

$$\left. \begin{array}{l} \kappa + 1 \end{array} \right)^{2i} .$$

7. The Method of Conjugate Directions

7.1. Conjugacy

Steepest Descent often finds itself taking steps in the same direction as earlier steps (see Figure 8). Wouldn't it be better if, every time we took a step, we got it right the first time? Here's an idea: let's pick a set of orthogonal *search directions* $d_{(0)}, d_{(1)}, \dots, d_{(n-1)}$. In each search direction, we'll take exactly one step, and that step will be just the right length to line up evenly with x . After n steps, we'll be done.

Figure 21 illustrates this idea, using the coordinate axes as search directions. The first (horizontal) step leads to the correct x_1 -coordinate; the second (vertical) step will hit home. Notice that $e_{(1)}$ is orthogonal to $d_{(0)}$. In general, for each step we choose a point

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)}. \quad (29)$$

To find the value of $\alpha_{(i)}$, use the fact that $e_{(i+1)}$ should be orthogonal to $d_{(i)}$, so that we need never step in

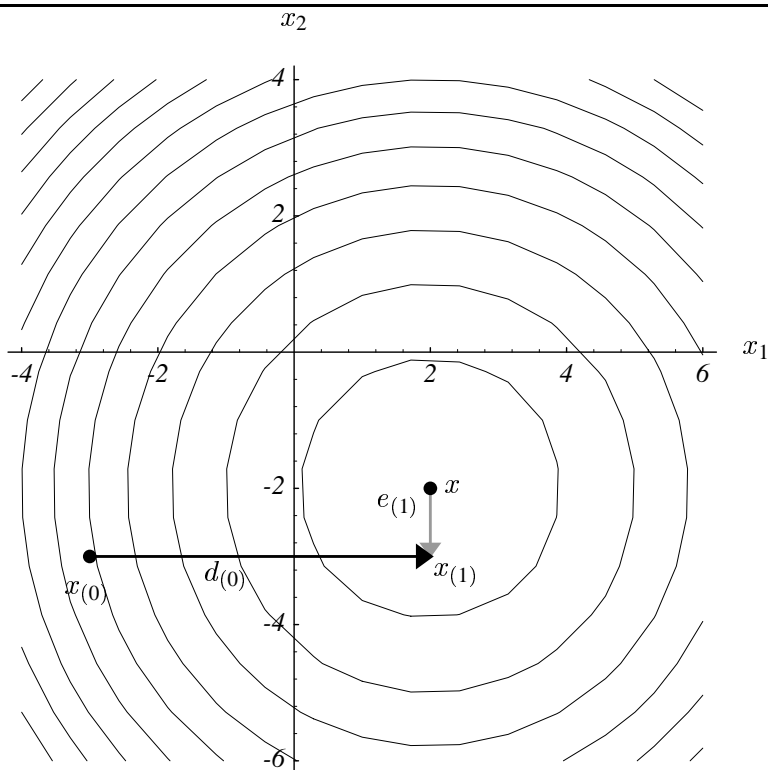


Figure 21: The Method of Orthogonal Directions. Unfortunately, this method only works if you already know the answer.

the direction of $d_{(i)}$ again. Using this condition, we have

$$\begin{aligned} d_{(i)}^T e_{(i+1)} &= 0 \\ d_{(i)}^T (e_{(i)} + \alpha_{(i)} d_{(i)}) &= 0 \quad (\text{by Equation 29}) \\ \alpha_{(i)} &= -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}}. \end{aligned} \quad (30)$$

Unfortunately, we haven't accomplished anything, because we can't compute $\alpha_{(i)}$ without knowing $e_{(i)}$; and if we knew $e_{(i)}$, the problem would already be solved.

The solution is to make the search directions A -orthogonal instead of orthogonal. Two vectors $d_{(i)}$ and $d_{(j)}$ are A -orthogonal, or *conjugate*, if

$$d_{(i)}^T A d_{(j)} = 0.$$

Figure 22(a) shows what A -orthogonal vectors look like. Imagine if this article were printed on bubble gum, and you grabbed Figure 22(a) by the ends and stretched it until the ellipses appeared circular. The vectors would then appear orthogonal, as in Figure 22(b).

Our new requirement is that $e_{(i+1)}$ be A -orthogonal to $d_{(i)}$ (see Figure 23(a)). Not coincidentally, this orthogonality condition is equivalent to finding the minimum point along the search direction $d_{(i)}$, as in

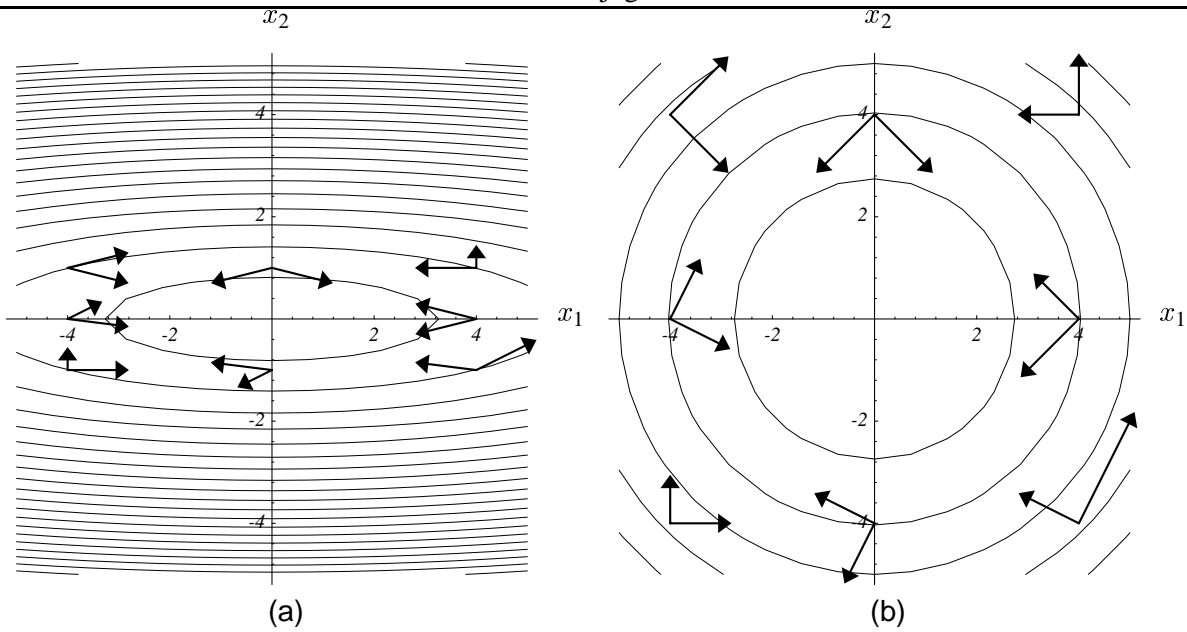


Figure 22: These pairs of vectors are A -orthogonal ... because these pairs of vectors are orthogonal.

Steepest Descent. To see this, set the directional derivative to zero:

$$\begin{aligned} \frac{d}{d\alpha} f(x_{(i+1)}) &= 0 \\ f'(x_{(i+1)})^T \frac{d}{d\alpha} x_{(i+1)} &= 0 \\ -r_{(i+1)}^T d_{(i)} &= 0 \\ d_{(i)}^T A e_{(i+1)} &= 0. \end{aligned}$$

Following the derivation of Equation 30, here is the expression for $\alpha_{(i)}$ when the search directions are A -orthogonal:

$$\alpha_{(i)} = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} \tag{31}$$

$$= \frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}}. \tag{32}$$

Unlike Equation 30, we can calculate this expression. Note that if the search vector were the residual, this formula would be identical to the formula used by Steepest Descent.

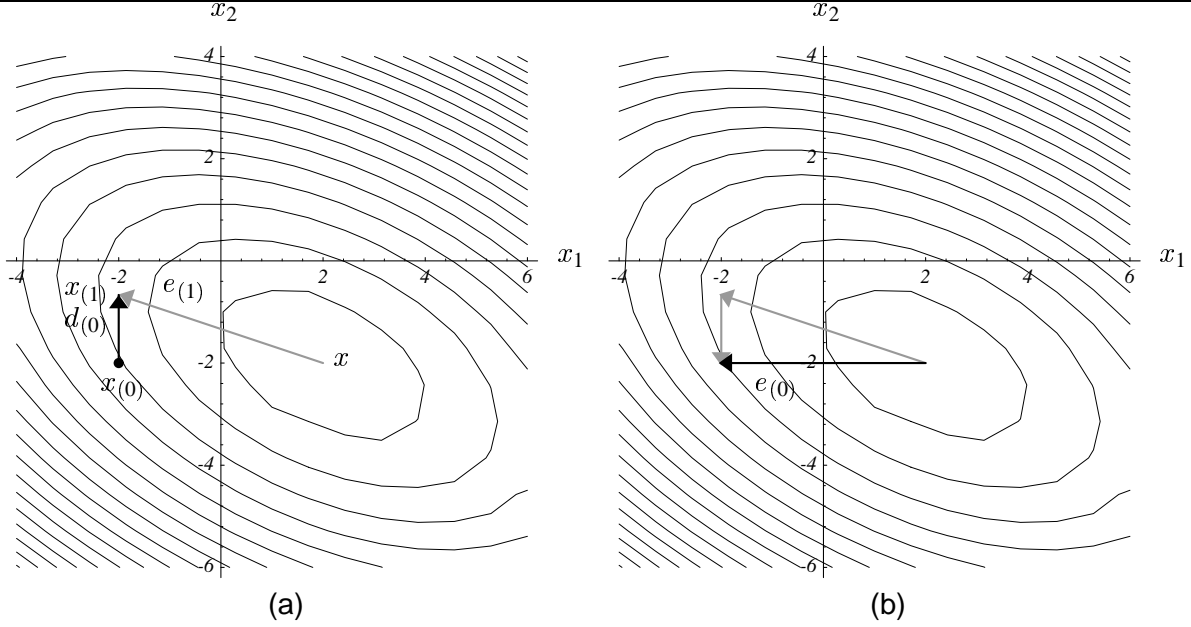


Figure 23: The method of Conjugate Directions converges in n steps. (a) The first step is taken along some direction $d_{(0)}$. The minimum point $x_{(1)}$ is chosen by the constraint that $e_{(1)}$ must be A -orthogonal to $d_{(0)}$. (b) The initial error $e_{(0)}$ can be expressed as a sum of A -orthogonal components (gray arrows). Each step of Conjugate Directions eliminates one of these components.

To prove that this procedure really does compute x in n steps, express the error term as a linear combination of search directions; namely,

$$e_{(0)} = \sum_{j=0}^{n-1} \delta_j d_{(j)}. \quad (33)$$

The values of δ_j can be found by a mathematical trick. Because the search directions are A -orthogonal, it is possible to eliminate all the δ_j values but one from Expression 33 by premultiplying the expression by $d_{(k)}^T A$:

$$\begin{aligned} d_{(k)}^T A e_{(0)} &= \sum_j \delta_j d_{(k)}^T A d_{(j)} \\ d_{(k)}^T A e_{(0)} &= \delta_{(k)} d_{(k)}^T A d_{(k)} \quad (\text{by } A\text{-orthogonality of } d \text{ vectors}) \\ \delta_{(k)} &= \frac{d_{(k)}^T A e_{(0)}}{d_{(k)}^T A d_{(k)}} \\ &= \frac{d_{(k)}^T A (e_{(0)} + \sum_{i=0}^{k-1} \alpha_{(i)} d_{(i)})}{d_{(k)}^T A d_{(k)}} \quad (\text{by } A\text{-orthogonality of } d \text{ vectors}) \\ &= \frac{d_{(k)}^T A e_{(k)}}{d_{(k)}^T A d_{(k)}} \quad (\text{By Equation 29}). \end{aligned} \quad (34)$$

By Equations 31 and 34, we find that $\alpha_{(i)} = -\delta_{(i)}$. This fact gives us a new way to look at the error term. As the following equation shows, the process of building up x component by component can also be

viewed as a process of cutting down the error term component by component (see Figure 23(b)).

$$\begin{aligned}
 e_{(i)} &= e_{(0)} + \sum_{j=0}^{i-1} \alpha_{(j)} d_{(j)} \\
 &= \sum_{j=0}^{n-1} \delta_{(j)} d_{(j)} - \sum_{j=0}^{i-1} \delta_{(j)} d_{(j)} \\
 &= \sum_{j=i}^{n-1} \delta_{(j)} d_{(j)}.
 \end{aligned} \tag{35}$$

After n iterations, every component is cut away, and $e_{(n)} = 0$; the proof is complete.

7.2. Gram-Schmidt Conjugation

All that is needed now is a set of A -orthogonal search directions $\{d_{(i)}\}$. Fortunately, there is a simple way to generate them, called a *conjugate Gram-Schmidt process*.

Suppose we have a set of n linearly independent vectors u_0, u_1, \dots, u_{n-1} . The coordinate axes will do in a pinch, although more intelligent choices are possible. To construct $d_{(i)}$, take u_i and subtract out any components that are not A -orthogonal to the previous d vectors (see Figure 24). In other words, set $d_{(0)} = u_0$, and for $i > 0$, set

$$d_{(i)} = u_i + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}, \tag{36}$$

where the β_{ik} are defined for $i > k$. To find their values, use the same trick used to find δ_j :

$$\begin{aligned}
 d_{(i)}^T A d_{(j)} &= u_i^T A d_{(j)} + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}^T A d_{(j)} \\
 0 &= u_i^T A d_{(j)} + \beta_{ij} d_{(j)}^T A d_{(j)}, \quad i > j \quad (\text{by } A\text{-orthogonality of } d \text{ vectors}) \\
 \beta_{ij} &= -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}}
 \end{aligned} \tag{37}$$

The difficulty with using Gram-Schmidt conjugation in the method of Conjugate Directions is that all the old search vectors must be kept in memory to construct each new one, and furthermore $\mathcal{O}(n^3)$ operations

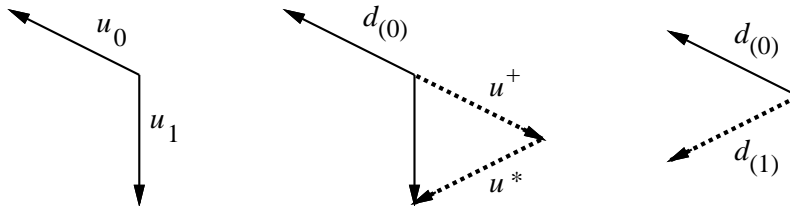


Figure 24: Gram-Schmidt conjugation of two vectors. Begin with two linearly independent vectors u_0 and u_1 . Set $d_{(0)} = u_0$. The vector u_1 is composed of two components: u^* , which is A -orthogonal (or conjugate) to $d_{(0)}$, and u^+ , which is parallel to $d_{(0)}$. After conjugation, only the A -orthogonal portion remains, and $d_{(1)} = u^*$.

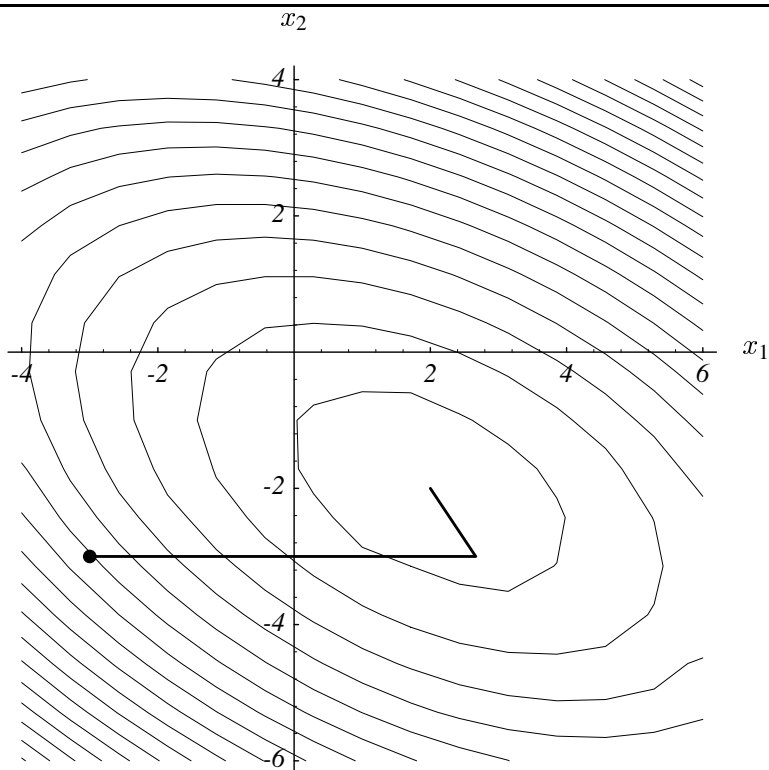


Figure 25: The method of Conjugate Directions using the axial unit vectors, also known as Gaussian elimination.

are required to generate the full set. In fact, if the search vectors are constructed by conjugation of the axial unit vectors, Conjugate Directions becomes equivalent to performing Gaussian elimination (see Figure 25). As a result, the method of Conjugate Directions enjoyed little use until the discovery of CG — which *is* a method of Conjugate Directions — cured these disadvantages.

An important key to understanding the method of Conjugate Directions (and also CG) is to notice that Figure 25 is just a stretched copy of Figure 21! Remember that when one is performing the method of Conjugate Directions (including CG), one is simultaneously performing the method of Orthogonal Directions in a stretched (scaled) space.

7.3. Optimality of the Error Term

Conjugate Directions has an interesting property: it finds at every step the best solution within the bounds of where it's been allowed to explore. Where has it been allowed to explore? Let \mathcal{D}_i be the i -dimensional subspace $\text{span}\{d_{(0)}, d_{(1)}, \dots, d_{(i-1)}\}$; the value $e_{(i)}$ is chosen from $e_{(0)} + \mathcal{D}_i$. What do I mean by “best solution”? I mean that Conjugate Directions chooses the value from $e_{(0)} + \mathcal{D}_i$ that minimizes $\|e_{(i)}\|_A$ (see Figure 26). In fact, some authors derive CG by trying to minimize $\|e_{(i)}\|_A$ within $e_{(0)} + \mathcal{D}_i$.

In the same way that the error term can be expressed as a linear combination of search directions

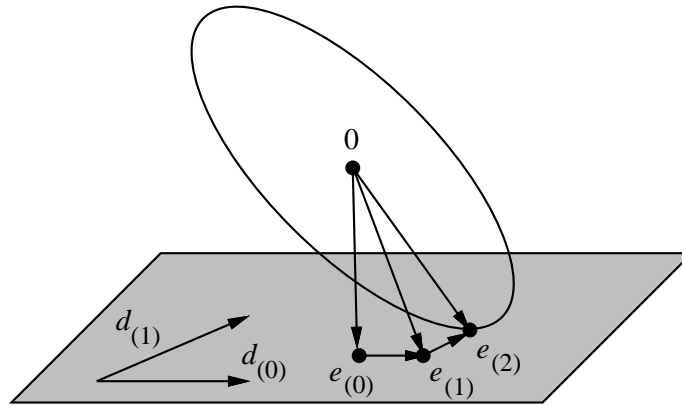


Figure 26: In this figure, the shaded area is $e_{(0)} + \mathcal{D}_2 = e_{(0)} + \text{span}\{d_{(0)}, d_{(1)}\}$. The ellipsoid is a contour on which the energy norm is constant. After two steps, Conjugate Directions finds $e_{(2)}$, the point on $e_{(0)} + \mathcal{D}_2$ that minimizes $\|e\|_A$.

(Equation 35), its energy norm can be expressed as a summation.

$$\begin{aligned} \|e_{(i)}\|_A &= \sum_{j=i}^{n-1} \sum_{k=i}^{n-1} \delta_{(j)} \delta_{(k)} d_{(j)}^T A d_{(k)} \quad (\text{by Equation 35}) \\ &= \sum_{j=i}^{n-1} \delta_{(j)}^2 d_{(j)}^T A d_{(j)} \quad (\text{by } A\text{-orthogonality of } d \text{ vectors}). \end{aligned}$$

Each term in this summation is associated with a search direction that has not yet been traversed. Any other vector e chosen from $e_{(0)} + \mathcal{D}_i$ must have these same terms in its expansion, which proves that $e_{(i)}$ must have the minimum energy norm.

Having proven the optimality with equations, let's turn to the intuition. Perhaps the best way to visualize the workings of Conjugate Directions is by comparing the space we are working in with a "stretched" space, as in Figure 22. Figures 27(a) and 27(c) demonstrate the behavior of Conjugate Directions in \mathbb{R}^2 and \mathbb{R}^3 ; lines that appear perpendicular in these illustrations are orthogonal. On the other hand, Figures 27(b) and 27(d) show the same drawings in spaces that are stretched (along the eigenvector axes) so that the ellipsoidal contour lines become spherical. Lines that appear perpendicular in these illustrations are A -orthogonal.

In Figure 27(a), the Method of Conjugate Directions begins at $x_{(0)}$, takes a step in the direction of $d_{(0)}$, and stops at the point $x_{(1)}$, where the error vector $e_{(1)}$ is A -orthogonal to $d_{(0)}$. Why should we expect this to be the minimum point on $x_{(0)} + \mathcal{D}_1$? The answer is found in Figure 27(b): in this stretched space, $e_{(1)}$ appears perpendicular to $d_{(0)}$ because they are A -orthogonal. The error vector $e_{(1)}$ is a radius of the concentric circles that represent contours on which $\|e\|_A$ is constant, so $x_{(0)} + \mathcal{D}_1$ must be tangent at $x_{(1)}$ to the circle that $x_{(1)}$ lies on. Hence, $x_{(1)}$ is the point on $x_{(0)} + \mathcal{D}_1$ that minimizes $\|e_{(1)}\|_A$.

This is not surprising; we have already seen in Section 7.1 that A -conjugacy of the search direction and the error term is equivalent to minimizing f (and therefore $\|e\|_A$) along the search direction. However, after Conjugate Directions takes a second step, minimizing $\|e\|_A$ along a second search direction $d_{(1)}$, why should we expect that $\|e\|_A$ will *still* be minimized in the direction of $d_{(0)}$? After taking i steps, why will $f(x_{(i)})$ be minimized over all of $x_{(0)} + \mathcal{D}_i$?

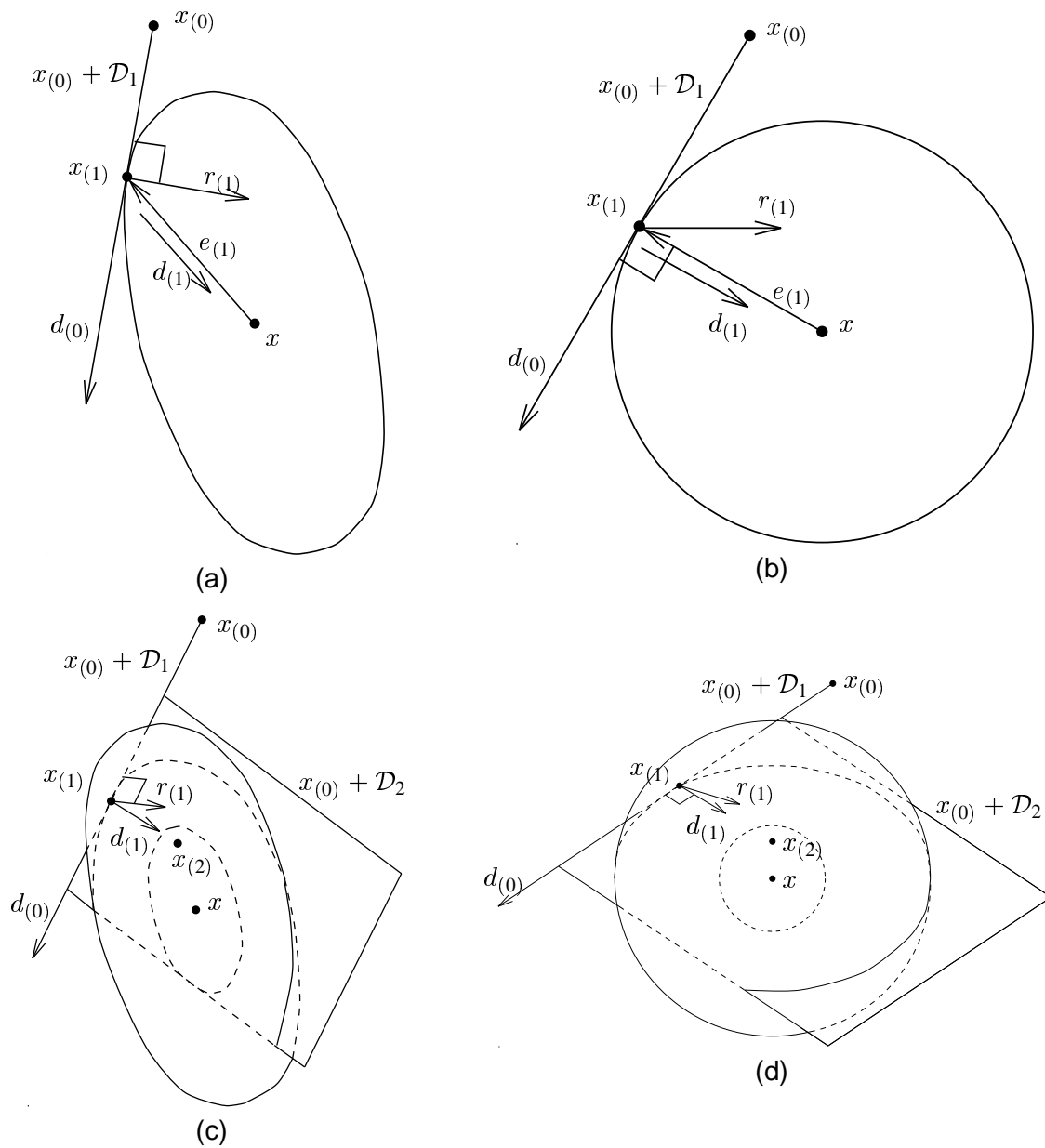


Figure 27: Optimality of the Method of Conjugate Directions. (a) A two-dimensional problem. Lines that appear perpendicular are orthogonal. (b) The same problem in a “stretched” space. Lines that appear perpendicular are A -orthogonal. (c) A three-dimensional problem. Two concentric ellipsoids are shown; x is at the center of both. The line $x(0) + \mathcal{D}_1$ is tangent to the outer ellipsoid at $x(1)$. The plane $x(0) + \mathcal{D}_2$ is tangent to the inner ellipsoid at $x(2)$. (d) Stretched view of the three-dimensional problem.

In Figure 27(b), $d_{(0)}$ and $d_{(1)}$ appear perpendicular because they are A -orthogonal. It is clear that $d_{(1)}$ must point to the solution x , because $d_{(0)}$ is tangent at $x_{(1)}$ to a circle whose center is x . However, a three-dimensional example is more revealing. Figures 27(c) and 27(d) each show two concentric ellipsoids. The point $x_{(1)}$ lies on the outer ellipsoid, and $x_{(2)}$ lies on the inner ellipsoid. Look carefully at these figures: the plane $x_{(0)} + \mathcal{D}_2$ slices through the larger ellipsoid, and is tangent to the smaller ellipsoid at $x_{(2)}$. The point x is at the center of the ellipsoids, underneath the plane.

Looking at Figure 27(c), we can rephrase our question. Suppose you and I are standing at $x_{(1)}$, and want to walk to the point that minimizes $\|e\|$ on $x_{(0)} + \mathcal{D}_2$; but we are constrained to walk along the search direction $d_{(1)}$. If $d_{(1)}$ points to the minimum point, we will succeed. Is there any reason to expect that $d_{(1)}$ will point the right way?

Figure 27(d) supplies an answer. Because $d_{(1)}$ is A -orthogonal to $d_{(0)}$, they are perpendicular in this diagram. Now, suppose you were staring down at the plane $x_{(0)} + \mathcal{D}_2$ as if it were a sheet of paper; the sight you'd see would be identical to Figure 27(b). The point $x_{(2)}$ would be at the center of the paper, and the point x would lie underneath the paper, directly under the point $x_{(2)}$. Because $d_{(0)}$ and $d_{(1)}$ are perpendicular, $d_{(1)}$ points directly to $x_{(2)}$, which is the point in $x_{(0)} + \mathcal{D}_2$ closest to x . The plane $x_{(0)} + \mathcal{D}_2$ is tangent to the sphere on which $x_{(2)}$ lies. If you took a third step, it would be straight down from $x_{(2)}$ to x , in a direction A -orthogonal to \mathcal{D}_2 .

Another way to understand what is happening in Figure 27(d) is to imagine yourself standing at the solution point x , pulling a string connected to a bead that is constrained to lie in $x_{(0)} + \mathcal{D}_i$. Each time the *expanding subspace* \mathcal{D} is enlarged by a dimension, the bead becomes free to move a little closer to you. Now if you stretch the space so it looks like Figure 27(c), you have the Method of Conjugate Directions.

Another important property of Conjugate Directions is visible in these illustrations. We have seen that, at each step, the hyperplane $x_{(0)} + \mathcal{D}_i$ is tangent to the ellipsoid on which $x_{(i)}$ lies. Recall from Section 4 that the residual at any point is orthogonal to the ellipsoidal surface at that point. It follows that $r_{(i)}$ is orthogonal to \mathcal{D}_i as well. To show this fact mathematically, premultiply Equation 35 by $-d_{(i)}^T A$:

$$-d_{(i)}^T A e_{(j)} = -\sum_{j=i}^{n-1} \delta_{(j)} d_{(i)}^T A d_{(j)} \quad (38)$$

$$d_{(i)}^T r_{(j)} = 0, \quad i < j \quad (\text{by } A\text{-orthogonality of } d\text{-vectors}). \quad (39)$$

We could have derived this identity by another tack. Recall that once we take a step in a search direction, we need never step in that direction again; the error term is evermore A -orthogonal to all the old search directions. Because $r_{(i)} = -Ae_{(i)}$, the residual is evermore orthogonal to all the old search directions.

Because the search directions are constructed from the u vectors, the subspace spanned by u_0, \dots, u_{i-1} is \mathcal{D}_i , and the residual $r_{(i)}$ is orthogonal to these previous u vectors as well (see Figure 28). This is proven by taking the inner product of Equation 36 and $r_{(j)}$:

$$d_{(i)}^T r_{(j)} = u_i^T r_{(j)} + \sum_{k=0}^{i-1} \beta_{ik} d_{(k)}^T r_{(j)} \quad (40)$$

$$0 = u_i^T r_{(j)}, \quad i < j \quad (\text{by Equation 39}). \quad (41)$$

There is one more identity we will use later. From Equation 40 (and Figure 28),

$$d_{(i)}^T r_{(i)} = u_i^T r_{(i)}. \quad (42)$$

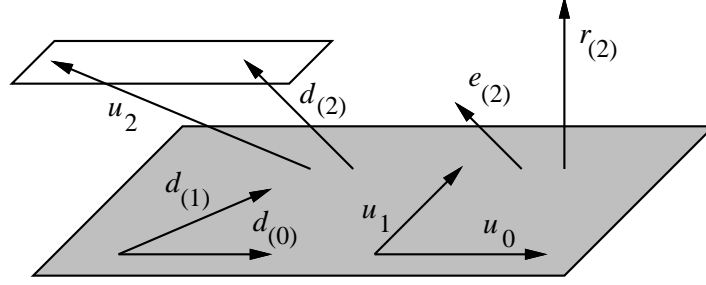


Figure 28: Because the search directions $d_{(0)}, d_{(1)}$ are constructed from the vectors u_0, u_1 , they span the same subspace \mathcal{D}_2 (the gray-colored plane). The error term $e_{(2)}$ is A -orthogonal to \mathcal{D}_2 , the residual $r_{(2)}$ is orthogonal to \mathcal{D}_2 , and a new search direction $d_{(2)}$ is constructed (from u_2) to be A -orthogonal to \mathcal{D}_2 . The endpoints of u_2 and $d_{(2)}$ lie on a plane parallel to \mathcal{D}_2 , because $d_{(2)}$ is constructed from u_2 by Gram-Schmidt conjugation.

To conclude this section, I note that as with the method of Steepest Descent, the number of matrix-vector products per iteration can be reduced to one by using a recurrence to find the residual:

$$\begin{aligned}
 r_{(i+1)} &= -Ae_{(i+1)} \\
 &= -A(e_{(i)} + \alpha_{(i)}d_{(i)}) \\
 &= r_{(i)} - \alpha_{(i)}Ad_{(i)}.
 \end{aligned} \tag{43}$$

8. The Method of Conjugate Gradients

It may seem odd that an article about CG doesn't describe CG until page 30, but all the machinery is now in place. In fact, CG is simply the method of Conjugate Directions where the search directions are constructed by conjugation of the residuals (that is, by setting $u_i = r_{(i)}$).

This choice makes sense for many reasons. First, the residuals worked for Steepest Descent, so why not for Conjugate Directions? Second, the residual has the nice property that it's orthogonal to the previous search directions (Equation 39), so it's guaranteed always to produce a new, linearly independent search direction unless the residual is zero, in which case the problem is already solved. As we shall see, there is an even better reason to choose the residual.

Let's consider the implications of this choice. Because the search vectors are built from the residuals, the subspace $\text{span}\{r_{(0)}, r_{(1)}, \dots, r_{(i-1)}\}$ is equal to \mathcal{D}_i . As each residual is orthogonal to the previous search directions, it is also orthogonal to the previous residuals (see Figure 29); Equation 41 becomes

$$r_{(i)}^T r_{(j)} = 0, \quad i \neq j. \tag{44}$$

Interestingly, Equation 43 shows that each new residual $r_{(i)}$ is just a linear combination of the previous residual and $Ad_{(i-1)}$. Recalling that $d_{(i-1)} \in \mathcal{D}_i$, this fact implies that each new subspace \mathcal{D}_{i+1} is formed from the union of the previous subspace \mathcal{D}_i and the subspace $A\mathcal{D}_i$. Hence,

$$\begin{aligned}
 \mathcal{D}_i &= \text{span}\{d_{(0)}, Ad_{(0)}, A^2d_{(0)}, \dots, A^{i-1}d_{(0)}\} \\
 &= \text{span}\{r_{(0)}, Ar_{(0)}, A^2r_{(0)}, \dots, A^{i-1}r_{(0)}\}.
 \end{aligned}$$

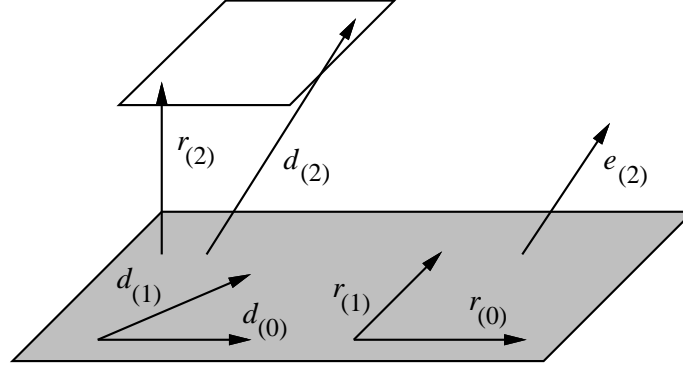


Figure 29: In the method of Conjugate Gradients, each new residual is orthogonal to all the previous residuals and search directions; and each new search direction is constructed (from the residual) to be A -orthogonal to all the previous residuals and search directions. The endpoints of $r_{(2)}$ and $d_{(2)}$ lie on a plane parallel to \mathcal{D}_2 (the shaded subspace). In CG, $d_{(2)}$ is a linear combination of $r_{(2)}$ and $d_{(1)}$.

This subspace is called a *Krylov subspace*, a subspace created by repeatedly applying a matrix to a vector. It has a pleasing property: because Ad_i is included in \mathcal{D}_{i+1} , the fact that the next residual $r_{(i+1)}$ is orthogonal to \mathcal{D}_{i+1} (Equation 39) implies that $r_{(i+1)}$ is A -orthogonal to \mathcal{D}_i . Gram-Schmidt conjugation becomes easy, because $r_{(i+1)}$ is already A -orthogonal to all of the previous search directions except $d_{(i)}$!

Recall from Equation 37 that the Gram-Schmidt constants are $\beta_{ij} = -r_{(i)}^T Ad_{(j)} / d_{(j)}^T Ad_{(j)}$; let us simplify this expression. Taking the inner product of $r_{(i)}$ and Equation 43,

$$\begin{aligned} r_{(i)}^T r_{(j+1)} &= r_{(i)}^T r_{(j)} - \alpha_{(j)} r_{(i)}^T Ad_{(j)} \\ \alpha_{(j)} r_{(i)}^T Ad_{(j)} &= r_{(i)}^T r_{(j)} - r_{(i)}^T r_{(j+1)} \\ r_{(i)}^T Ad_{(j)} &= \begin{cases} \frac{1}{\alpha_{(i)}} r_{(i)}^T r_{(i)}, & i = j, \\ -\frac{1}{\alpha_{(i-1)}} r_{(i)}^T r_{(i)}, & i = j + 1, \\ 0, & \text{otherwise.} \end{cases} \quad (\text{By Equation 44.}) \\ \therefore \beta_{ij} &= \begin{cases} \frac{1}{\alpha_{(i-1)}} \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T Ad_{(i-1)}}, & i = j + 1, \\ 0, & i > j + 1. \end{cases} \quad (\text{By Equation 37.}) \end{aligned}$$

As if by magic, most of the β_{ij} terms have disappeared. It is no longer necessary to store old search vectors to ensure the A -orthogonality of new search vectors. This major advance is what makes CG as important an algorithm as it is, because both the space complexity and time complexity per iteration are reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(m)$, where m is the number of nonzero entries of A . Henceforth, I shall use the abbreviation $\beta_{(i)} = \beta_{i,i-1}$. Simplifying further:

$$\begin{aligned} \beta_{(i)} &= \frac{r_{(i)}^T r_{(i)}}{d_{(i-1)}^T r_{(i-1)}} \quad (\text{by Equation 32}) \\ &= \frac{r_{(i)}^T r_{(i)}}{r_{(i-1)}^T r_{(i-1)}} \quad (\text{by Equation 42}). \end{aligned}$$

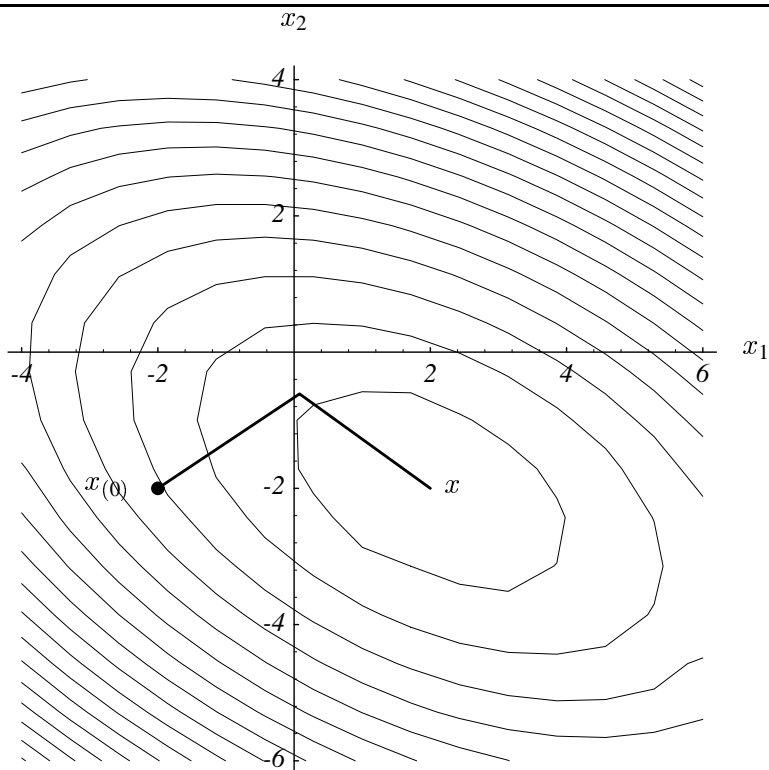


Figure 30: The method of Conjugate Gradients.

Let's put it all together into one piece now. The method of Conjugate Gradients is:

$$d_{(0)} = r_{(0)} = b - Ax_{(0)}, \quad (45)$$

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \quad (\text{by Equations 32 and 42}), \quad (46)$$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} d_{(i)},$$

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A d_{(i)}, \quad (47)$$

$$\beta_{(i+1)} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}, \quad (48)$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)} d_{(i)}. \quad (49)$$

The performance of CG on our sample problem is demonstrated in Figure 30. The name “Conjugate Gradients” is a bit of a misnomer, because the gradients are not conjugate, and the conjugate directions are not all gradients. “Conjugated Gradients” would be more accurate.

9. Convergence Analysis of Conjugate Gradients

CG is complete after n iterations, so why should we care about convergence analysis? In practice, accumulated floating point roundoff error causes the residual to gradually lose accuracy, and cancellation

error causes the search vectors to lose A -orthogonality. The former problem could be dealt with as it was for Steepest Descent, but the latter problem is not easily curable. Because of this loss of conjugacy, the mathematical community discarded CG during the 1960s, and interest only resurged when evidence for its effectiveness as an iterative procedure was published in the seventies.

Times have changed, and so has our outlook. Today, convergence analysis is important because CG is commonly used for problems so large it is not feasible to run even n iterations. Convergence analysis is seen less as a ward against floating point error, and more as a proof that CG is useful for problems that are out of the reach of any exact algorithm.

The first iteration of CG is identical to the first iteration of Steepest Descent, so without changes, Section 6.1 describes the conditions under which CG converges on the first iteration.

9.1. Picking Perfect Polynomials

We have seen that at each step of CG, the value $e_{(i)}$ is chosen from $e_{(0)} + \mathcal{D}_i$, where

$$\begin{aligned} \mathcal{D}_i &= \text{span}\{r_{(0)}, Ar_{(0)}, A^2r_{(0)}, \dots, A^{i-1}r_{(0)}\} \\ &= \text{span}\{Ae_{(0)}, A^2e_{(0)}, A^3e_{(0)}, \dots, A^ie_{(0)}\}. \end{aligned}$$

Krylov subspaces such as this have another pleasing property. For a fixed i , the error term has the form

$$e_{(i)} = \left(I + \sum_{j=1}^i \psi_j A^j \right) e_{(0)}.$$

The coefficients ψ_j are related to the values $\alpha_{(i)}$ and $\beta_{(i)}$, but the precise relationship is not important here. What *is* important is the proof in Section 7.3 that CG chooses the ψ_j coefficients that minimize $\|e_{(i)}\|_A$.

The expression in parentheses above can be expressed as a polynomial. Let $P_i(\lambda)$ be a polynomial of degree i . P_i can take either a scalar or a matrix as its argument, and will evaluate to the same; for instance, if $P_2(\lambda) = 2\lambda^2 + 1$, then $P_2(A) = 2A^2 + I$. This flexible notation comes in handy for eigenvectors, for which you should convince yourself that $P_i(A)v = P_i(\lambda)v$. (Note that $Av = \lambda v$, $A^2v = \lambda^2v$, and so on.)

Now we can express the error term as

$$e_{(i)} = P_i(A)e_{(0)},$$

if we require that $P_i(0) = 1$. CG chooses this polynomial when it chooses the ψ_j coefficients. Let's examine the effect of applying this polynomial to $e_{(0)}$. As in the analysis of Steepest Descent, express $e_{(0)}$ as a linear combination of orthogonal unit eigenvectors

$$e_{(0)} = \sum_{j=1}^n \xi_j v_j,$$

and we find that

$$\begin{aligned} e_{(i)} &= \sum_j \xi_j P_i(\lambda_j) v_j \\ Ae_{(i)} &= \sum_j \xi_j P_i(\lambda_j) \lambda_j v_j \\ \|e_{(i)}\|_A^2 &= \sum_j \xi_j^2 [P_i(\lambda_j)]^2 \lambda_j. \end{aligned}$$

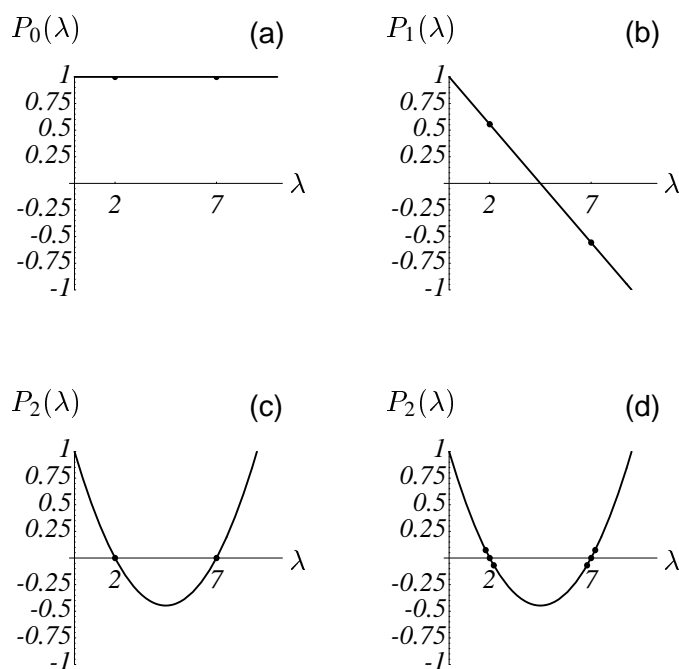


Figure 31: The convergence of CG after i iterations depends on how close a polynomial P_i of degree i can be to zero on each eigenvalue, given the constraint that $P_i(0) = 1$.

CG finds the polynomial that minimizes this expression, but convergence is only as good as the convergence of the worst eigenvector. Letting $\Lambda(A)$ be the set of eigenvalues of A , we have

$$\begin{aligned} \|e_{(i)}\|_A^2 &\leq \min_{P_i} \max_{\lambda \in \Lambda(A)} [P_i(\lambda)]^2 \sum_j \xi_j^2 \lambda_j \\ &= \min_{P_i} \max_{\lambda \in \Lambda(A)} [P_i(\lambda)]^2 \|e_{(0)}\|_A^2. \end{aligned} \quad (50)$$

Figure 31 illustrates, for several values of i , the P_i that minimizes this expression for our sample problem with eigenvalues 2 and 7. There is only one polynomial of degree zero that satisfies $P_0(0) = 1$, and that is $P_0(\lambda) = 1$, graphed in Figure 31(a). The optimal polynomial of degree one is $P_1(\lambda) = 1 - 2\lambda/9$, graphed in Figure 31(b). Note that $P_1(2) = 5/9$ and $P_1(7) = -5/9$, and so the energy norm of the error term after one iteration of CG is no greater than $5/9$ its initial value. Figure 31(c) shows that, after two iterations, Equation 50 evaluates to zero. This is because a polynomial of degree two can be fit to three points ($P_2(0) = 1$, $P_2(2) = 0$, and $P_2(7) = 0$). In general, a polynomial of degree n can fit $n + 1$ points, and thereby accommodate n separate eigenvalues.

The foregoing discussing reinforces our understanding that CG yields the exact result after n iterations; and furthermore proves that CG is quicker if there are duplicated eigenvalues. Given infinite floating point precision, the number of iterations required to compute an exact solution is at most the number of distinct eigenvalues. (There is one other possibility for early termination: $x_{(0)}$ may already be A -orthogonal to some of the eigenvectors of A . If eigenvectors are missing from the expansion of $x_{(0)}$, their eigenvalues may be omitted from consideration in Equation 50. Be forewarned, however, that these eigenvectors may be reintroduced by floating point roundoff error.)

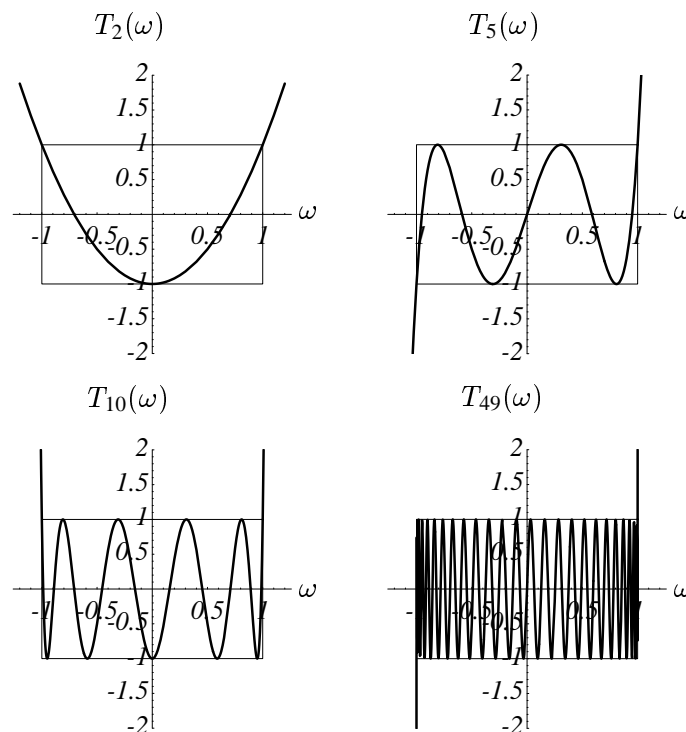


Figure 32: Chebyshev polynomials of degree 2, 5, 10, and 49.

We also find that CG converges more quickly when eigenvalues are clustered together (Figure 31(d)) than when they are irregularly distributed between λ_{min} and λ_{max} , because it is easier for CG to choose a polynomial that makes Equation 50 small.

If we know something about the characteristics of the eigenvalues of A , it is sometimes possible to suggest a polynomial that leads to a proof of a fast convergence. For the remainder of this analysis, however, I shall assume the most general case: the eigenvalues are evenly distributed between λ_{min} and λ_{max} , the number of distinct eigenvalues is large, and floating point roundoff occurs.

9.2. Chebyshev Polynomials

A useful approach is to minimize Equation 50 over the range $[\lambda_{min}, \lambda_{max}]$ rather than at a finite number of points. The polynomials that accomplish this are based on Chebyshev polynomials.

The *Chebyshev polynomial* of degree i is

$$T_i(\omega) = \frac{1}{2} \left[(\omega + \sqrt{\omega^2 - 1})^i + (\omega - \sqrt{\omega^2 - 1})^i \right].$$

(If this expression doesn't look like a polynomial to you, try working it out for i equal to 1 or 2.) Several Chebyshev polynomials are illustrated in Figure 32. The Chebyshev polynomials have the property that $|T_i(\omega)| \leq 1$ (in fact, they oscillate between 1 and -1) on the domain $\omega \in [-1, 1]$, and furthermore that $|T_i(\omega)|$ is maximum on the domain $\omega \notin [-1, 1]$ among all such polynomials. Loosely speaking, $|T_i(\omega)|$ increases as quickly as possible outside the boxes in the illustration.

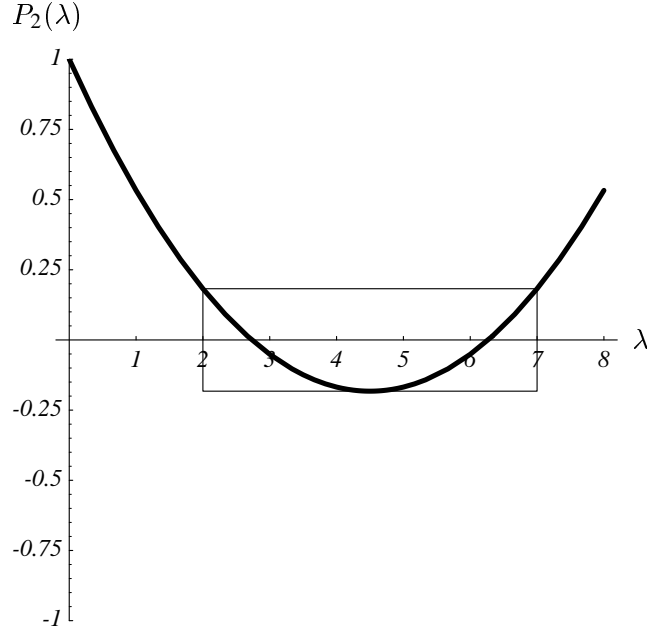


Figure 33: The polynomial $P_2(\lambda)$ that minimizes Equation 50 for $\lambda_{min} = 2$ and $\lambda_{max} = 7$ in the general case. This curve is a scaled version of the Chebyshev polynomial of degree 2. The energy norm of the error term after two iterations is no greater than 0.183 times its initial value. Compare with Figure 31(c), where it is known that there are only two eigenvalues.

It is shown in Appendix C3 that Equation 50 is minimized by choosing

$$P_i(\lambda) = \frac{T_i\left(\frac{\lambda_{max} + \lambda_{min} - 2\lambda}{\lambda_{max} - \lambda_{min}}\right)}{T_i\left(\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}}\right)}.$$

This polynomial has the oscillating properties of Chebyshev polynomials on the domain $\lambda_{min} \leq \lambda \leq \lambda_{max}$ (see Figure 33). The denominator enforces our requirement that $P_i(0) = 1$. The numerator has a maximum value of one on the interval between λ_{min} and λ_{max} , so from Equation 50 we have

$$\begin{aligned} \|e_{(i)}\|_A &\leq T_i\left(\frac{\lambda_{max} + \lambda_{min}}{\lambda_{max} - \lambda_{min}}\right)^{-1} \|e_{(0)}\|_A \\ &= T_i\left(\frac{\kappa + 1}{\kappa - 1}\right)^{-1} \|e_{(0)}\|_A \\ &= 2 \left[\left(\frac{\sqrt{\kappa} + 1}{\sqrt{\kappa} - 1}\right)^i + \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^i \right]^{-1} \|e_{(0)}\|_A. \end{aligned} \quad (51)$$

The second addend inside the square brackets converges to zero as i grows, so it is more common to express the convergence of CG with the weaker inequality

$$\|e_{(i)}\|_A \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1}\right)^i \|e_{(0)}\|_A. \quad (52)$$

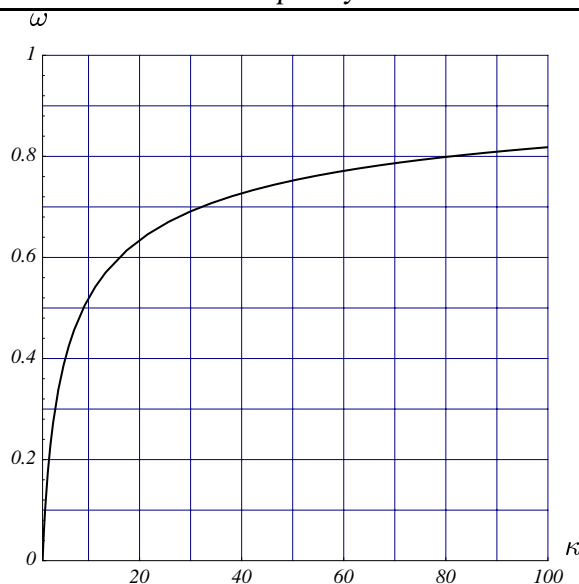


Figure 34: Convergence of Conjugate Gradients (per iteration) as a function of condition number. Compare with Figure 20.

The first step of CG is identical to a step of Steepest Descent. Setting $i = 1$ in Equation 51, we obtain Equation 28, the convergence result for Steepest Descent. This is just the linear polynomial case illustrated in Figure 31(b).

Figure 34 charts the convergence per iteration of CG, ignoring the lost factor of 2. In practice, CG usually converges faster than Equation 52 would suggest, because of good eigenvalue distributions or good starting points. Comparing Equations 52 and 28, it is clear that the convergence of CG is much quicker than that of Steepest Descent (see Figure 35). However, it is not necessarily true that every iteration of CG enjoys faster convergence; for example, the first iteration of CG is an iteration of Steepest Descent. The factor of 2 in Equation 52 allows CG a little slack for these poor iterations.

10. Complexity

The dominating operations during an iteration of either Steepest Descent or CG are matrix-vector products. In general, matrix-vector multiplication requires $\mathcal{O}(m)$ operations, where m is the number of non-zero entries in the matrix. For many problems, including those listed in the introduction, A is sparse and $m \in \mathcal{O}(n)$.

Suppose we wish to perform enough iterations to reduce the norm of the error by a factor of ε ; that is, $\|e_{(i)}\| \leq \varepsilon \|e_{(0)}\|$. Equation 28 can be used to show that the maximum number of iterations required to achieve this bound using Steepest Descent is

$$i \leq \left\lceil \frac{1}{2} \kappa \ln \left(\frac{1}{\varepsilon} \right) \right\rceil,$$

whereas Equation 52 suggests that the maximum number of iterations CG requires is

$$i \leq \left\lceil \frac{1}{2} \sqrt{\kappa} \ln \left(\frac{2}{\varepsilon} \right) \right\rceil.$$

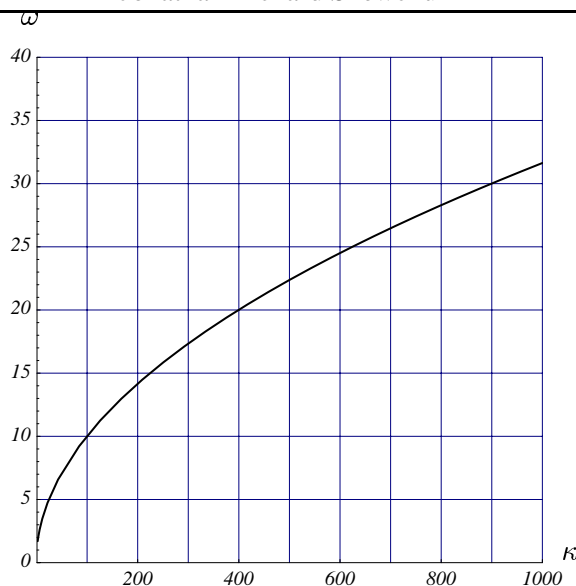


Figure 35: Number of iterations of Steepest Descent required to match one iteration of CG.

I conclude that Steepest Descent has a time complexity of $\mathcal{O}(m\kappa)$, whereas CG has a time complexity of $\mathcal{O}(m\sqrt{\kappa})$. Both algorithms have a space complexity of $\mathcal{O}(m)$.

Finite difference and finite element approximations of second-order elliptic boundary value problems posed on d -dimensional domains often have $\kappa \in \mathcal{O}(n^{2/d})$. Thus, Steepest Descent has a time complexity of $\mathcal{O}(n^2)$ for two-dimensional problems, versus $\mathcal{O}(n^{3/2})$ for CG; and Steepest Descent has a time complexity of $\mathcal{O}(n^{5/3})$ for three-dimensional problems, versus $\mathcal{O}(n^{4/3})$ for CG.

11. Starting and Stopping

In the preceding presentation of the Steepest Descent and Conjugate Gradient algorithms, several details have been omitted; particularly, how to choose a starting point, and when to stop.

11.1. Starting

There's not much to say about starting. If you have a rough estimate of the value of x , use it as the starting value $x_{(0)}$. If not, set $x_{(0)} = 0$; either Steepest Descent or CG will eventually converge when used to solve linear systems. Nonlinear minimization (coming up in Section 14) is trickier, though, because there may be several local minima, and the choice of starting point will determine which minimum the procedure converges to, or whether it will converge at all.

11.2. Stopping

When Steepest Descent or CG reaches the minimum point, the residual becomes zero, and if Equation 11 or 48 is evaluated an iteration later, a division by zero will result. It seems, then, that one must stop

immediately when the residual is zero. To complicate things, though, accumulated roundoff error in the recursive formulation of the residual (Equation 47) may yield a false zero residual; this problem could be resolved by restarting with Equation 45.

Usually, however, one wishes to stop before convergence is complete. Because the error term is not available, it is customary to stop when the norm of the residual falls below a specified value; often, this value is some small fraction of the initial residual ($\|r_{(i)}\| < \varepsilon \|r_{(0)}\|$). See Appendix B for sample code.

12. Preconditioning

Preconditioning is a technique for improving the condition number of a matrix. Suppose that M is a symmetric, positive-definite matrix that approximates A , but is easier to invert. We can solve $Ax = b$ indirectly by solving

$$M^{-1}Ax = M^{-1}b. \quad (53)$$

If $\kappa(M^{-1}A) \ll \kappa(A)$, or if the eigenvalues of $M^{-1}A$ are better clustered than those of A , we can iteratively solve Equation 53 more quickly than the original problem. The catch is that $M^{-1}A$ is not generally symmetric nor definite, even if M and A are.

We can circumvent this difficulty, because for every symmetric, positive-definite M there is a (not necessarily unique) matrix E that has the property that $EE^T = M$. (Such an E can be obtained, for instance, by Cholesky factorization.) The matrices $M^{-1}A$ and $E^{-1}AE^{-T}$ have the same eigenvalues. This is true because if v is an eigenvector of $M^{-1}A$ with eigenvalue λ , then $E^T v$ is an eigenvector of $E^{-1}AE^{-T}$ with eigenvalue λ :

$$(E^{-1}AE^{-T})(E^T v) = (E^T E^{-T})E^{-1}Av = E^T M^{-1}Av = \lambda E^T v.$$

The system $Ax = b$ can be transformed into the problem

$$E^{-1}AE^{-T}\hat{x} = E^{-1}b, \quad \hat{x} = E^T x,$$

which we solve first for \hat{x} , then for x . Because $E^{-1}AE^{-T}$ is symmetric and positive-definite, \hat{x} can be found by Steepest Descent or CG. The process of using CG to solve this system is called the *Transformed Preconditioned Conjugate Gradient Method*:

$$\begin{aligned} \hat{d}_{(0)} &= \hat{r}_{(0)} = E^{-1}b - E^{-1}AE^{-T}\hat{x}_{(0)}, \\ \alpha_{(i)} &= \frac{\hat{r}_{(i)}^T \hat{r}_{(i)}}{\hat{d}_{(i)}^T E^{-1}AE^{-T}\hat{d}_{(i)}}, \\ \hat{x}_{(i+1)} &= \hat{x}_{(i)} + \alpha_{(i)}\hat{d}_{(i)}, \\ \hat{r}_{(i+1)} &= \hat{r}_{(i)} - \alpha_{(i)}E^{-1}AE^{-T}\hat{d}_{(i)}, \\ \beta_{(i+1)} &= \frac{\hat{r}_{(i+1)}^T \hat{r}_{(i+1)}}{\hat{r}_{(i)}^T \hat{r}_{(i)}}, \\ \hat{d}_{(i+1)} &= \hat{r}_{(i+1)} + \beta_{(i+1)}\hat{d}_{(i)}. \end{aligned}$$

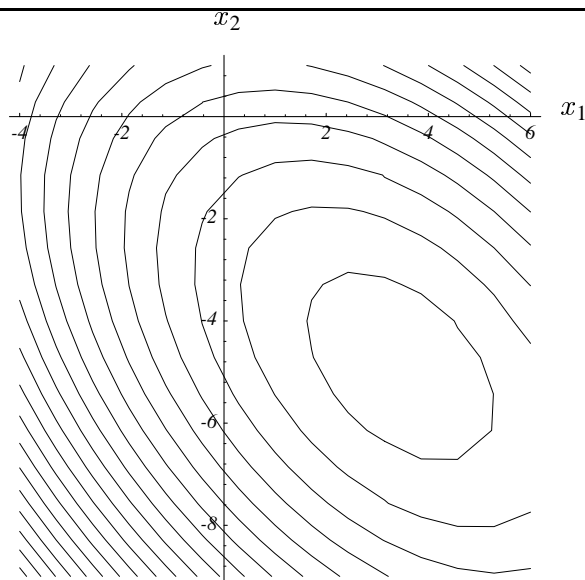


Figure 36: Contour lines of the quadratic form of the diagonally preconditioned sample problem.

This method has the undesirable characteristic that E must be computed. However, a few careful variable substitutions can eliminate E . Setting $\hat{r}_{(i)} = E^{-1}r_{(i)}$ and $\hat{d}_{(i)} = E^T d_{(i)}$, and using the identities $\hat{x}_{(i)} = E^T x_{(i)}$ and $E^{-T}E^{-1} = M^{-1}$, we derive the *Untransformed Preconditioned Conjugate Gradient Method*:

$$\begin{aligned} r_{(0)} &= b - Ax_{(0)}, \\ d_{(0)} &= M^{-1}r_{(0)}, \\ \alpha_{(i)} &= \frac{r_{(i)}^T M^{-1}r_{(i)}}{d_{(i)}^T A d_{(i)}}, \\ x_{(i+1)} &= x_{(i)} + \alpha_{(i)}d_{(i)}, \\ r_{(i+1)} &= r_{(i)} - \alpha_{(i)}A d_{(i)}, \\ \beta_{(i+1)} &= \frac{r_{(i+1)}^T M^{-1}r_{(i+1)}}{r_{(i)}^T M^{-1}r_{(i)}}, \\ d_{(i+1)} &= M^{-1}r_{(i+1)} + \beta_{(i+1)}d_{(i)}. \end{aligned}$$

The matrix E does not appear in these equations; only M^{-1} is needed. By the same means, it is possible to derive a Preconditioned Steepest Descent Method that does not use E .

The effectiveness of a *preconditioner* M is determined by the condition number of $M^{-1}A$, and occasionally by its clustering of eigenvalues. The problem remains of finding a preconditioner that approximates A well enough to improve convergence enough to make up for the cost of computing the product $M^{-1}r_{(i)}$ once per iteration. (It is not necessary to explicitly compute M or M^{-1} ; it is only necessary to be able to compute the effect of applying M^{-1} to a vector.) Within this constraint, there is a surprisingly rich supply of possibilities, and I can only scratch the surface here.

Intuitively, preconditioning is an attempt to stretch the quadratic form to make it appear more spherical, so that the eigenvalues are close to each other. A perfect preconditioner is $M = A$; for this preconditioner, $M^{-1}A$ has a condition number of one, and the quadratic form is perfectly spherical, so solution takes only one iteration. Unfortunately, the preconditioning step is solving the system $Mx = b$, so this isn't a useful preconditioner at all.

The simplest preconditioner is a diagonal matrix whose diagonal entries are identical to those of A . The process of applying this preconditioner, known as *diagonal preconditioning* or *Jacobi preconditioning*, is equivalent to scaling the quadratic form along the coordinate axes. (By comparison, the perfect preconditioner $M = A$ scales the quadratic form along its eigenvector axes.) A diagonal matrix is trivial to invert, but is often only a mediocre preconditioner. The contour lines of our sample problem are shown, after diagonal preconditioning, in Figure 36. Comparing with Figure 3, it is clear that some improvement has occurred. The condition number has improved from 3.5 to roughly 2.8. Of course, this improvement is much more beneficial for systems where $n \gg 2$.

A more elaborate preconditioner is incomplete Cholesky preconditioning. *Cholesky factorization* is a technique for factoring a matrix A into the form LL^T , where L is a lower triangular matrix. Incomplete Cholesky factorization is a variant in which little or no fill is allowed; A is approximated by the product $\hat{L}\hat{L}^T$, where \hat{L} might be restricted to have the same pattern of nonzero elements as A ; other elements of L are thrown away. To use $\hat{L}\hat{L}^T$ as a preconditioner, the solution to $\hat{L}\hat{L}^T w = z$ is computed by backsubstitution (the inverse of $\hat{L}\hat{L}^T$ is never explicitly computed). Unfortunately, incomplete Cholesky preconditioning is not always stable.

Many preconditioners, some quite sophisticated, have been developed. Whatever your choice, it is generally accepted that for large-scale applications, CG should nearly always be used with a preconditioner.

13. Conjugate Gradients on the Normal Equations

CG can be used to solve systems where A is not symmetric, not positive-definite, and even not square. A solution to the least squares problem

$$\min_x \|Ax - b\|^2 \quad (54)$$

can be found by setting the derivative of Expression 54 to zero:

$$A^T Ax = A^T b. \quad (55)$$

If A is square and nonsingular, the solution to Equation 55 is the solution to $Ax = b$. If A is not square, and $Ax = b$ is *overconstrained* — that is, has more linearly independent equations than variables — then there may or may not be a solution to $Ax = b$, but it is always possible to find a value of x that minimizes Expression 54, the sum of the squares of the errors of each linear equation.

$A^T A$ is symmetric and positive (for any x , $x^T A^T Ax = \|Ax\|^2 \geq 0$). If $Ax = b$ is not underconstrained, then $A^T A$ is nonsingular, and methods like Steepest Descent and CG can be used to solve Equation 55. The only nuisance in doing so is that the condition number of $A^T A$ is the square of that of A , so convergence is significantly slower.

An important technical point is that the matrix $A^T A$ is never formed explicitly, because it is less sparse than A . Instead, $A^T A$ is multiplied by d by first finding the product Ad , and then $A^T Ad$. Also, numerical

stability is improved if the value $d^T A^T A d$ (in Equation 46) is computed by taking the inner product of $A d$ with itself.

14. The Nonlinear Conjugate Gradient Method

CG can be used not only to find the minimum point of a quadratic form, but to minimize any continuous function $f(x)$ for which the gradient f' can be computed. Applications include a variety of optimization problems, such as engineering design, neural net training, and nonlinear regression.

14.1. Outline of the Nonlinear Conjugate Gradient Method

To derive nonlinear CG, there are three changes to the linear algorithm: the recursive formula for the residual cannot be used, it becomes more complicated to compute the step size α , and there are several different choices for β .

In nonlinear CG, the residual is always set to the negation of the gradient; $r_{(i)} = -f'(x_{(i)})$. The search directions are computed by Gram-Schmidt conjugation of the residuals as with linear CG. Performing a line search along this search direction is much more difficult than in the linear case, and a variety of procedures can be used. As with the linear CG, a value of $\alpha_{(i)}$ that minimizes $f(x_{(i)} + \alpha_{(i)}d_{(i)})$ is found by ensuring that the gradient is orthogonal to the search direction. We can use any algorithm that finds the zeros of the expression $[f'(x_{(i)} + \alpha_{(i)}d_{(i)})]^T d_{(i)}$.

In linear CG, there are several equivalent expressions for the value of β . In nonlinear CG, these different expressions are no longer equivalent; researchers are still investigating the best choice. Two choices are the Fletcher-Reeves formula, which we used in linear CG for its ease of computation, and the Polak-Ribière formula:

$$\beta_{(i+1)}^{FR} = \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}}, \quad \beta_{(i+1)}^{PR} = \frac{r_{(i+1)}^T (r_{(i+1)} - r_{(i)})}{r_{(i)}^T r_{(i)}}.$$

The Fletcher-Reeves method converges if the starting point is sufficiently close to the desired minimum, whereas the Polak-Ribière method can, in rare cases, cycle infinitely without converging. However, Polak-Ribière often converges much more quickly.

Fortunately, convergence of the Polak-Ribière method can be guaranteed by choosing $\beta = \max\{\beta^{PR}, 0\}$. Using this value is equivalent to restarting CG if $\beta^{PR} < 0$. To *restart* CG is to forget the past search directions, and start CG anew in the direction of steepest descent.

Here is an outline of the nonlinear CG method:

$$\begin{aligned} d_{(0)} &= r_{(0)} = -f'(x_{(0)}), \\ \text{Find } \alpha_{(i)} &\text{ that minimizes } f(x_{(i)} + \alpha_{(i)}d_{(i)}), \\ x_{(i+1)} &= x_{(i)} + \alpha_{(i)}d_{(i)}, \\ r_{(i+1)} &= -f'(x_{(i+1)}), \\ \beta_{(i+1)} &= \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}} \quad \text{or} \quad \beta_{(i+1)} = \max \left\{ \frac{r_{(i+1)}^T (r_{(i+1)} - r_{(i)})}{r_{(i)}^T r_{(i)}}, 0 \right\}, \end{aligned}$$

$$d_{(i+1)} = r_{(i+1)} + \beta_{(i+1)}d_{(i)}.$$

Nonlinear CG comes with few of the convergence guarantees of linear CG. The less similar f is to a quadratic function, the more quickly the search directions lose conjugacy. (It will become clear shortly that the term “conjugacy” still has some meaning in nonlinear CG.) Another problem is that a general function f may have many local minima. CG is not guaranteed to converge to the global minimum, and may not even find a local minimum if f has no lower bound.

Figure 37 illustrates nonlinear CG. Figure 37(a) is a function with many local minima. Figure 37(b) demonstrates the convergence of nonlinear CG with the Fletcher-Reeves formula. In this example, CG is not nearly as effective as in the linear case; this function is deceptively difficult to minimize. Figure 37(c) shows a cross-section of the surface, corresponding to the first line search in Figure 37(b). Notice that there are several minima; the line search finds a value of α corresponding to a nearby minimum. Figure 37(d) shows the superior convergence of Polak-Ribière CG.

Because CG can only generate n conjugate vectors in an n -dimensional space, it makes sense to restart CG every n iterations, especially if n is small. Figure 38 shows the effect when nonlinear CG is restarted every second iteration. (For this particular example, both the Fletcher-Reeves method and the Polak-Ribière method appear the same.)

14.2. General Line Search

Depending on the value of f' , it might be possible to use a fast algorithm to find the zeros of $f'^T d$. For instance, if f' is polynomial in α , then an efficient algorithm for polynomial zero-finding can be used. However, we will only consider general-purpose algorithms.

Two iterative methods for zero-finding are the Newton-Raphson method and the Secant method. Both methods require that f be twice continuously differentiable. Newton-Raphson also requires that it be possible to compute the second derivative of $f(x + \alpha d)$ with respect to α .

The Newton-Raphson method relies on the Taylor series approximation

$$f(x + \alpha d) \approx f(x) + \alpha \left[\frac{d}{d\alpha} f(x + \alpha d) \right]_{\alpha=0} + \frac{\alpha^2}{2} \left[\frac{d^2}{d\alpha^2} f(x + \alpha d) \right]_{\alpha=0} \quad (56)$$

$$= f(x) + \alpha [f'(x)]^T d + \frac{\alpha^2}{2} d^T f''(x) d$$

$$\frac{d}{d\alpha} f(x + \alpha d) \approx [f'(x)]^T d + \alpha d^T f''(x) d. \quad (57)$$

where $f''(x)$ is the *Hessian matrix*

$$f''(x) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix}.$$

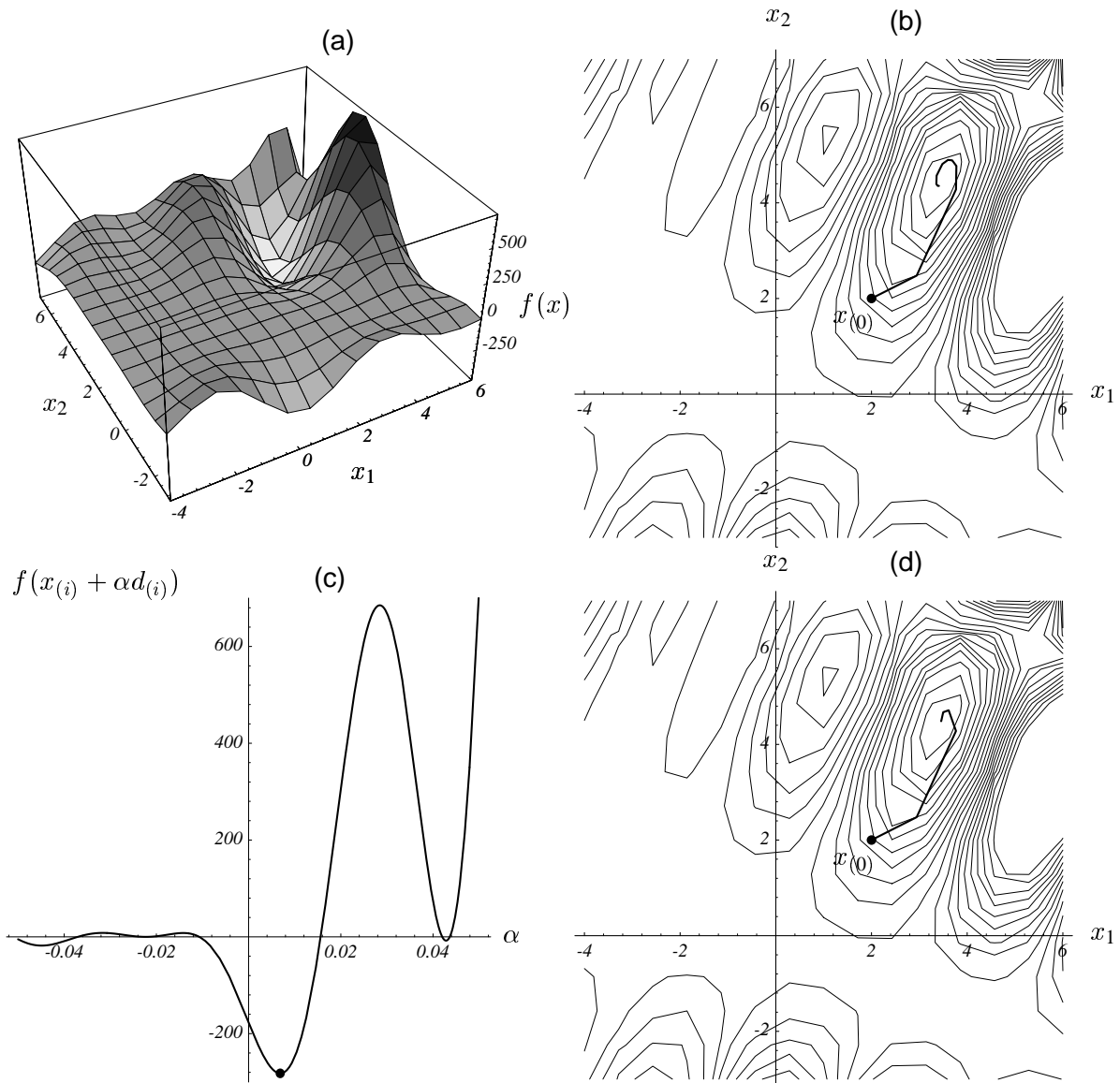


Figure 37: Convergence of the nonlinear Conjugate Gradient Method. (a) A complicated function with many local minima and maxima. (b) Convergence path of Fletcher-Reeves CG. Unlike linear CG, convergence does not occur in two steps. (c) Cross-section of the surface corresponding to the first line search. (d) Convergence path of Polak-Ribière CG.

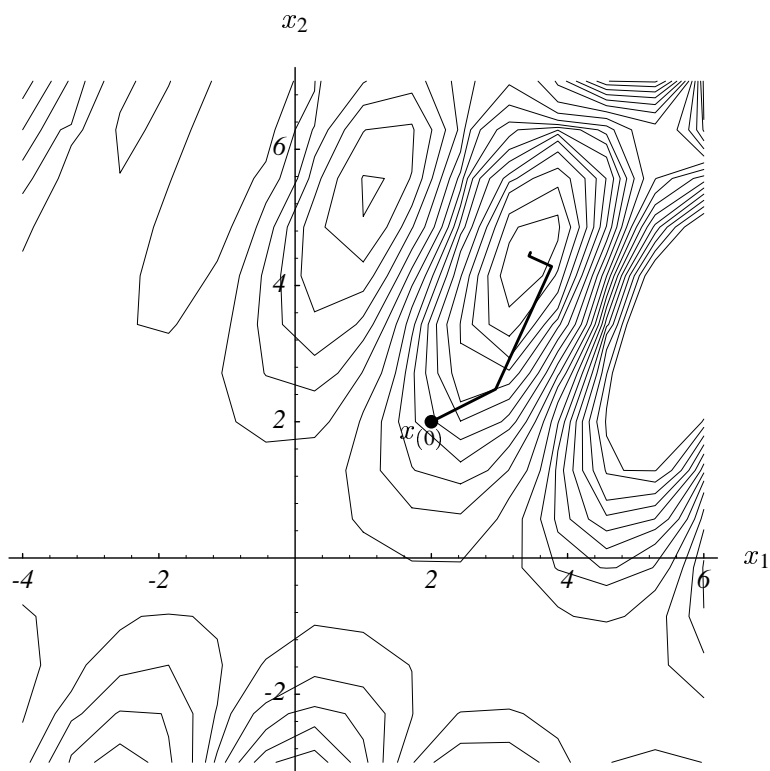


Figure 38: Nonlinear CG can be more effective with periodic restarts.

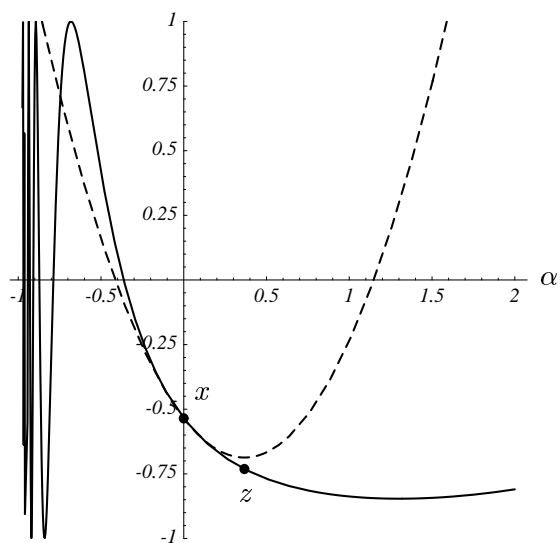


Figure 39: The Newton-Raphson method for minimizing a one-dimensional function (solid curve). Starting from the point x , calculate the first and second derivatives, and use them to construct a quadratic approximation to the function (dashed curve). A new point z is chosen at the base of the parabola. This procedure is iterated until convergence is reached.

The function $f(x + \alpha d)$ is approximately minimized by setting Expression 57 to zero, giving

$$\alpha = -\frac{f'^T d}{d^T f'' d}.$$

The truncated Taylor series approximates $f(x + \alpha d)$ with a parabola; we step to the bottom of the parabola (see Figure 39). In fact, if f is a quadratic form, then this parabolic approximation is exact, because f'' is just the familiar matrix A . In general, the search directions are *conjugate* if they are f'' -orthogonal. The meaning of “conjugate” keeps changing, though, because f'' varies with x . The more quickly f'' varies with x , the more quickly the search directions lose conjugacy. On the other hand, the closer $x_{(i)}$ is to the solution, the less f'' varies from iteration to iteration. The closer the starting point is to the solution, the more similar the convergence of nonlinear CG is to that of linear CG.

To perform an exact line search of a non-quadratic function, repeated steps must be taken along the line until $f'^T d$ is zero; hence, one CG iteration may include many Newton-Raphson iterations. The values of $f'^T d$ and $d^T f'' d$ must be evaluated at each step. These evaluations may be inexpensive if $d^T f'' d$ can be analytically simplified, but if the full matrix f'' must be evaluated repeatedly, the algorithm is prohibitively slow. For some applications, it is possible to circumvent this problem by performing an approximate line search that uses only the diagonal elements of f'' . Of course, there are functions for which it is not possible to evaluate f'' at all.

To perform an exact line search without computing f'' , the Secant method approximates the second derivative of $f(x + \alpha d)$ by evaluating the first derivative at the distinct points $\alpha = 0$ and $\alpha = \sigma$, where σ is an arbitrary small nonzero number:

$$\begin{aligned} \frac{d^2}{d\alpha^2} f(x + \alpha d) &\approx \frac{[\frac{d}{d\alpha} f(x + \alpha d)]_{\alpha=\sigma} - [\frac{d}{d\alpha} f(x + \alpha d)]_{\alpha=0}}{\sigma} && \sigma \neq 0 \\ &= \frac{[f'(x + \sigma d)]^T d - [f'(x)]^T d}{\sigma}, \end{aligned} \quad (58)$$

which becomes a better approximation to the second derivative as α and σ approach zero. If we substitute Expression 58 for the third term of the Taylor expansion (Equation 56), we have

$$\frac{d}{d\alpha} f(x + \alpha d) \approx [f'(x)]^T d + \frac{\alpha}{\sigma} \{ [f'(x + \sigma d)]^T d - [f'(x)]^T d \}.$$

Minimize $f(x + \alpha d)$ by setting its derivative to zero:

$$\alpha = -\sigma \frac{[f'(x)]^T d}{[f'(x + \sigma d)]^T d - [f'(x)]^T d} \quad (59)$$

Like Newton-Raphson, the Secant method also approximates $f(x + \alpha d)$ with a parabola, but instead of choosing the parabola by finding the first and second derivative at a point, it finds the first derivative at two different points (see Figure 40). Typically, we will choose an arbitrary σ on the first Secant method iteration; on subsequent iterations we will choose $x + \sigma d$ to be the value of x from the previous Secant method iteration. In other words, if we let $\alpha_{[i]}$ denote the value of α calculated during Secant iteration i , then $\sigma_{[i+1]} = -\alpha_{[i]}$.

Both the Newton-Raphson and Secant methods should be terminated when x is reasonably close to the exact solution. Demanding too little precision could cause a failure of convergence, but demanding too fine precision makes the computation unnecessarily slow and gains nothing, because conjugacy will break

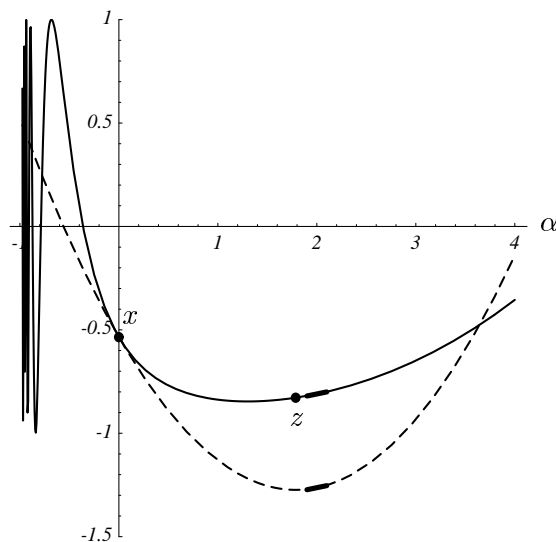


Figure 40: The Secant method for minimizing a one-dimensional function (solid curve). Starting from the point x , calculate the first derivatives at two different points (here, $\alpha = 0$ and $\alpha = 2$), and use them to construct a quadratic approximation to the function (dashed curve). Notice that both curves have the same slope at $\alpha = 0$, and also at $\alpha = 2$. As in the Newton-Raphson method, a new point z is chosen at the base of the parabola, and the procedure is iterated until convergence.

down quickly anyway if $f''(x)$ varies much with x . Therefore, a quick but inexact line search is often the better policy (for instance, use only a fixed number of Newton-Raphson or Secant method iterations). Unfortunately, inexact line search may lead to the construction of a search direction that is not a descent direction. A common solution is to test for this eventuality (is $r^T d$ nonpositive?), and restart CG if necessary by setting $d = r$.

A bigger problem with both methods is that they cannot distinguish minima from maxima. The result of nonlinear CG generally depends strongly on the starting point, and if CG with the Newton-Raphson or Secant method starts near a local maximum, it is likely to converge to that point.

Each method has its own advantages. The Newton-Raphson method has a better convergence rate, and is to be preferred if $d^T f'' d$ can be calculated (or well approximated) quickly (i.e., in $\mathcal{O}(n)$ time). The Secant method only requires first derivatives of f , but its success may depend on a good choice of the parameter σ . It is easy to derive a variety of other methods as well. For instance, by sampling f at three different points, it is possible to generate a parabola that approximates $f(x + \alpha d)$ without the need to calculate even a first derivative of f .

14.3. Preconditioning

Nonlinear CG can be preconditioned by choosing a preconditioner M that approximates f'' and has the property that $M^{-1}r$ is easy to compute. In linear CG, the preconditioner attempts to transform the quadratic form so that it is similar to a sphere; a nonlinear CG preconditioner performs this transformation for a region near $x_{(i)}$.

Even when it is too expensive to compute the full Hessian f'' , it is often reasonable to compute its diagonal

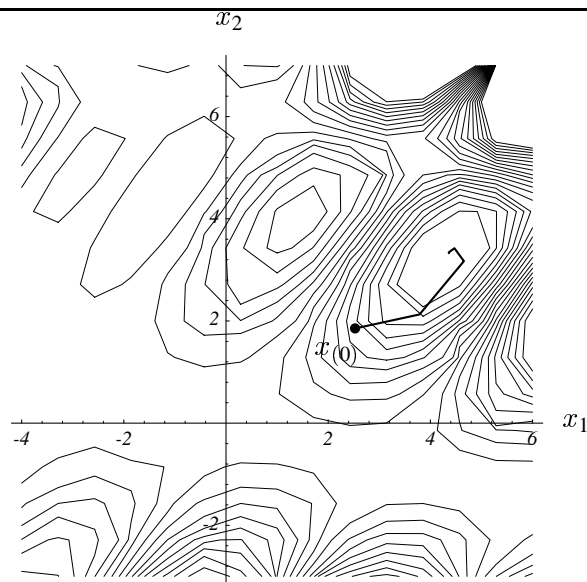


Figure 41: The preconditioned nonlinear Conjugate Gradient Method, using the Polak-Ribière formula and a diagonal preconditioner. The space has been “stretched” to show the improvement in circularity of the contour lines around the minimum.

for use as a preconditioner. However, be forewarned that if x is sufficiently far from a local minimum, the diagonal elements of the Hessian may not all be positive. A preconditioner should be positive-definite, so nonpositive diagonal elements cannot be allowed. A conservative solution is to not precondition (set $M = I$) when the Hessian cannot be guaranteed to be positive-definite. Figure 41 demonstrates the convergence of diagonally preconditioned nonlinear CG, with the Polak-Ribière formula, on the same function illustrated in Figure 37. Here, I have cheated by using the diagonal of f'' at the solution point x to precondition every iteration.

A Notes

Conjugate Direction methods were probably first presented by Schmidt [14] in 1908, and were independently reinvented by Fox, Huskey, and Wilkinson [7] in 1948. In the early fifties, the method of Conjugate Gradients was discovered independently by Hestenes [10] and Stiefel [15]; shortly thereafter, they jointly published what is considered the seminal reference on CG [11]. Convergence bounds for CG in terms of Chebyshev polynomials were developed by Kaniel [12]. A more thorough analysis of CG convergence is provided by van der Sluis and van der Vorst [16]. CG was popularized as an iterative method for large, sparse matrices by Reid [13] in 1971.

CG was generalized to nonlinear problems in 1964 by Fletcher and Reeves [6], based on work by Davidon [4] and Fletcher and Powell [5]. Convergence of nonlinear CG with inexact line searches was analyzed by Daniel [3]. The choice of β for nonlinear CG is discussed by Gilbert and Nocedal [8].

A history and extensive annotated bibliography of CG to the mid-seventies is provided by Golub and O’Leary [9]. Most research since that time has focused on nonsymmetric systems. A survey of iterative methods for the solution of linear systems is offered by Barrett et al. [1].

B Canned Algorithms

The code given in this section represents efficient implementations of the algorithms discussed in this article.

B1. Steepest Descent

Given the inputs A , b , a starting value x , a maximum number of iterations i_{max} , and an error tolerance $\varepsilon < 1$:

```

i ← 0
r ← b − Ax
δ ← rTr
δ0 ← δ
While i < imax and δ > ε2δ0 do
  q ← Ar
  α ←  $\frac{\delta}{r^T q}$ 
  x ← x + αr
  If i is divisible by 50
    r ← b − Ax
  else
    r ← r − αq
  δ ← rTr
  i ← i + 1

```

This algorithm terminates when the maximum number of iterations i_{max} has been exceeded, or when $\|r_{(i)}\| \leq \varepsilon \|r_{(0)}\|$.

The fast recursive formula for the residual is usually used, but once every 50 iterations, the exact residual is recalculated to remove accumulated floating point error. Of course, the number 50 is arbitrary; for large n , \sqrt{n} might be appropriate. If the tolerance is large, the residual need not be corrected at all (in practice, this correction is rarely used). If the tolerance is close to the limits of the floating point precision of the machine, a test should be added after δ is evaluated to check if $\delta \leq \varepsilon^2 \delta_0$, and if this test holds true, the exact residual should also be recomputed and δ reevaluated. This prevents the procedure from terminating early due to floating point roundoff error.

B2. Conjugate Gradients

Given the inputs A , b , a starting value x , a maximum number of iterations i_{max} , and an error tolerance $\varepsilon < 1$:

```
 $i \leftarrow 0$   
 $r \leftarrow b - Ax$   
 $d \leftarrow r$   
 $\delta_{new} \leftarrow r^T r$   
 $\delta_0 \leftarrow \delta_{new}$   
While  $i < i_{max}$  and  $\delta_{new} > \varepsilon^2 \delta_0$  do  
   $q \leftarrow Ad$   
   $\alpha \leftarrow \frac{\delta_{new}}{d^T q}$   
   $x \leftarrow x + \alpha d$   
  If  $i$  is divisible by 50  
     $r \leftarrow b - Ax$   
  else  
     $r \leftarrow r - \alpha q$   
   $\delta_{old} \leftarrow \delta_{new}$   
   $\delta_{new} \leftarrow r^T r$   
   $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$   
   $d \leftarrow r + \beta d$   
   $i \leftarrow i + 1$ 
```

See the comments at the end of Section B1.

B3. Preconditioned Conjugate Gradients

Given the inputs A , b , a starting value x , a (perhaps implicitly defined) preconditioner M , a maximum number of iterations i_{max} , and an error tolerance $\varepsilon < 1$:

```

i ← 0
r ← b − Ax
d ← M−1r
δnew ← rTd
δ0 ← δnew
While i < imax and δnew > ε2δ0 do
    q ← Ad
    α ←  $\frac{\delta_{new}}{d^T q}$ 
    x ← x + αd
    If i is divisible by 50
        r ← b − Ax
    else
        r ← r − αq
    s ← M−1r
    δold ← δnew
    δnew ← rTs
    β ←  $\frac{\delta_{new}}{\delta_{old}}$ 
    d ← s + βd
    i ← i + 1

```

The statement “ $s \leftarrow M^{-1}r$ ” implies that one should apply the preconditioner, which may not actually be in the form of a matrix.

See also the comments at the end of Section B1.

B4. Nonlinear Conjugate Gradients with Newton-Raphson and Fletcher-Reeves

Given a function f , a starting value x , a maximum number of CG iterations i_{max} , a CG error tolerance $\epsilon < 1$, a maximum number of Newton-Raphson iterations j_{max} , and a Newton-Raphson error tolerance $\epsilon < 1$:

```

i ← 0
k ← 0
r ← -f'(x)
d ← r
δnew ← rTr
δ0 ← δnew
While i < imax and δnew >  $\epsilon^2 \delta_0$  do
  j ← 0
  δd ← dTd
  Do
     $\alpha \leftarrow -\frac{[f'(x)]^T d}{d^T f''(x) d}$ 
    x ← x +  $\alpha d$ 
    j ← j + 1
  while j < jmax and  $\alpha^2 \delta_d > \epsilon^2$ 
  r ← -f'(x)
  δold ← δnew
  δnew ← rTr
   $\beta \leftarrow \frac{\delta_{new}}{\delta_{old}}$ 
  d ← r +  $\beta d$ 
  k ← k + 1
  If k = n or rTd ≤ 0
    d ← r
    k ← 0
  i ← i + 1

```

This algorithm terminates when the maximum number of iterations i_{max} has been exceeded, or when $\|r_{(i)}\| \leq \epsilon \|r_{(0)}\|$.

Each Newton-Raphson iteration adds αd to x ; the iterations are terminated when each update αd falls below a given tolerance ($\|\alpha d\| \leq \epsilon$), or when the number of iterations exceeds j_{max} . A fast inexact line search can be accomplished by using a small j_{max} and/or by approximating the Hessian $f''(x)$ with its diagonal.

Nonlinear CG is restarted (by setting $d \leftarrow r$) whenever a search direction is computed that is not a descent direction. It is also restarted once every n iterations, to improve convergence for small n .

The calculation of α may result in a divide-by-zero error. This may occur because the starting point $x_{(0)}$ is not sufficiently close to the desired minimum, or because f is not twice continuously differentiable. In the former case, the solution is to choose a better starting point or a more sophisticated line search. In the latter case, CG might not be the most appropriate minimization algorithm.

B5. Preconditioned Nonlinear Conjugate Gradients with Secant and Polak-Ribière

Given a function f , a starting value x , a maximum number of CG iterations i_{max} , a CG error tolerance $\varepsilon < 1$, a Secant method step parameter σ_0 , a maximum number of Secant method iterations j_{max} , and a Secant method error tolerance $\epsilon < 1$:

```

i ← 0
k ← 0
r ← -f'(x)
Calculate a preconditioner  $M \approx f''(x)$ 
s ←  $M^{-1}r$ 
d ← s
 $\delta_{new} \leftarrow r^T d$ 
 $\delta_0 \leftarrow \delta_{new}$ 
While i <  $i_{max}$  and  $\delta_{new} > \varepsilon^2 \delta_0$  do
  j ← 0
   $\delta_d \leftarrow d^T d$ 
   $\alpha \leftarrow -\sigma_0$ 
   $\eta_{prev} \leftarrow [f'(x + \sigma_0 d)]^T d$ 
  Do
     $\eta \leftarrow [f'(x)]^T d$ 
     $\alpha \leftarrow \alpha \frac{\eta}{\eta_{prev} - \eta}$ 
    x ← x +  $\alpha d$ 
     $\eta_{prev} \leftarrow \eta$ 
    j ← j + 1
  while j <  $j_{max}$  and  $\alpha^2 \delta_d > \epsilon^2$ 
  r ← -f'(x)
   $\delta_{old} \leftarrow \delta_{new}$ 
   $\delta_{mid} \leftarrow r^T s$ 
  Calculate a preconditioner  $M \approx f''(x)$ 
  s ←  $M^{-1}r$ 
   $\delta_{new} \leftarrow r^T s$ 
   $\beta \leftarrow \frac{\delta_{new} - \delta_{mid}}{\delta_{old}}$ 
  k ← k + 1
  If k = n or  $\beta \leq 0$ 
    d ← s
    k ← 0
  else
    d ← s +  $\beta d$ 
  i ← i + 1

```

This algorithm terminates when the maximum number of iterations i_{max} has been exceeded, or when $\|r_{(i)}\| \leq \varepsilon \|r_{(0)}\|$.

Each Secant method iteration adds αd to x ; the iterations are terminated when each update αd falls below a given tolerance ($\|\alpha d\| \leq \epsilon$), or when the number of iterations exceeds j_{max} . A fast inexact line search can be accomplished by using a small j_{max} . The parameter σ_0 determines the value of σ in Equation 59 for the first step of each Secant method minimization. Unfortunately, it may be necessary to adjust this parameter

to achieve convergence.

The Polak-Ribière β parameter is $\frac{\delta_{new} - \delta_{mid}}{\delta_{old}} = \frac{r_{(i+1)}^T s_{(i+1)} - r_{(i+1)}^T s_{(i)}}{r_{(i)}^T s_{(i)}} = \frac{r_{(i+1)}^T M^{-1}(r_{(i+1)} - r_{(i)})}{r_{(i)}^T M^{-1} r_{(i)}}$. Care must be taken that the preconditioner M is always positive-definite. The preconditioner is not necessarily in the form of a matrix.

Nonlinear CG is restarted (by setting $d \leftarrow r$) whenever the Polak-Ribière β parameter is negative. It is also restarted once every n iterations, to improve convergence for small n .

Nonlinear CG presents several choices: Preconditioned or not, Newton-Raphson method or Secant method or another method, Fletcher-Reeves or Polak-Ribière. It should be possible to construct any of the variations from the versions presented above. (Polak-Ribière is almost always to be preferred.)

C Ugly Proofs

C1. The Solution to $Ax = b$ Minimizes the Quadratic Form

Suppose A is symmetric. Let x be a point that satisfies $Ax = b$ and minimizes the quadratic form (Equation 3), and let e be an error term. Then

$$\begin{aligned} f(x + e) &= \frac{1}{2}(x + e)^T A(x + e) - b^T(x + e) + c && \text{(by Equation 3)} \\ &= \frac{1}{2}x^T Ax + e^T Ax + \frac{1}{2}e^T Ae - b^T x - b^T e + c && \text{(by symmetry of } A\text{)} \\ &= \frac{1}{2}x^T Ax - b^T x + c + e^T b + \frac{1}{2}e^T Ae - b^T e \\ &= f(x) + \frac{1}{2}e^T Ae. \end{aligned}$$

If A is positive-definite, then the latter term is positive for all $e \neq 0$; therefore x minimizes f .

C2. A Symmetric Matrix Has n Orthogonal Eigenvectors.

Any matrix has at least one eigenvector. To see this, note that $\det(A - \lambda I)$ is a polynomial in λ , and therefore has at least one zero (possibly complex); call it λ_A . The matrix $A - \lambda_A I$ has determinant zero and is therefore singular, so there must be a nonzero vector v for which $(A - \lambda_A I)v = 0$. The vector v is an eigenvector, because $Av = \lambda_A v$.

Any symmetric matrix has a full complement of n orthogonal eigenvectors. To prove this, I will demonstrate for the case of a 4×4 matrix A and let you make the obvious generalization to matrices of arbitrary size. It has already been proven that A has at least one eigenvector v with eigenvalue λ_A . Let $x_1 = v/\|v\|$, which has unit length. Choose three arbitrary vectors x_2, x_3, x_4 so that the x_i are mutually orthogonal and all of unit length (such vectors can be found by Gram-Schmidt orthogonalization). Let $X = [x_1 \ x_2 \ x_3 \ x_4]$. Because the x_i are orthonormal, $X^T X = I$, and so $X^T = X^{-1}$. Note also that for

$i \neq 1$, $x_1^T A x_i = x_i^T A x_1 = x_i^T \lambda_A x_1 = 0$. Thus,

$$\begin{aligned} X^T A X &= \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \end{bmatrix} A \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \end{bmatrix} \begin{bmatrix} \lambda_A x_1 & A x_2 & A x_3 & A x_4 \end{bmatrix} \\ &= \begin{bmatrix} \lambda_A & 0 & 0 & 0 \\ 0 & & & \\ 0 & & B & \\ 0 & & & \end{bmatrix}, \end{aligned}$$

where B is a 3×3 symmetric matrix. B must have some eigenvalue w with eigenvalue λ_B . Let \hat{w} be a vector of length 4 whose first element is zero, and whose remaining elements are the elements of w . Clearly,

$$X^{-1} A X \hat{w} = X^T A X \hat{w} = \begin{bmatrix} \lambda_A & 0 & 0 & 0 \\ 0 & & & \\ 0 & & B & \\ 0 & & & \end{bmatrix} \hat{w} = \lambda_B \hat{w}.$$

In other words, $A X \hat{w} = \lambda_B X \hat{w}$, and thus $X \hat{w}$ is an eigenvector of A . Furthermore, $x_1^T X \hat{w} = [1 \ 0 \ 0 \ 0] \hat{w} = 0$, so x_1 and $X \hat{w}$ are orthogonal. Thus, A has at least two orthogonal eigenvectors!

The more general statement that a symmetric matrix has n orthogonal eigenvectors is proven by induction. For the example above, assume any 3×3 matrix (such as B) has 3 orthogonal eigenvectors; call them \hat{w}_1 , \hat{w}_2 , and \hat{w}_3 . Then $X \hat{w}_1$, $X \hat{w}_2$, and $X \hat{w}_3$ are eigenvectors of A , and they are orthogonal because X has orthonormal columns, and therefore maps orthogonal vectors to orthogonal vectors. Thus, A has 4 orthogonal eigenvectors.

C3. Optimality of Chebyshev Polynomials

Chebyshev polynomials are optimal for minimization of expressions like Expression 50 because they increase in magnitude more quickly outside the range $[-1, 1]$ than any other polynomial that is restricted to have magnitude no greater than one inside the range $[-1, 1]$.

The Chebyshev polynomial of degree i ,

$$T_i(\omega) = \frac{1}{2} \left[(\omega + \sqrt{\omega^2 - 1})^i + (\omega - \sqrt{\omega^2 - 1})^i \right],$$

can also be expressed on the region $[-1, 1]$ as

$$T_i(\omega) = \cos(i \cos^{-1} \omega), \quad -1 \leq \omega \leq 1.$$

From this expression (and from Figure 32), one can deduce that the Chebyshev polynomials have the property

$$|T_i(\omega)| \leq 1, \quad -1 \leq \omega \leq 1$$

and oscillate rapidly between -1 and 1 :

$$T_i \left(\cos \left(\frac{k\pi}{i} \right) \right) = (-1)^k, \quad k = 0, 1, \dots, i.$$