▷ Search direction
```

```
function CONJUGATE-GRAD-2($A, \vec{b}, \vec{x}_0$)
    $\vec{x} \leftarrow \vec{x}_0$
    $\vec{r} \leftarrow \vec{b} - A\vec{x}$
    $\vec{v} \leftarrow \vec{r}$
    $\beta \leftarrow 0$
    for $k \leftarrow 1, 2, 3, \ldots$
        $\vec{v} \leftarrow \vec{r} + \beta\vec{v}$          ▷ Search direction
        $\alpha \leftarrow \frac{\|\vec{r}\|_2^2}{\vec{v}^\top A\vec{v}}$          ▷ Line search
        $\vec{x} \leftarrow \vec{x} + \alpha\vec{v}$          ▷ Update estimate
        $\vec{r}_{\text{old}} \leftarrow \vec{r}$          ▷ Save old residual
        $\vec{r} \leftarrow \vec{r} - \alpha A\vec{v}$          ▷ Update residual
        if $\|\vec{r}\|_2^2 < \varepsilon\|\vec{r}_0\|_2^2$ then
            return $x^* = \vec{x}$
        $\beta \leftarrow \|\vec{r}\|_2^2 / \|\vec{r}_{\text{old}}\|_2^2$          ▷ Direction step
```

Figure 11.4 Two equivalent formulations of the conjugate gradients algorithm for solving $A\vec{x} = \vec{b}$ when A is symmetric and positive definite. The initial guess $\vec{x}_0$ can be $\vec{0}$ in the absence of a better estimate.

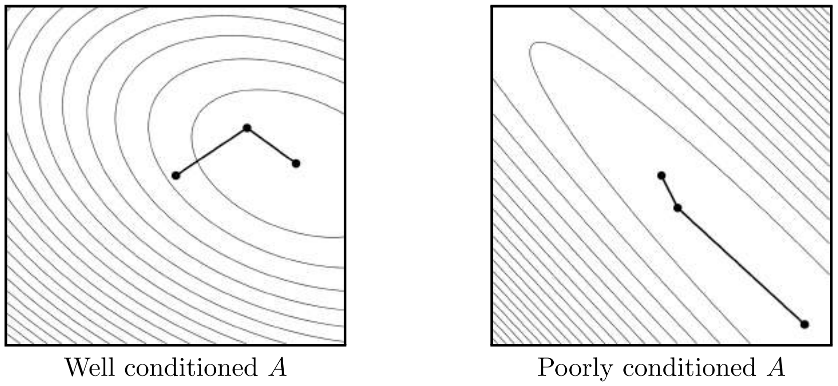


Figure 11.5 The conjugate gradients algorithm solves both linear systems in Figure 11.2 in two steps.

$$= \vec{r}_{k-2}^\top \vec{r}_{k-2} - \frac{\vec{r}_{k-2}^\top \vec{r}_{k-2}}{\vec{v}_{k-1}^\top A \vec{v}_{k-1}} \vec{r}_{k-2}^\top A \vec{v}_{k-1} \text{ by our formula for } \alpha_k$$

$$= \vec{r}_{k-2}^\top \vec{r}_{k-2} - \frac{\vec{r}_{k-2}^\top \vec{r}_{k-2}}{\vec{v}_{k-1}^\top A \vec{v}_{k-1}} \vec{v}_{k-1}^\top A \vec{v}_{k-1}$$

$$\text{by the update for } \vec{v}_k \text{ and } A\text{-conjugacy of the } \vec{v}_k\text{'s}$$

$$= 0, \text{ as needed.}$$

Our new observations about the iterates of CG provide an alternative but equivalent formulation that can have better numerical properties; it is shown in Figure 11.4(b). Also 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 requires an extra matrix-vector multiply but repairs numerical "drift" caused by finite-precision rounding. 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.

11.2.5 Convergence and Stopping Conditions

By construction, the conjugate gradients (CG) algorithm is guaranteed to converge as fast as gradient descent on f, while being no harder to implement and having a number of other favorable properties. A detailed discussion of CG convergence is out of the scope of our treatment, but in general the algorithm behaves best on matrices with eigenvalues evenly distributed over a small range.

One rough bound paralleling the estimate in §11.1.2 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 $\kappa \equiv \operatorname{cond} A$. Broadly speaking, the number of iterations needed for conjugate gradient to reach a given error level usually can be bounded by a function of $\sqrt{\kappa}$, whereas bounds for convergence of gradient descent are proportional to κ.

Conjugate gradients is guaranteed to converge to $\vec{x}^*$ exactly in n steps—m steps if A has $m < n$ unique eigenvalues—but when n is large it may be preferable to stop earlier. 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 is halted when the ratio $\|\vec{r}_k\|/\|\vec{r}_0\|$ is sufficiently small.

11.3 PRECONDITIONING

We now have two powerful iterative algorithms for solving $A\vec{x} = \vec{b}$ when A is symmetric and positive definite: gradient descent and conjugate gradients. Both converge *unconditionally*, meaning that regardless of the initial guess $\vec{x}_0$, with enough iterations they will get arbitrarily close to the true solution $\vec{x}^*$; conjugate gradients reaches $\vec{x}^*$ exactly in a finite number of iterations. The "clock time" taken to solve $A\vec{x} = \vec{b}$ for both of these methods is proportional to the number of iterations needed to reach $\vec{x}^*$ within an acceptable tolerance, so it makes sense to minimize the number of iterations until convergence.

We characterized the convergence rates of both algorithms in terms of the condition number $\operatorname{cond} A$. The smaller the value of $\operatorname{cond} A$, the less time it should take to solve $A\vec{x} = \vec{b}$. This situation contrasts with Gaussian elimination, which takes the same number of steps regardless of A; what is new here is that 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.

For any invertible matrix P, solving $PA\vec{x} = P\vec{b}$ is equivalent to solving $A\vec{x} = \vec{b}$. The condition number of PA, however, does *not* need to be the same as that of A. In the extreme, if we took $P = A^{-1}$, then conditioning issues would be removed altogether! More generally, suppose $P \approx A^{-1}$. Then, we expect $\operatorname{cond} PA \ll \operatorname{cond} A$, making it advisable to apply P before solving the linear system using iterative methods. 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.

11.3.1 CG with Preconditioning

We will focus our discussion of preconditioning on conjugate gradients since it has better convergence properties than gradient descent, although most of our constructions can be paralleled to precondition other iterative linear methods.

Starting from the steps in §11.2.1, the construction of CG fundamentally depended on both the symmetry and positive definiteness of A. Hence, running CG on PA usually will not converge, since it may violate these assumptions. Suppose, however, that the preconditioner P is itself symmetric and positive definite. This is a reasonable assumption since the inverse A^{-1} of a symmetric, positive definite matrix A is itself symmetric and positive definite. Under this assumption, we can write a Cholesky factorization of the inverse $P^{-1} = EE^\top$. Then, $E^{-1}AE^{-\top} \approx E^{-1}P^{-1}E^{-\top} = E^{-1}EE^\top E^{-\top} = I_{n\times n}$. In words, we expect $E^{-1}AE^{-\top}$ to be well-conditioned when PA is well-conditioned. This intuition is partially confirmed by the following observation:

Proposition 11.6. PA and $E^{-1}AE^{-\top}$ have the same eigenvalues.

Proof. Suppose $E^{-1}AE^{-\top}\vec{x} = \lambda\vec{x}$; notice the vectors $\vec{x}$ span $\mathbb{R}^n$ because $E^{-1}AE^{-\top}$ is symmetric. By construction, $P^{-1} = EE^\top$, so $P = E^{-\top}E^{-1}$. If we pre-multiply both sides of the 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 $PA\vec{y} = \lambda\vec{y}$. Hence, each eigenvector $\vec{x}$ of $E^{-1}AE^{-\top}$ provides a corresponding eigenvector $\vec{y}$ of PA, showing that 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 similar conditioning benefits enjoyed by PA. Imitating the construction in Proposition 11.6 above, we can carry out our new solve for $\vec{y} = E^\top\vec{x}$ in two steps:

1. Solve $E^{-1}AE^{-\top}\vec{y} = E^{-1}\vec{b}$ for $\vec{y}$ using the CG algorithm.

2. Multiply to find $\vec{x} = E^{-\top}\vec{y}$.

Evaluating E and its inverse would be integral to this strategy, but doing so can induce fill and take too much time. By modifying the steps of CG for the first step above, however, we can make this factorization unnecessary.

If we had computed E, we could perform step 1 using CG as follows:

```
$\beta_k \leftarrow \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{r}_{k-2}^\top \vec{r}_{k-2}}$                ▷ Update search direction
$\vec{v}_k \leftarrow \vec{r}_{k-1} + \beta_k \vec{v}_{k-1}$
$\alpha_k \leftarrow \frac{\vec{r}_{k-1}^\top \vec{r}_{k-1}}{\vec{v}_k^\top E^{-1} A E^{-\top} \vec{v}_k}$                ▷ Line search
$\vec{y}_k \leftarrow \vec{y}_{k-1} + \alpha_k \vec{v}_k$                ▷ Update estimate
$\vec{r}_k \leftarrow \vec{r}_{k-1} - \alpha_k E^{-1} A E^{-\top} \vec{v}_k$                ▷ Update residual
```

This iteration will converge according to the conditioning of $E^{-1}AE^{-\top}$.

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

```
$\beta_k \leftarrow \frac{\tilde{r}_{k-1}^\top P \tilde{r}_{k-1}}{\tilde{r}_{k-2}^\top P \tilde{r}_{k-2}}$                ▷ Update search direction
$\tilde{v}_k \leftarrow P\tilde{r}_{k-1} + \beta_k \tilde{v}_{k-1}$
$\alpha_k \leftarrow \frac{\tilde{r}_{k-1}^\top P \tilde{r}_{k-1}}{\tilde{v}_k^\top A \tilde{v}_k}$                ▷ Line search
$\vec{x}_k \leftarrow \vec{x}_{k-1} + \alpha_k \tilde{v}_k$                ▷ Update estimate
$\tilde{r}_k \leftarrow \tilde{r}_{k-1} - \alpha_k A \tilde{v}_k$                ▷ Update residual
```

This iteration does not depend on the Cholesky factorization of P^{-1}, but instead can be carried out using only P and A. By the substitutions above, $\vec{x}_k \to \vec{x}^*$, and this scheme enjoys the benefits of preconditioning without needing to compute the Cholesky factorization of P.

As a side note, more general 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 extension presents a common trade-off: If a preconditioner takes too long to apply in each iteration of CG, it may not be worth the reduced number of iterations.

11.3.2 Common Preconditioners

Finding good preconditioners in practice is as much an art as it is a science. Finding an effective 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, so rarely do applications of CG appear that do *not* use a preconditioner.

The best strategy for finding P often is application-specific, and generally it is necessary to test a few possibilities for P before settling on the most effective option. A few common generic preconditioners include the following:

- A *diagonal* (or "*Jacobi*") preconditioner 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 preconditioner 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., that it only has nonzeros on its diagonal or where A has nonzeros.

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

- The nonzero values in A can be used to construct a graph with edge (i, j) whenever $a_{ij} \neq 0$. 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* can be effective for linear systems arising from differential equations like those considered in Chapter 16.

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.

11.4 OTHER ITERATIVE ALGORITHMS

The algorithms we have developed in this chapter apply to 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 (see, e.g., [7, 50, 56, 105]), but we summarize some techniques here:

- *Splitting* methods decompose $A = M - N$ and use the fact 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 $M\vec{x}_k = N\vec{x}_{k-1} + \vec{b}$; 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 applies the CG algorithm to the normal equations $A^\top A\vec{x} = A^\top\vec{b}$. This method is guaranteed to converge so long as A is full-rank, but convergence can be slow thanks to poor conditioning of $A^\top A$ as in §5.1.

- 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 $A^\top\vec{y}$.

- Methods such as MINRES and SYMMLQ apply to all symmetric matrices A by replacing the quadratic form $f(\vec{x})$ with $g(\vec{x}) \equiv \|\vec{b} - A\vec{x}\|_2^2$ [93]; 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 [94, 42].

- 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 [106, 44, 40, 115, 126]. 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, methods like the *Fletcher-Reeves*, *Hestenes-Stiefel*, *Polak-Ribière*, and *Dai-Yuan* algorithms return to the more general problem of minimizing a non-quadratic function f, applying conjugate gradient steps to finding new line search directions [30, 41, 59, 100]. Functions f that are well-approximated by quadratics can be minimized very effectively using these strategies, even though they do not necessarily make use of the Hessian. For instance, the Fletcher-Reeves method replaces the residual in CG iterations with the negative gradient $-\nabla f$.

Most of these algorithms are nearly as easy to implement as CG or gradient descent. Pre-packaged implementations are readily available that only require A and $\vec{b}$ as input; they typically require the end user to implement subroutines for multiplying vectors by A and by $A^\top$, which can be a technical challenge in some cases when A is only known implicitly.

As a rule of thumb, the more general 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 need to compensate for this lack of assumptions. This said, there are no hard-and-fast rules that can be applied by examining the elements of A for guessing the most successful iterative scheme.

11.5 EXERCISES

11.1 If we use infinite-precision arithmetic (so rounding is not an issue), can the conjugate gradients algorithm be used to recover *exact* solutions to $A\vec{x} = \vec{b}$ for symmetric positive definite matrices A? Why or why not?

11.2 Suppose $A \in \mathbb{R}^{n\times n}$ is invertible but not symmetric or positive definite.

(a) Show that $A^\top A$ is symmetric and positive definite.

(b) Propose a strategy for solving $A\vec{x} = \vec{b}$ using the conjugate gradients algorithm based on your observation in (a).

(c) How quickly do you expect conjugate gradients to converge in this case? Why?

11.3 Propose a method for preconditioning the gradient descent algorithm from §11.1.1, paralleling the derivation in §11.3.

11.4 In this problem we will derive an iterative method of solving $A\vec{x} = \vec{b}$ via *splitting* [50].

(a) Suppose we decompose $A = M - N$, where M is invertible. Show that the iterative scheme $\vec{x}_k = M^{-1}(N\vec{x}_{k-1} + \vec{b})$ converges to $A^{-1}\vec{b}$ when $\max\{|\lambda| : \lambda \text{ is an eigenvalue of } M^{-1}N\} < 1$.
Hint: Define $\vec{x}^* = A^{-1}\vec{b}$ and take $\vec{e}_k = \vec{x}_k - \vec{x}^*$. Show that $\vec{e}_k = G^k\vec{e}_0$, where $G = M^{-1}N$. For this problem, you can assume that the eigenvectors of G span $\mathbb{R}^n$ (it is possible to prove this statement without the assumption but doing so requires more analysis than we have covered).

(b) Suppose A is strictly diagonally dominant, that is, for each i it satisfies

$$\sum_{j \neq i} |a_{ij}| < |a_{ii}|.$$

Suppose we define M to be the diagonal part of A and $N = M - A$. Show that the iterative scheme from Exercise 11.4a converges in this case. You can assume the statement from Exercise 11.4a holds regardless of the eigenspace of G.

11.5 As introduced in §10.4.3, a graph is a data structure $G = (V, E)$ consisting of n vertices in a set $V = \{1, \ldots, n\}$ and a set of edges $E \subseteq V \times V$. A common problem is *graph layout*, where we choose positions of the vertices in V on the plane $\mathbb{R}^2$ respecting the connectivity of G. For this problem we will assume $(i, i) \notin E$ for all $i \in V$.

(a) Take $\vec{v}_1, \ldots, \vec{v}_n \in \mathbb{R}^2$ to be the positions of the vertices in V; these are the unknowns in graph layout. The Dirichlet energy of a layout is

$$E(\vec{v}_1, \ldots, \vec{v}_n) = \sum_{(i,j) \in E} \|\vec{v}_i - \vec{v}_j\|_2^2.$$

Suppose an artist specifies positions of vertices in a nonempty subset $V_0 \subseteq V$. We will label these positions as $\vec{v}_k^0$ for $k \in V_0$. Derive two $(n - |V_0|) \times (n - |V_0|)$ linear systems of equations satisfied by the x and y components of the unknown $\vec{v}_i$'s solving the following minimization problem:

$$\begin{aligned} &\text{minimize } E(\vec{v}_1, \ldots, \vec{v}_n) \\ &\text{subject to } \vec{v}_k = \vec{v}_k^0 \ \forall k \in V_0. \end{aligned}$$

Hint: Your answer can be written as two independent linear systems $A\vec{x} = \vec{b}_x$ and $A\vec{y} = \vec{b}_y$.

(b) Show that your systems from the previous part are symmetric and positive definite.

(c) Implement both gradient descent and conjugate gradients for solving this system, updating a display of the graph layout after each iteration. Compare the number of iterations needed to reach a reasonable solution using both strategies.

(d) Implement preconditioned conjugate gradients using a preconditioner of your choice. How much does convergence improve?

DH 11.6 The *successive over-relaxation* (SOR) method is an example of an iterative splitting method for solving $A\vec{x} = \vec{b}$, for $A \in \mathbb{R}^{n \times n}$. Suppose we decompose $A = D + L + U$, where D, L, and U are the diagonal, strictly lower-triangular, and strictly upper-triangular parts of A, respectively. Then, the SOR iteration is given by:

$$(\omega^{-1} D + L)\vec{x}_{k+1} = ((\omega^{-1} - 1)D - U)\vec{x}_k + \vec{b},$$

for some constant $\omega \in \mathbb{R}$. We will show that if A is symmetric and positive definite and $\omega \in (0, 2)$, then the SOR method converges.

(a) Show how SOR is an instance of the splitting method in Exercise 11.4 by defining matrices M and N appropriately. Hence, using this problem we now only need to show that $\rho(G) < 1$ for $G = M^{-1}N$ to establish convergence of SOR.

(b) Define $Q \equiv (\omega^{-1}D + L)$ and let $\vec{y} = (I_{n\times n} - G)\vec{x}$ for an arbitrary eigenvector $\vec{x} \in \mathbb{C}^n$ of G with corresponding eigenvalue $\lambda \in \mathbb{C}$. Derive expressions for $Q\vec{y}$ and $(Q - A)\vec{y}$ in terms of A, $\vec{x}$, and λ.

(c) Show that $d_{ii} > 0$ for all i. This expression shows that all the possibly nonzero elements of the diagonal matrix D are positive.

(d) Substitute the definition of Q into your relationships from Exercise 11.6b and simplify to show that:

$$\begin{aligned}\omega^{-1}\langle\vec{y},\vec{y}\rangle_D + \langle\vec{y},\vec{y}\rangle_L &= (1-\bar{\lambda})\langle\vec{x},\vec{x}\rangle_A\\ (\omega^{-1}-1)\langle\vec{y},\vec{y}\rangle_D - \langle\vec{y},\vec{y}\rangle_{U^\top} &= (1-\lambda)\bar{\lambda}\langle\vec{x},\vec{x}\rangle_A.\end{aligned}$$

Note: We are dealing with complex values here, so inner products in this problem are given by $\langle\vec{x},\vec{y}\rangle_A \equiv (A\vec{x})^\top\text{conjugate}(\vec{y})$.

(e) Recalling our assumptions on A, write a relationship between L and U. Use this and the previous part to conclude that

$$(2\omega^{-1}-1)\langle\vec{y},\vec{y}\rangle_D = (1-|\lambda|^2)\langle\vec{x},\vec{x}\rangle_A.$$

(f) Justify why, under the given assumptions and results of the previous parts, each of $(2\omega^{-1}-1)$, $\langle\vec{y},\vec{y}\rangle_D$, and $\langle\vec{x},\vec{x}\rangle_A$ must be positive. What does this imply about $|\lambda|$? Conclude that the SOR method converges under our assumptions.

$^{\text{DH}}$11.7 ("Gradient domain painting," [86]) Let $I : S \to \mathbb{R}$ be a monochromatic image, where $S \subset \mathbb{R}^2$ is a rectangle. We know I on a collection of square pixels tiling S.

Suppose an artist is editing I in the gradient domain. This means the artist edits the x and y derivatives g_x and g_y of I rather than values in I. After editing g_x and g_y, we need to recover a new image $\tilde{I}$ that has the edited gradients, at least approximately.

(a) For the artist to paint in the gradient domain, we first have to calculate discrete approximations of g_x and g_y using the values of I on different pixels. How might you estimate the derivatives of I in the x and y directions from a pixel using the values of I at one or both of the two horizontally adjacent pixels?

(b) Describe matrices A_x and A_y such that $A_xI = g_x$ and $A_yI = g_y$, where in this case we have written I as a vector $I = [I_{1,1}, I_{1,2}, ..., I_{1,n}, I_{2,1}, ..., I_{m,n}]^T$ and $I_{i,j}$ is the value of I at pixel (i,j). Assume the image I is m pixels tall and n pixels wide.

(c) Give an example of a function $g : \mathbb{R}^2 \to \mathbb{R}^2$ that is not a gradient, that is, g admits no f such that $\nabla f = g$. Justify your answer.

(d) In light of the fact that $\nabla\tilde{I} = g$ may not be solvable exactly, propose an optimization problem whose solution is the "best" approximate solution (in the L_2 norm) to this equation. Describe the advantage of using conjugate gradients to solve such a system.

11.8 The locally optimal block preconditioned conjugate gradient (LOBPCG) algorithm applies conjugate gradients to finding generalized eigenvectors $\vec{x}$ of matrices A and B satisfying $A\vec{x} = \lambda B\vec{x}$ [75, 76]. Assume $A, B \in \mathbb{R}^{n\times n}$ are symmetric and positive definite.

(a) Define the *generalized Rayleigh quotient* $\rho(\vec{x})$ as the function

$$\rho(\vec{x}) \equiv \frac{\vec{x}^\top A\vec{x}}{\vec{x}^\top B\vec{x}}.$$

Show that $\nabla\rho$ is parallel to $A\vec{x} - \rho(\vec{x})B\vec{x}$.

(b) Show that critical points of $\rho(\vec{x})$ with $\vec{x} \neq \vec{0}$ are the generalized eigenvectors of (A, B). Argue that the largest and smallest generalized eigenvalues come from maximizing and minimizing $\rho(\vec{x})$, respectively.

(c) Suppose we wish to find the generalized eigenvector with the largest eigenvalue. If we search in the gradient direction from the current iterate $\vec{x}$, we must solve the following line search problem:

$$\max_{\alpha\in\mathbb{R}} \rho(\vec{x} + \alpha\vec{r}(\vec{x})),$$

where $r(\vec{x}) \equiv A\vec{x} - \rho(\vec{x})B\vec{x}$. Show that α can be found by computing roots of a low-degree polynomial.

(d) Based on our construction above, propose an iteration for finding $\vec{x}$. When $B = I_{n\times n}$, is this method the same as the power method?