An Introduction to the Conjugate Gradient Method Without the Agonizing Pain

Edition $1\frac{1}{4}$

Jonathan Richard Shewchuk

August 4, 1994

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

The Conjugate Gradient Method is the most prominent iterative method for solving sparse systems of linear equations. Unfortunately, many textbook treatments of the topic are written with neither illustrations nor intuition, and their victims can be found to this day babbling senselessly in the corners of dusty libraries. For this reason, a deep, geometric understanding of the method has been reserved for the elite brilliant few who have painstakingly decoded the mumblings of their forebears. Nevertheless, the Conjugate Gradient Method is a composite of simple, elegant ideas that almost anyone can understand. Of course, a reader as intelligent as yourself will learn them almost effortlessly.

The idea of quadratic forms is introduced and used to derive the methods of Steepest Descent, Conjugate Directions, and Conjugate Gradients. Eigenvectors are explained and used to examine the convergence of the Jacobi Method, Steepest Descent, and Conjugate Gradients. Other topics include preconditioning and the nonlinear Conjugate Gradient Method. I have taken pains to make this article easy to read. Sixty-six illustrations are provided. Dense prose is avoided. Concepts are explained in several different ways. Most equations are coupled with an intuitive interpretation.

Supported in part by the Natural Sciences and Engineering Research Council of Canada under a 1967 Science and Engineering Scholarship and by the National Science Foundation under Grant ASC-9318163. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either express or implied, of NSERC, NSF, or the U.S. Government.

Keywords: conjugate gradient method, preconditioning, convergence analysis, agonizing pain

Contents

About this Article

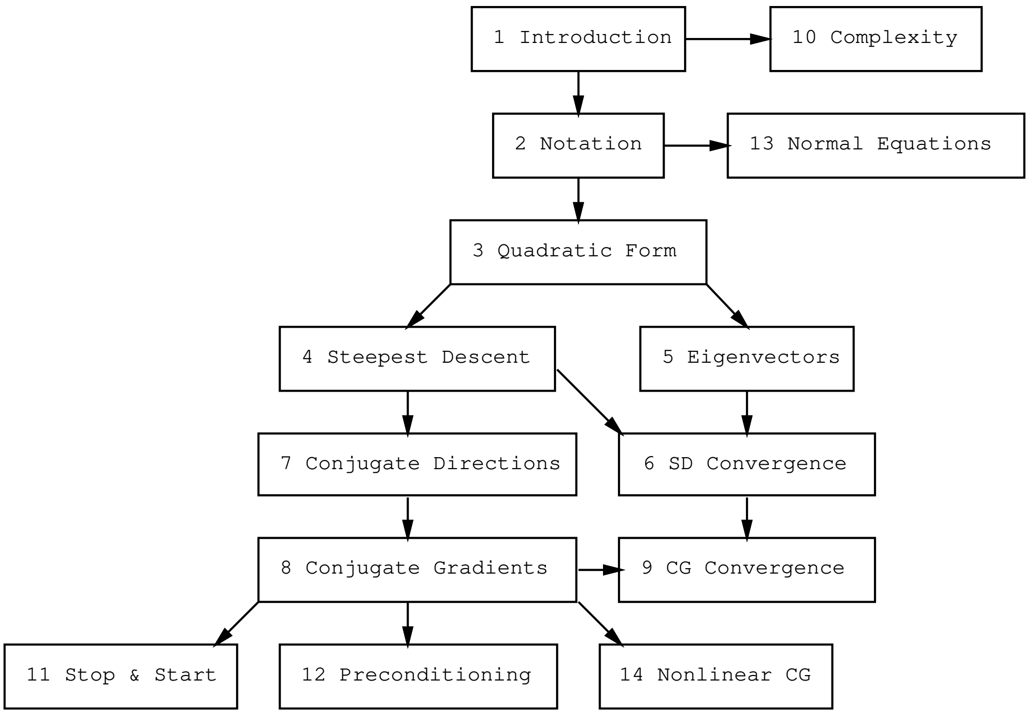
An electronic copy of this article is available by anonymous FTP to WARP.CS.CMU.EDU (IP address 128.2.209.103) under the filename quake-papers/painless-conjugate-gradient.ps. A PostScript file containing full-page copies of the figures herein, suitable for transparencies, is available as quake-papers/painless-conjugate-gradient-pics.ps. Most of the illustrations were created using Mathematica.

This guide was created to help students learn Conjugate Gradient Methods as easily as possible. Please mail me (jrs@cs.cmu.edu) comments, corrections, and any intuitions I might have missed; some of these will be incorporated into a second edition. I am particularly interested in hearing about use of this guide for classroom teaching.

For those who wish to learn more about iterative methods, I recommend William L. Briggs' "A Multigrid Tutorial" [2], one of the best-written mathematical books I have read.

Special thanks to Omar Ghattas, who taught me much of what I know about numerical methods, and provided me with extensive comments on the first draft of this article. Thanks also to James Epperson, David O'Hallaron, James Stichnoth, Nick Trefethen, and Daniel Tunkelang for their comments.

To help you skip chapters, here's a dependence graph of the sections:



This article is dedicated to every mathematician who uses figures as abundantly as I have herein.

1. Introduction

When I decided to learn the Conjugate Gradient Method (henceforth, CG), I read four different descriptions, which I shall politely not identify. I understood none of them. Most of them simply wrote down the method, then proved its properties without any intuitive explanation or hint of how anybody might have invented CG in the first place. This article was born of my frustration, with the wish that future students of CG will learn a rich and elegant algorithm, rather than a confusing mass of equations.

CG is the most popular iterative method for solving large systems of linear equations. CG is effective for systems of the form

$$Ax = b \tag{1}$$

where x is an unknown vector, b is a known vector, and A is a known, square, symmetric, positive-definite (or positive-indefinite) matrix. (Don't worry if you've forgotten what "positive-definite" means; we shall review it.) These systems arise in many important settings, such as finite difference and finite element methods for solving partial differential equations, structural analysis, circuit analysis, and math homework.

Iterative methods like CG are suited for use with sparse matrices. If A is dense, your best course of action is probably to factor A and solve the equation by backsubstitution. The time spent factoring a dense A is roughly equivalent to the time spent solving the system iteratively; and once A is factored, the system can be backsolved quickly for multiple values of b. Compare this dense matrix with a sparse matrix of larger size that fills the same amount of memory. The triangular factors of a sparse A usually have many more nonzero elements than A itself. Factoring may be impossible due to limited memory, and will be time-consuming as well; even the backsolving step may be slower than iterative solution. On the other hand, most iterative methods are memory-efficient and run quickly with sparse matrices.

I assume that you have taken a first course in linear algebra, and that you have a solid understanding of matrix multiplication and linear independence, although you probably don't remember what those eigenthingies were all about. From this foundation, I shall build the edifice of CG as clearly as I can.

2. Notation

Let us begin with a few definitions and notes on notation.

With a few exceptions, I shall use capital letters to denote matrices, lower case letters to denote vectors, and Greek letters to denote scalars. A is an $n \times n$ matrix, and x and b are vectors — that is, $n \times 1$ matrices. Written out fully, Equation 1 is

$$\begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & & A_{2n} \\ \vdots & & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.$$

The *inner product* of two vectors is written $x^T y$, and represents the scalar sum $\sum_{i=1}^{n} x_i y_i$. Note that $x^T y = y^T x$. If x and y are orthogonal, then $x^T y = 0$. In general, expressions that reduce to 1×1 matrices, such as $x^T y$ and $x^T A x$, are treated as scalar values.

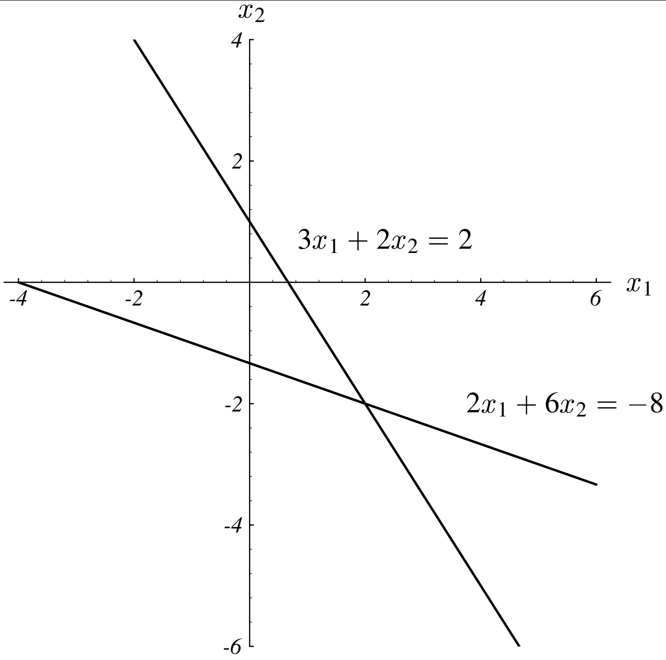


Figure 1: Sample two-dimensional linear system. The solution lies at the intersection of the lines.

A matrix A is *positive-definite* if, for every nonzero vector x,

$$x^T A x > 0. \tag{2}$$

This may mean little to you, but don't feel bad; it's not a very intuitive idea, and it's hard to imagine how a matrix that is positive-definite might look differently from one that isn't. We will get a feeling for what positive-definiteness is about when we see how it affects the shape of quadratic forms.

Finally, don't forget the important basic identities $(AB)^T = B^T A^T$ and $(AB)^{-1} = B^{-1} A^{-1}$.

3. The Quadratic Form

A *quadratic form* is simply a scalar, quadratic function of a vector with the form

$$f(x) = \frac{1}{2} x^T A x - b^T x + c \tag{3}$$

where A is a matrix, x and b are vectors, and c is a scalar constant. I shall show shortly that if A is symmetric and positive-definite, $f(x)$ is minimized by the solution to $Ax = b$.

Throughout this paper, I will demonstrate ideas with the simple sample problem

$$A = \begin{bmatrix} 3 & 2 \\ 2 & 6 \end{bmatrix}, \qquad b = \begin{bmatrix} 2 \\ -8 \end{bmatrix}, \qquad c = 0. \tag{4}$$

The system $Ax = b$ is illustrated in Figure 1. In general, the solution x lies at the intersection point of n hyperplanes, each having dimension $n - 1$. For this problem, the solution is $x = [2, -2]^T$. The corresponding quadratic form $f(x)$ appears in Figure 2. A contour plot of $f(x)$ is illustrated in Figure 3.

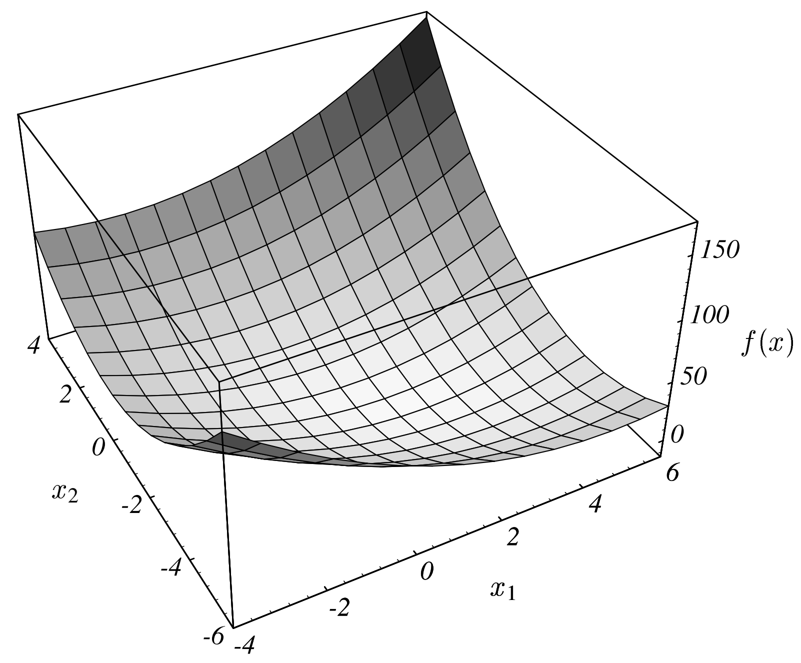


Figure 2: Graph of a quadratic form $f(x)$. The minimum point of this surface is the solution to $Ax = b$.

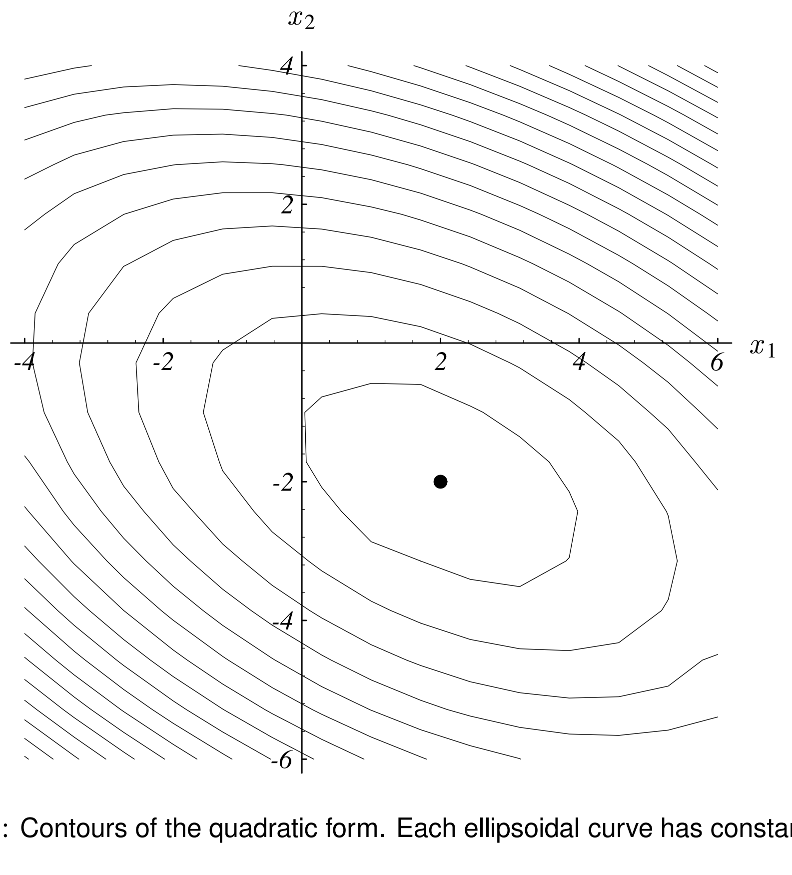


Figure 3: Contours of the quadratic form. Each ellipsoidal curve has constant $f(x)$.

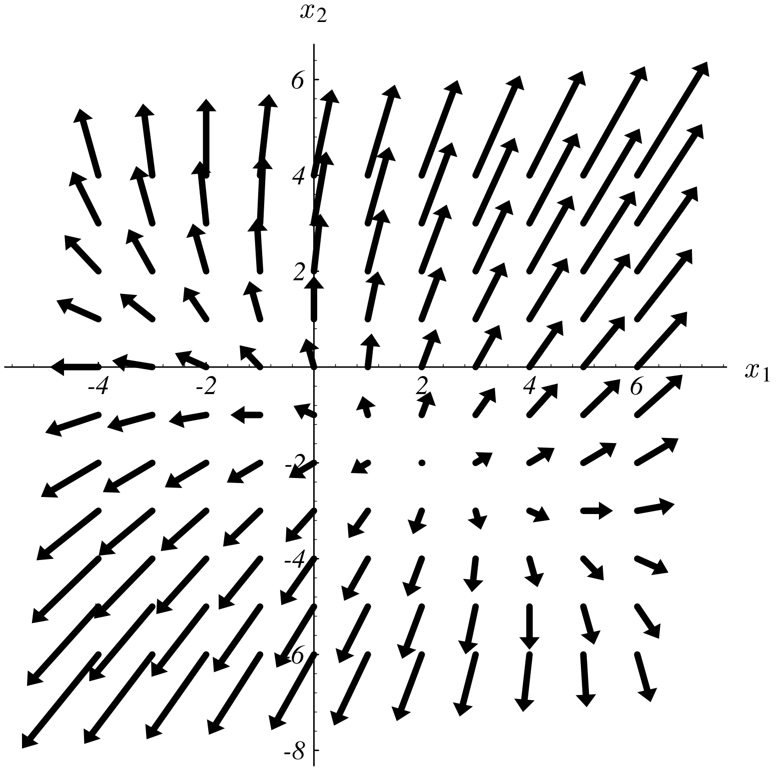


Figure 4: Gradient $f'(x)$ of the quadratic form. For every x, the gradient points in the direction of steepest increase of $f(x)$, and is orthogonal to the contour lines.

Because A is positive-definite, the surface defined by $f(x)$ is shaped like a paraboloid bowl. (I'll have more to say about this in a moment.)

The *gradient* of a quadratic form is defined to be

$$f'(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix}. \tag{5}$$

The gradient is a vector field that, for a given point x, points in the direction of greatest increase of $f(x)$. Figure 4 illustrates the gradient vectors for Equation 3 with the constants given in Equation 4. At the bottom of the paraboloid bowl, the gradient is zero. One can minimize $f(x)$ by setting $f'(x)$ equal to zero.

With a little bit of tedious math, one can apply Equation 5 to Equation 3, and derive

$$f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b. \tag{6}$$

If A is symmetric, this equation reduces to

$$f'(x) = Ax - b. \tag{7}$$

Setting the gradient to zero, we obtain Equation 1, the linear system we wish to solve. Therefore, the solution to $Ax = b$ is a critical point of $f(x)$. If A is positive-definite as well as symmetric, then this

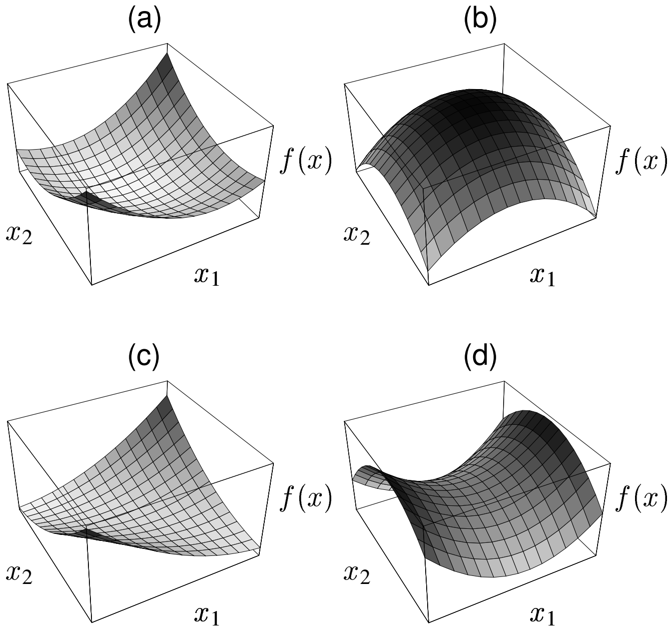


Figure 5: (a) Quadratic form for a positive-definite matrix. (b) For a negative-definite matrix. (c) For a singular (and positive-indefinite) matrix. A line that runs through the bottom of the valley is the set of solutions. (d) For an indefinite matrix. Because the solution is a saddle point, Steepest Descent and CG will not work. In three dimensions or higher, a singular matrix can also have a saddle.

solution is a minimum of $f(x)$, so $Ax = b$ can be solved by finding an x that minimizes $f(x)$. (If A is not symmetric, then Equation 6 hints that CG will find a solution to the system $\frac{1}{2}(A^T + A)x = b$. Note that $\frac{1}{2}(A^T + A)$ is symmetric.)

Why do symmetric positive-definite matrices have this nice property? Consider the relationship between f at some arbitrary point p and at the solution point $x = A^{-1}b$. From Equation 3 one can show (Appendix C1) that if A is symmetric (be it positive-definite or not),

$$f(p) = f(x) + \frac{1}{2}(p - x)^T A(p - x). \tag{8}$$

If A is positive-definite as well, then by Inequality 2, the latter term is positive for all $p \neq x$. It follows that x is a global minimum of f.

The fact that $f(x)$ is a paraboloid is our best intuition of what it means for a matrix to be positive-definite. If A is not positive-definite, there are several other possibilities. A could be negative-definite — the result of negating a positive-definite matrix (see Figure 2, but hold it upside-down). A might be singular, in which case no solution is unique; the set of solutions is a line or hyperplane having a uniform value for f. If A is none of the above, then x is a saddle point, and techniques like Steepest Descent and CG will likely fail. Figure 5 demonstrates the possibilities. The values of b and c determine where the minimum point of the paraboloid lies, but do not affect the paraboloid's shape.

Why go to the trouble of converting the linear system into a tougher-looking problem? The methods under study — Steepest Descent and CG — were developed and are intuitively understood in terms of minimization problems like Figure 2, not in terms of intersecting hyperplanes such as Figure 1.

4. The Method of Steepest Descent

In the method of Steepest Descent, we start at an arbitrary point $x_{(0)}$ and slide down to the bottom of the paraboloid. We take a series of steps $x_{(1)}, x_{(2)}, \ldots$ until we are satisfied that we are close enough to the solution x.

When we take a step, we choose the direction in which f decreases most quickly, which is the direction opposite $f'(x_{(i)})$. According to Equation 7, this direction is $-f'(x_{(i)}) = b - Ax_{(i)}$.

Allow me to introduce a few definitions, which you should memorize. The *error* $e_{(i)} = x_{(i)} - x$ is a vector that indicates how far we are from the solution. The *residual* $r_{(i)} = b - Ax_{(i)}$ indicates how far we are from the correct value of b. It is easy to see that $r_{(i)} = -Ae_{(i)}$, and you should think of the residual as being the error transformed by A into the same space as b. More importantly, $r_{(i)} = -f'(x_{(i)})$, and you should also think of the residual as the direction of steepest descent. For nonlinear problems, discussed in Section 14, only the latter definition applies. So remember, whenever you read "residual", think "direction of steepest descent."

Suppose we start at $x_{(0)} = [-2, -2]^T$. Our first step, along the direction of steepest descent, will fall somewhere on the solid line in Figure 6(a). In other words, we will choose a point

$$x_{(1)} = x_{(0)} + \alpha r_{(0)}. \tag{9}$$

The question is, how big a step should we take?

A *line search* is a procedure that chooses α to minimize f along a line. Figure 6(b) illustrates this task: we are restricted to choosing a point on the intersection of the vertical plane and the paraboloid. Figure 6(c) is the parabola defined by the intersection of these surfaces. What is the value of α at the base of the parabola?

From basic calculus, α minimizes f when the *directional derivative* $\frac{d}{d\alpha} f(x_{(1)})$ is equal to zero. By the chain rule, $\frac{d}{d\alpha} f(x_{(1)}) = f'(x_{(1)})^T \frac{d}{d\alpha} x_{(1)} = f'(x_{(1)})^T r_{(0)}$. Setting this expression to zero, we find that α should be chosen so that $r_{(0)}$ and $f'(x_{(1)})$ are orthogonal (see Figure 6(d)).

There is an intuitive reason why we should expect these vectors to be orthogonal at the minimum. Figure 7 shows the gradient vectors at various points along the search line. The slope of the parabola (Figure 6(c)) at any point is equal to the magnitude of the projection of the gradient onto the line (Figure 7, dotted arrows). These projections represent the rate of increase of f as one traverses the search line. f is minimized where the projection is zero — where the gradient is orthogonal to the search line.

To determine α, note that $f'(x_{(1)}) = -r_{(1)}$, and we have

$$\begin{aligned}
r_{(1)}^T r_{(0)} &= 0 \\
(b - Ax_{(1)})^T r_{(0)} &= 0 \\
(b - A(x_{(0)} + \alpha r_{(0)}))^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} - \alpha (Ar_{(0)})^T r_{(0)} &= 0 \\
(b - Ax_{(0)})^T r_{(0)} &= \alpha (Ar_{(0)})^T r_{(0)} \\
r_{(0)}^T r_{(0)} &= \alpha r_{(0)}^T (Ar_{(0)}) \\
\alpha &= \frac{r_{(0)}^T r_{(0)}}{r_{(0)}^T A r_{(0)}}.
\end{aligned}$$

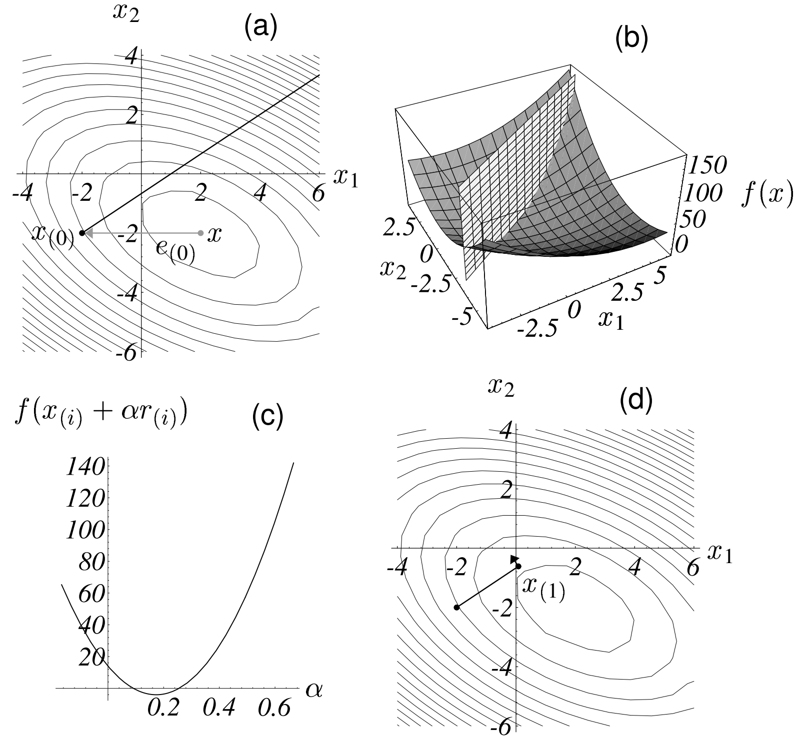


Figure 6: The method of Steepest Descent. (a) Starting at $[-2, -2]^T$, take a step in the direction of steepest descent of f. (b) Find the point on the intersection of these two surfaces that minimizes f. (c) This parabola is the intersection of surfaces. The bottommost point is our target. (d) The gradient at the bottommost point is orthogonal to the gradient of the previous step.

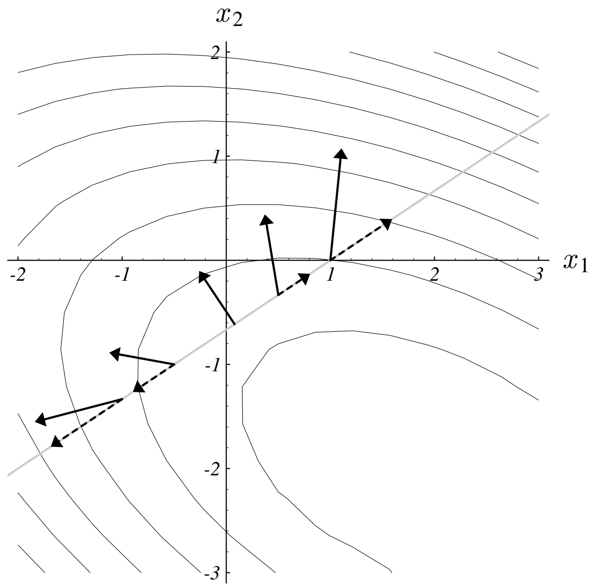


Figure 7: The gradient f' is shown at several locations along the search line (solid arrows). Each gradient's projection onto the line is also shown (dotted arrows). The gradient vectors represent the direction of steepest increase of f, and the projections represent the rate of increase as one traverses the search line. On the search line, f is minimized where the gradient is orthogonal to the search line.

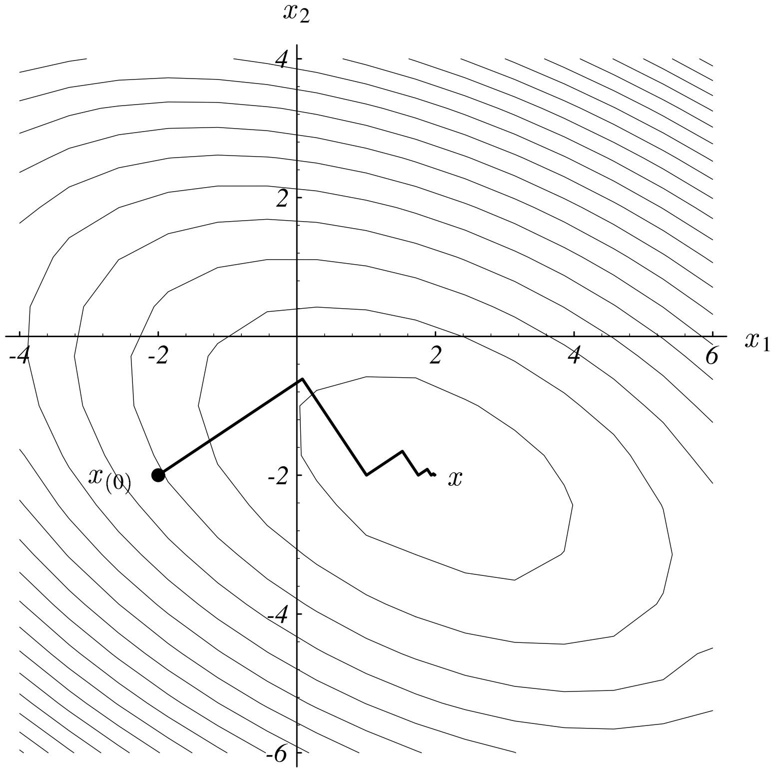


Figure 8: Here, the method of Steepest Descent starts at $[-2, -2]^T$ and converges at $[2, -2]^T$.

Putting it all together, the method of Steepest Descent is:

$$r_{(i)} = b - Ax_{(i)}, \tag{10}$$

$$\alpha_{(i)} = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}, \tag{11}$$

$$x_{(i+1)} = x_{(i)} + \alpha_{(i)} r_{(i)}. \tag{12}$$

The example is run until it converges in Figure 8. Note the zigzag path, which appears because each gradient is orthogonal to the previous gradient.

The algorithm, as written above, requires two matrix-vector multiplications per iteration. The computational cost of Steepest Descent is dominated by matrix-vector products; fortunately, one can be eliminated. By premultiplying both sides of Equation 12 by $-A$ and adding b, we have

$$r_{(i+1)} = r_{(i)} - \alpha_{(i)} A r_{(i)}. \tag{13}$$

Although Equation 10 is still needed to compute $r_{(0)}$, Equation 13 can be used for every iteration thereafter. The product Ar, which occurs in both Equations 11 and 13, need only be computed once. The disadvantage of using this recurrence is that the sequence defined by Equation 13 is generated without any feedback from the value of $x_{(i)}$, so that accumulation of floating point roundoff error may cause $x_{(i)}$ to converge to some point near x. This effect can be avoided by periodically using Equation 10 to recompute the correct residual.

Before analyzing the convergence of Steepest Descent, I must digress to ensure that you have a solid understanding of eigenvectors.

5. Thinking with Eigenvectors and Eigenvalues

After my one course in linear algebra, I knew eigenvectors and eigenvalues like the back of my head. If your instructor was anything like mine, you recall solving problems involving eigendoohickeys, but you never *really* understood them. Unfortunately, without an intuitive grasp of them, CG won't make sense either. If you're already eigentalented, feel free to skip this section.

Eigenvectors are used primarily as an analysis tool; Steepest Descent and CG do not calculate the value of any eigenvectors as part of the algorithm[1].

5.1. Eigen do it if I try

An *eigenvector* v of a matrix B is a nonzero vector that does not rotate when B is applied to it (except perhaps to point in precisely the opposite direction). v may change length or reverse its direction, but it won't turn sideways. In other words, there is some scalar constant λ such that $Bv = \lambda v$. The value λ is an *eigenvalue* of B. For any constant α, the vector αv is also an eigenvector with eigenvalue λ, because $B(\alpha v) = \alpha Bv = \lambda \alpha v$. In other words, if you scale an eigenvector, it's still an eigenvector.

Why should you care? Iterative methods often depend on applying B to a vector over and over again. When B is repeatedly applied to an eigenvector v, one of two things can happen. If $|\lambda| < 1$, then $B^i v = \lambda^i v$ will vanish as i approaches infinity (Figure 9). If $|\lambda| > 1$, then $B^i v$ will grow to infinity (Figure 10). Each time B is applied, the vector grows or shrinks according to the value of $|\lambda|$.

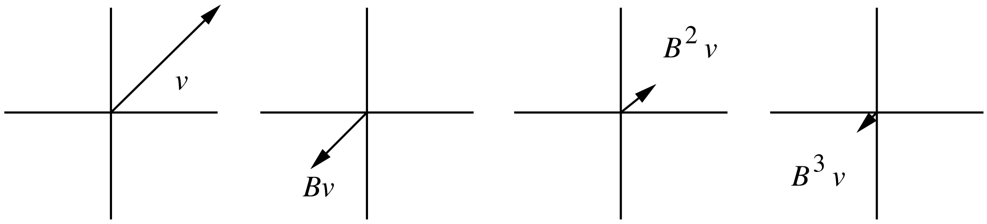


Figure 9: v is an eigenvector of B with a corresponding eigenvalue of -0.5. As i increases, $B^i v$ converges to zero.

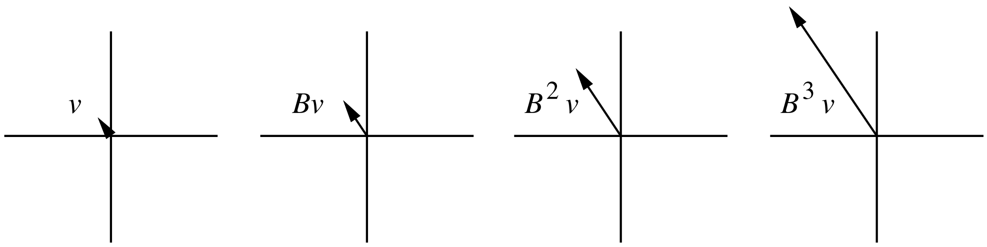


Figure 10: Here, v has a corresponding eigenvalue of 2. As i increases, $B^i v$ diverges to infinity.

[1]However, there are practical applications for eigenvectors. The eigenvectors of the stiffness matrix associated with a discretized structure of uniform density represent the natural modes of vibration of the structure being studied. For instance, the eigenvectors of the stiffness matrix associated with a one-dimensional uniformly-spaced mesh are sine waves, and to express vibrations as a linear combination of these eigenvectors is equivalent to performing a discrete Fourier transform.

If B is symmetric (and often if it is not), then there exists a set of n linearly independent eigenvectors of B, denoted $v_1, v_2, \ldots, v_n$. This set is not unique, because each eigenvector can be scaled by an arbitrary nonzero constant. Each eigenvector has a corresponding eigenvalue, denoted $\lambda_1, \lambda_2, \ldots, \lambda_n$. These are uniquely defined for a given matrix. The eigenvalues may or may not be equal to each other; for instance, the eigenvalues of the identity matrix I are all one, and every nonzero vector is an eigenvector of I.

What if B is applied to a vector that is not an eigenvector? A very important skill in understanding linear algebra — the skill this section is written to teach — is to think of a vector as a sum of other vectors whose behavior is understood. Consider that the set of eigenvectors $\{v_i\}$ forms a basis for $\mathbb{R}^n$ (because a symmetric B has n eigenvectors that are linearly independent). Any n-dimensional vector can be expressed as a linear combination of these eigenvectors, and because matrix-vector multiplication is distributive, one can examine the effect of B on each eigenvector separately.

In Figure 11, a vector x is illustrated as a sum of two eigenvectors v_1 and v_2. Applying B to x is equivalent to applying B to the eigenvectors, and summing the result. On repeated application, we have $B^i x = B^i v_1 + B^i v_2 = \lambda_1^i v_1 + \lambda_2^i v_2$. If the magnitudes of all the eigenvalues are smaller than one, $B^i x$ will converge to zero, because the eigenvectors that compose x converge to zero when B is repeatedly applied. If one of the eigenvalues has magnitude greater than one, x will diverge to infinity. This is why numerical analysts attach importance to the *spectral radius* of a matrix:

$$\rho(B) = \max |\lambda_i|, \qquad \lambda_i \text{ is an eigenvalue of } B.$$

If we want x to converge to zero quickly, $\rho(B)$ should be less than one, and preferably as small as possible.

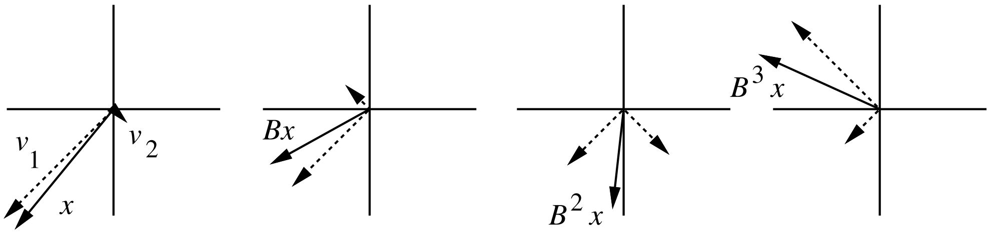


Figure 11: The vector x (solid arrow) can be expressed as a linear combination of eigenvectors (dashed arrows), whose associated eigenvalues are $\lambda_1 = 0.7$ and $\lambda_2 = -2$. The effect of repeatedly applying B to x is best understood by examining the effect of B on each eigenvector. When B is repeatedly applied, one eigenvector converges to zero while the other diverges; hence, $B^i x$ also diverges.

It's important to mention that there is a family of nonsymmetric matrices that do not have a full set of n independent eigenvectors. These matrices are known as *defective*, a name that betrays the well-deserved hostility they have earned from frustrated linear algebraists. The details are too complicated to describe in this article, but the behavior of defective matrices can be analyzed in terms of *generalized eigenvectors* and *generalized eigenvalues*. The rule that $B^i x$ converges to zero if and only if all the generalized eigenvalues have magnitude smaller than one still holds, but is harder to prove.

Here's a useful fact: the eigenvalues of a positive-definite matrix are all positive. This fact can be proven from the definition of eigenvalue:

$$\begin{aligned} Bv &= \lambda v \\ v^T B v &= \lambda v^T v. \end{aligned}$$

By the definition of positive-definite, the $v^T B v$ is positive (for nonzero v). Hence, λ must be positive also.

5.2. Jacobi iterations

Of course, a procedure that always converges to zero isn't going to help you attract friends. Consider a more useful procedure: the Jacobi Method for solving $Ax = b$. The matrix A is *split* into two parts: D, whose diagonal elements are identical to those of A, and whose off-diagonal elements are zero; and E, whose diagonal elements are zero, and whose off-diagonal elements are identical to those of A. Thus, $A = D + E$. We derive the Jacobi Method:

$$\begin{aligned} Ax &= b \\ Dx &= -Ex + b \\ x &= -D^{-1}Ex + D^{-1}b \\ x &= Bx + z, \qquad \text{where} \quad B = -D^{-1}E, \qquad z = D^{-1}b. \end{aligned} \tag{14}$$

Because D is diagonal, it is easy to invert. This identity can be converted into an iterative method by forming the recurrence

$$x_{(i+1)} = Bx_{(i)} + z. \tag{15}$$

Given a starting vector $x_{(0)}$, this formula generates a sequence of vectors. Our hope is that each successive vector will be closer to the solution x than the last. x is called a *stationary point* of Equation 15, because if $x_{(i)} = x$, then $x_{(i+1)}$ will also equal x.

Now, this derivation may seem quite arbitrary to you, and you're right. We could have formed any number of identities for x instead of Equation 14. In fact, simply by splitting A differently — that is, by choosing a different D and E — we could have derived the Gauss-Seidel method, or the method of Successive Over-Relaxation (SOR). Our hope is that we have chosen a splitting for which B has a small spectral radius. Here, I chose the Jacobi splitting arbitrarily for simplicity.

Suppose we start with some arbitrary vector $x_{(0)}$. For each iteration, we apply B to this vector, then add z to the result. What does each iteration do?

Again, apply the principle of thinking of a vector as a sum of other, well-understood vectors. Express each iterate $x_{(i)}$ as the sum of the exact solution x and the error term $e_{(i)}$. Then, Equation 15 becomes

$$\begin{aligned} x_{(i+1)} &= Bx_{(i)} + z \\ &= B(x + e_{(i)}) + z \\ &= Bx + z + Be_{(i)} \\ &= x + Be_{(i)} \qquad \text{(by Equation 14),} \\ \therefore e_{(i+1)} &= Be_{(i)}. \end{aligned} \tag{16}$$

Each iteration does not affect the "correct part" of $x_{(i)}$ (because x is a stationary point); but each iteration does affect the error term. It is apparent from Equation 16 that if $\rho(B) < 1$, then the error term $e_{(i)}$ will converge to zero as i approaches infinity. Hence, the initial vector $x_{(0)}$ has no effect on the inevitable outcome!

Of course, the choice of $x_{(0)}$ does affect the number of iterations required to converge to x within a given tolerance. However, its effect is less important than that of the spectral radius $\rho(B)$, which determines the speed of convergence. Suppose that v_j is the eigenvector of B with the largest eigenvalue (so that $\rho(B) = \lambda_j$). If the initial error $e_{(0)}$, expressed as a linear combination of eigenvectors, includes a component in the direction of v_j, this component will be the slowest to converge.

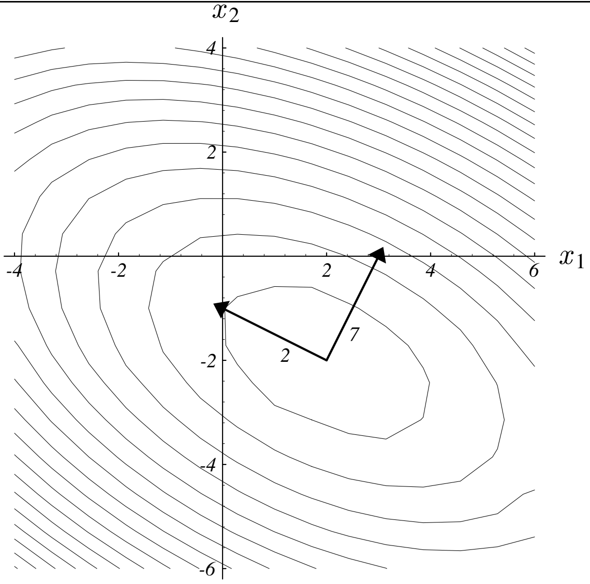


Figure 12: The eigenvectors of A are directed along the axes of the paraboloid defined by the quadratic form $f(x)$. Each eigenvector is labeled with its associated eigenvalue. Each eigenvalue is proportional to the steepness of the corresponding slope.

B is not generally symmetric (even if A is), and may even be defective. However, the rate of convergence of the Jacobi Method depends largely on $\rho(B)$, which depends on A. Unfortunately, Jacobi does not converge for every A, or even for every positive-definite A.

5.3. A Concrete Example

To demonstrate these ideas, I shall solve the system specified by Equation 4. First, we need a method of finding eigenvalues and eigenvectors. By definition, for any eigenvector v with eigenvalue λ,

$$Av = \lambda v = \lambda I v$$

$$(\lambda I - A)v = 0.$$

Eigenvectors are nonzero, so $\lambda I - A$ must be singular. Then,

$$\det(\lambda I - A) = 0.$$

The determinant of $\lambda I - A$ is called the *characteristic polynomial.* It is a degree n polynomial in λ whose roots are the set of eigenvalues. The characteristic polynomial of A (from Equation 4) is

$$\det \begin{bmatrix} \lambda - 3 & -2 \\ -2 & \lambda - 6 \end{bmatrix} = \lambda^2 - 9\lambda + 14 = (\lambda - 7)(\lambda - 2),$$

and the eigenvalues are 7 and 2. To find the eigenvector associated with $\lambda = 7$,

$$\begin{aligned} (\lambda I - A)v = \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} &= 0 \\ \therefore 4v_1 - 2v_2 &= 0. \end{aligned}$$

Any solution to this equation is an eigenvector; say, $v = [1, 2]^T$. By the same method, we find that $[-2, 1]^T$ is an eigenvector corresponding to the eigenvalue 2. In Figure 12, we see that these eigenvectors coincide with the axes of the familiar ellipsoid, and that a larger eigenvalue corresponds to a steeper slope. (Negative eigenvalues indicate that f decreases along the axis, as in Figures 5(b) and 5(d).)

Now, let's see the Jacobi Method in action. Using the constants specified by Equation 4, we have

$$\begin{aligned} x_{(i+1)} &= -\begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix} x_{(i)} + \begin{bmatrix} \frac{1}{3} & 0 \\ 0 & \frac{1}{6} \end{bmatrix} \begin{bmatrix} 2 \\ -8 \end{bmatrix} \\ &= \begin{bmatrix} 0 & -\frac{2}{3} \\ -\frac{1}{3} & 0 \end{bmatrix} x_{(i)} + \begin{bmatrix} \frac{2}{3} \\ -\frac{4}{3} \end{bmatrix} \end{aligned}$$

The eigenvectors of B are $[\sqrt{2}, 1]^T$ with eigenvalue $-\sqrt{2}/3$, and $[-\sqrt{2}, 1]^T$ with eigenvalue $\sqrt{2}/3$. These are graphed in Figure 13(a); note that they do not coincide with the eigenvectors of A, and are not related to the axes of the paraboloid.

Figure 13(b) shows the convergence of the Jacobi method. The mysterious path the algorithm takes can be understood by watching the eigenvector components of each successive error term (Figures 13(c), (d), and (e)). Figure 13(f) plots the eigenvector components as arrowheads. These are converging normally at the rate defined by their eigenvalues, as in Figure 11.

I hope that this section has convinced you that eigenvectors are useful tools, and not just bizarre torture devices inflicted on you by your professors for the pleasure of watching you suffer (although the latter is a nice fringe benefit).

6. Convergence Analysis of Steepest Descent

6.1. Instant Results

To understand the convergence of Steepest Descent, let's first consider the case where $e_{(i)}$ is an eigenvector with eigenvalue λ_e. Then, the residual $r_{(i)} = -Ae_{(i)} = -\lambda_e e_{(i)}$ is also an eigenvector. Equation 12 gives

$$\begin{aligned} e_{(i+1)} &= e_{(i)} + \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}} r_{(i)} \\ &= e_{(i)} + \frac{r_{(i)}^T r_{(i)}}{\lambda_e r_{(i)}^T r_{(i)}} (-\lambda_e e_{(i)}) \\ &= 0. \end{aligned}$$

Figure 14 demonstrates why it takes only one step to converge to the exact solution. The point $x_{(i)}$ lies on one of the axes of the ellipsoid, and so the residual points directly to the center of the ellipsoid. Choosing $\alpha_{(i)} = \lambda_e^{-1}$ gives us instant convergence.

For a more general analysis, we must express $e_{(i)}$ as a linear combination of eigenvectors, and we shall furthermore require these eigenvectors to be orthonormal. It is proven in Appendix C2 that if A is

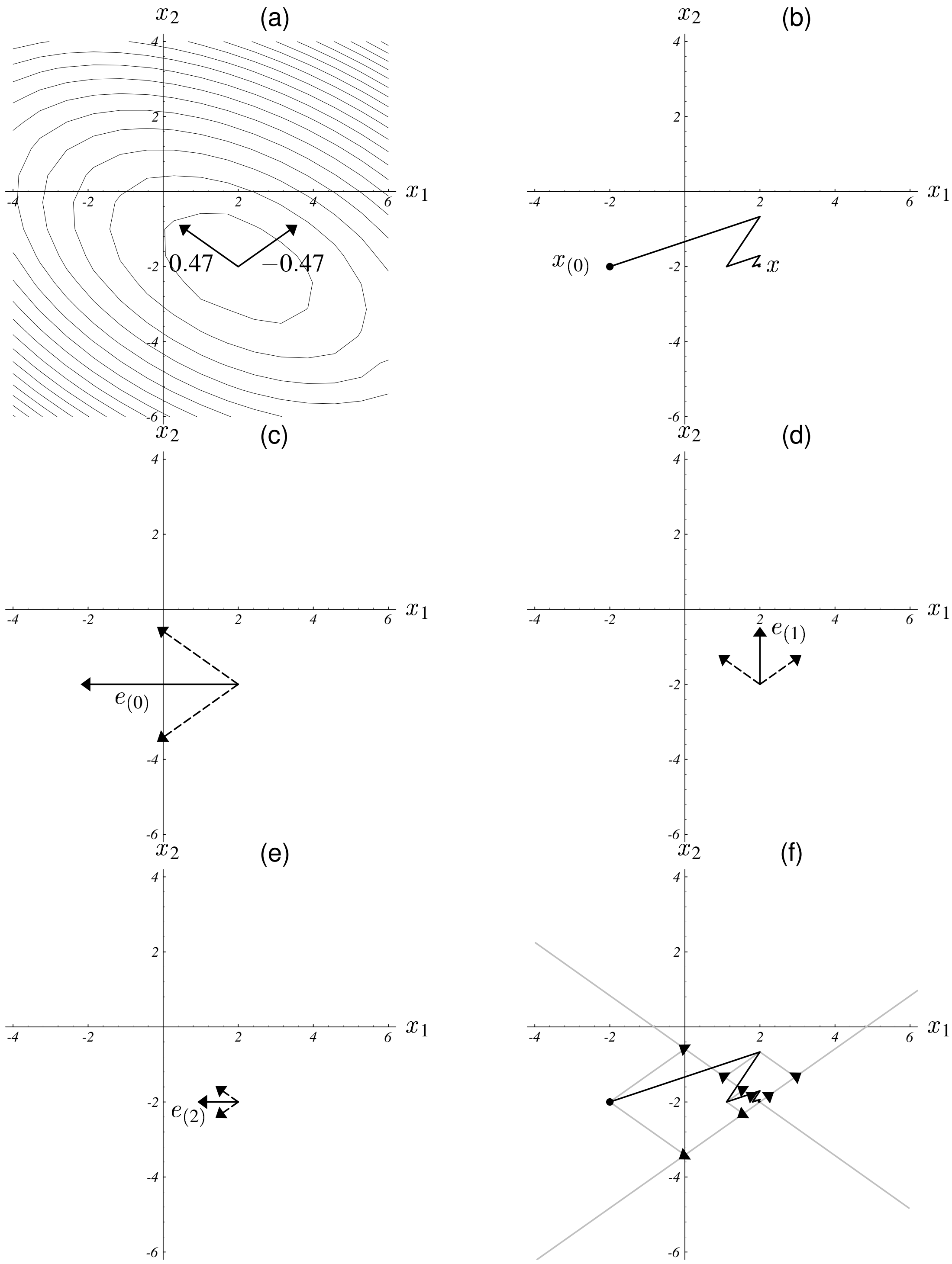


Figure 13: Convergence of the Jacobi Method. (a) The eigenvectors of B are shown with their corresponding eigenvalues. Unlike the eigenvectors of A, these eigenvectors are not the axes of the paraboloid. (b) The Jacobi Method starts at $[-2, -2]^T$ and converges at $[2, -2]^T$. (c, d, e) The error vectors $e_{(0)}$, $e_{(1)}$, $e_{(2)}$ (solid arrows) and their eigenvector components (dashed arrows). (f) Arrowheads represent the eigenvector components of the first four error vectors. Each eigenvector component of the error is converging to zero at the expected rate based on its eigenvalue.

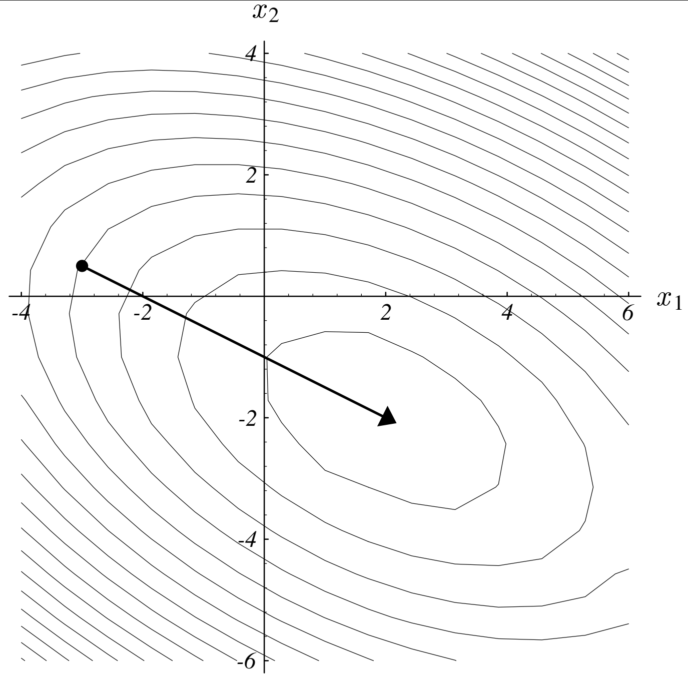


Figure 14: Steepest Descent converges to the exact solution on the first iteration if the error term is an eigenvector.

symmetric, there exists a set of n orthogonal eigenvectors of A. As we can scale eigenvectors arbitrarily, let us choose so that each eigenvector is of unit length. This choice gives us the useful property that

$$v_j^T v_k = \begin{cases} 1, & j = k, \\ 0, & j \neq k. \end{cases} \tag{17}$$

Express the error term as a linear combination of eigenvectors

$$e_{(i)} = \sum_{j=1}^{n} \xi_j v_j, \tag{18}$$

where ξ_j is the length of each component of $e_{(i)}$. From Equations 17 and 18 we have the following identities:

$$\begin{aligned}
r_{(i)} &= -Ae_{(i)} = -\sum_j \xi_j \lambda_j v_j, && (19) \\
\|e_{(i)}\|^2 = e_{(i)}^T e_{(i)} &= \sum_j \xi_j^2, && (20) \\
e_{(i)}^T A e_{(i)} &= \left(\sum_j \xi_j v_j^T\right)\left(\sum_j \xi_j \lambda_j v_j\right) \\
&= \sum_j \xi_j^2 \lambda_j, && (21) \\
\|r_{(i)}\|^2 = r_{(i)}^T r_{(i)} &= \sum_j \xi_j^2 \lambda_j^2, && (22) \\
r_{(i)}^T A r_{(i)} &= \sum_j \xi_j^2 \lambda_j^3. && (23)
\end{aligned}$$

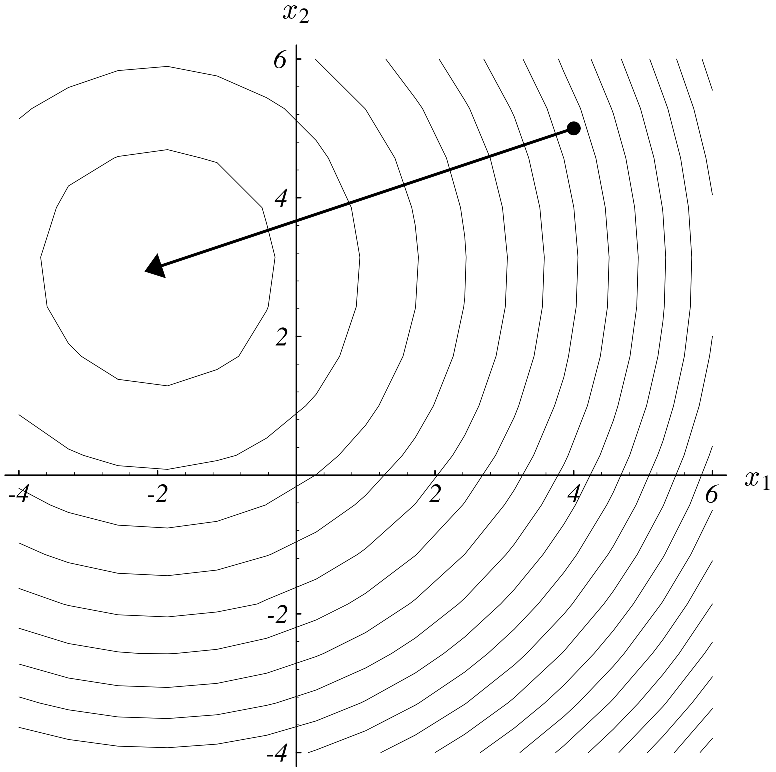


Figure 15: Steepest Descent converges to the exact solution on the first iteration if the eigenvalues are all equal.

Equation 19 shows that $r_{(i)}$ too can be expressed as a sum of eigenvector components, and the length of these components are $-\xi_j\lambda_j$. Equations 20 and 22 are just Pythagoras' Law.

Now we can proceed with the analysis. Equation 12 gives

$$\begin{aligned} e_{(i+1)} &= e_{(i)} + \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}} r_{(i)} \\ &= e_{(i)} + \frac{\sum_j \xi_j^2 \lambda_j^2}{\sum_j \xi_j^2 \lambda_j^3} r_{(i)} \end{aligned} \tag{24}$$

We saw in the last example that, if $e_{(i)}$ has only one eigenvector component, then convergence is achieved in one step by choosing $\alpha_{(i)} = \lambda_e^{-1}$. Now let's examine the case where $e_{(i)}$ is arbitrary, but all the eigenvectors have a common eigenvalue λ. Equation 24 becomes

$$\begin{aligned} e_{(i+1)} &= e_{(i)} + \frac{\lambda^2 \sum_j \xi_j^2}{\lambda^3 \sum_j \xi_j^2}(-\lambda e_{(i)}) \\ &= 0 \end{aligned}$$

Figure 15 demonstrates why, once again, there is instant convergence. Because all the eigenvalues are equal, the ellipsoid is spherical; hence, no matter what point we start at, the residual must point to the center of the sphere. As before, choose $\alpha_{(i)} = \lambda^{-1}$.

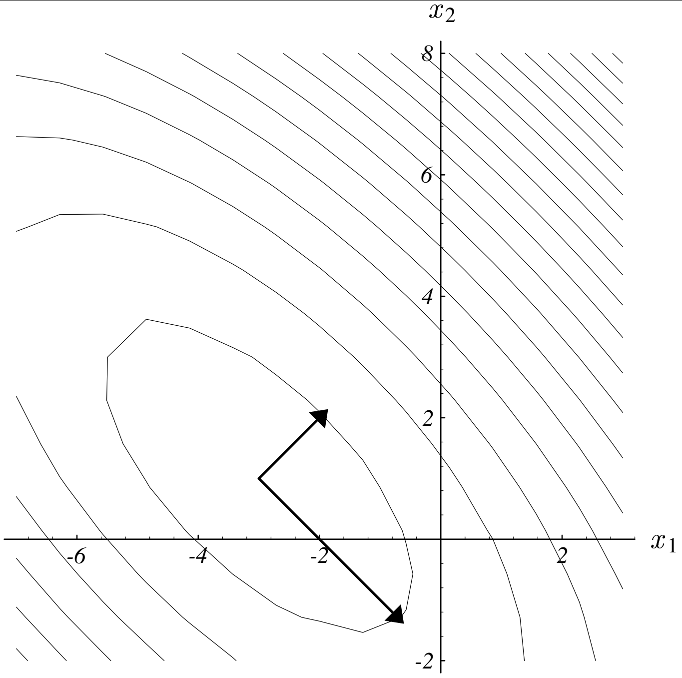


Figure 16: The energy norm of these two vectors is equal.

However, if there are several unequal, nonzero eigenvalues, then no choice of $\alpha_{(i)}$ will eliminate all the eigenvector components, and our choice becomes a sort of compromise. In fact, the fraction in Equation 24 is best thought of as a weighted average of the values of λ_j^{-1}. The weights ξ_j^2 ensure that longer components of $e_{(i)}$ are given precedence. As a result, on any given iteration, some of the shorter components of $e_{(i)}$ might actually *increase* in length (though never for long). For this reason, the methods of Steepest Descent and Conjugate Gradients are called *roughers*. By contrast, the Jacobi Method is a *smoother*, because every eigenvector component is reduced on every iteration. Steepest Descent and Conjugate Gradients are not smoothers, although they are often erroneously identified as such in the mathematical literature.

6.2. General Convergence

To bound the convergence of Steepest Descent in the general case, we shall define the *energy norm* $\|e\|_A = (e^T A e)^{1/2}$ (see Figure 16). This norm is easier to work with than the Euclidean norm, and is in some sense a more natural norm; examination of Equation 8 shows that minimizing $\|e_{(i)}\|_A$ is equivalent to minimizing $f(x_{(i)})$. With this norm, we have

$$\begin{aligned}
\|e_{(i+1)}\|_A^2 &= e_{(i+1)}^T A e_{(i+1)} \\
&= (e_{(i)}^T + \alpha_{(i)} r_{(i)}^T) A (e_{(i)} + \alpha_{(i)} r_{(i)}) \qquad \text{(by Equation 12)} \\
&= e_{(i)}^T A e_{(i)} + 2\alpha_{(i)} r_{(i)}^T A e_{(i)} + \alpha_{(i)}^2 r_{(i)}^T A r_{(i)} \qquad \text{(by symmetry of } A\text{)} \\
&= \|e_{(i)}\|_A^2 + 2\frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}} \left(-r_{(i)}^T r_{(i)}\right) + \left(\frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}\right)^2 r_{(i)}^T A r_{(i)} \\
&= \|e_{(i)}\|_A^2 - \frac{(r_{(i)}^T r_{(i)})^2}{r_{(i)}^T A r_{(i)}} \\
&= \|e_{(i)}\|_A^2 \left(1 - \frac{(r_{(i)}^T r_{(i)})^2}{(r_{(i)}^T A r_{(i)})(e_{(i)}^T A e_{(i)})}\right) \\
&= \|e_{(i)}\|_A^2 \left(1 - \frac{(\sum_j \xi_j^2 \lambda_j^2)^2}{(\sum_j \xi_j^2 \lambda_j^3)(\sum_j \xi_j^2 \lambda_j)}\right) \qquad \text{(by Identities 21, 22, 23)} \\
&= \|e_{(i)}\|_A^2 \omega^2, \qquad \omega^2 = 1 - \frac{(\sum_j \xi_j^2 \lambda_j^2)^2}{(\sum_j \xi_j^2 \lambda_j^3)(\sum_j \xi_j^2 \lambda_j)} \qquad (25)
\end{aligned}$$

The analysis depends on finding an upper bound for ω. To demonstrate how the weights and eigenvalues affect convergence, I shall derive a result for $n = 2$. Assume that $\lambda_1 \geq \lambda_2$. The *spectral condition number* of A is defined to be $\kappa = \lambda_1/\lambda_2 \geq 1$. The *slope* of $e_{(i)}$ (relative to the coordinate system defined by the eigenvectors), which depends on the starting point, is denoted $\mu = \xi_2/\xi_1$. We have

$$\begin{aligned}
\omega^2 &= 1 - \frac{(\xi_1^2 \lambda_1^2 + \xi_2^2 \lambda_2^2)^2}{(\xi_1^2 \lambda_1 + \xi_2^2 \lambda_2)(\xi_1^2 \lambda_1^3 + \xi_2^2 \lambda_2^3)} \\
&= 1 - \frac{(\kappa^2 + \mu^2)^2}{(\kappa + \mu^2)(\kappa^3 + \mu^2)} \qquad (26)
\end{aligned}$$

The value of ω, which determines the rate of convergence of Steepest Descent, is graphed as a function of μ and κ in Figure 17. The graph confirms my two examples. If $e_{(0)}$ is an eigenvector, then the slope μ is zero (or infinite); we see from the graph that ω is zero, so convergence is instant. If the eigenvalues are equal, then the condition number κ is one; again, we see that ω is zero.

Figure 18 illustrates examples from near each of the four corners of Figure 17. These quadratic forms are graphed in the coordinate system defined by their eigenvectors. Figures 18(a) and 18(b) are examples with a large condition number. Steepest Descent can converge quickly if a fortunate starting point is chosen (Figure 18(a)), but is usually at its worst when κ is large (Figure 18(b)). The latter figure gives us our best intuition for why a large condition number can be bad: $f(x)$ forms a trough, and Steepest Descent bounces back and forth between the sides of the trough while making little progress along its length. In Figures 18(c) and 18(d), the condition number is small, so the quadratic form is nearly spherical, and convergence is quick regardless of the starting point.

Holding κ constant (because A is fixed), a little basic calculus reveals that Equation 26 is maximized when $\mu = \pm\kappa$. In Figure 17, one can see a faint ridge defined by this line. Figure 19 plots worst-case starting points for our sample matrix A. These starting points fall on the lines defined by $\xi_2/\xi_1 = \pm\kappa$. An

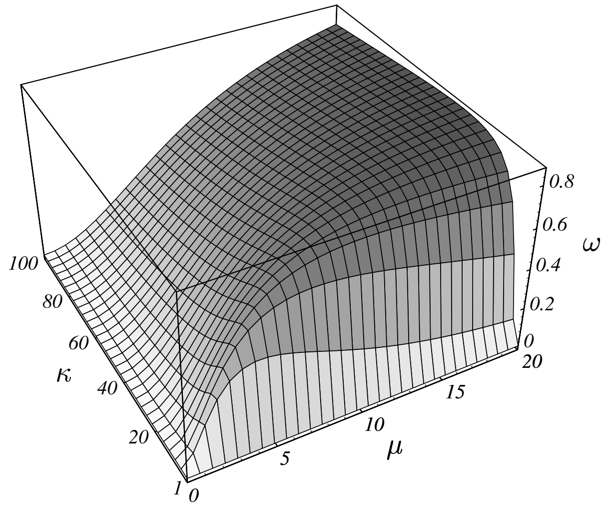


Figure 17: Convergence ω of Steepest Descent as a function of μ (the slope of $e_{(i)}$) and κ (the condition number of A). Convergence is fast when μ or κ are small. For a fixed matrix, convergence is worst when $\mu = \pm\kappa$.

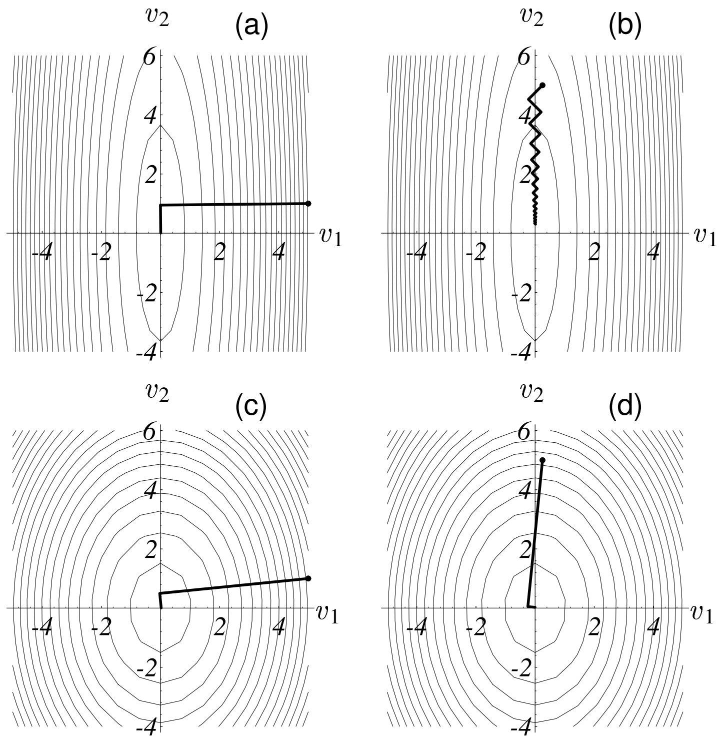


Figure 18: These four examples represent points near the corresponding four corners of the graph in Figure 17. (a) Large κ, small μ. (b) An example of poor convergence. κ and μ are both large. (c) Small κ and μ. (d) Small κ, large μ.

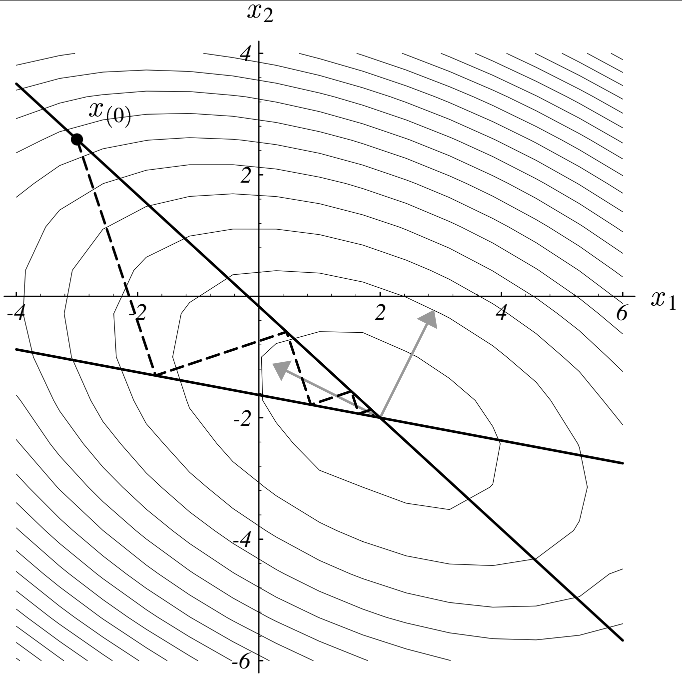


Figure 19: Solid lines represent the starting points that give the worst convergence for Steepest Descent. Dashed lines represent steps toward convergence. If the first iteration starts from a worst-case point, so do all succeeding iterations. Each step taken intersects the paraboloid axes (gray arrows) at precisely a $45°$ angle. Here, $\kappa = 3.5$.

upper bound for ω (corresponding to the worst-case starting points) is found by setting $\mu^2 = \kappa^2$:

$$\begin{aligned}
\omega^2 &\leq 1 - \frac{4\kappa^4}{\kappa^5 + 2\kappa^4 + \kappa^3} \\
&= \frac{\kappa^5 - 2\kappa^4 + \kappa^3}{\kappa^5 + 2\kappa^4 + \kappa^3} \\
&= \frac{(\kappa - 1)^2}{(\kappa + 1)^2} \\
\omega &\leq \frac{\kappa - 1}{\kappa + 1}. \qquad (27)
\end{aligned}$$

Inequality 27 is plotted in Figure 20. The more *ill-conditioned* the matrix (that is, the larger its condition number κ), the slower the convergence of Steepest Descent. In Section 9.2, it is proven that Equation 27 is also valid for $n > 2$, if the condition number of a symmetric, positive-definite matrix is defined to be

$$\kappa = \lambda_{max}/\lambda_{min},$$

the ratio of the largest to smallest eigenvalue. The convergence results for Steepest Descent are

$$\|e_{(i)}\|_A \leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^i \|e_{(0)}\|_A, \text{ and} \qquad (28)$$

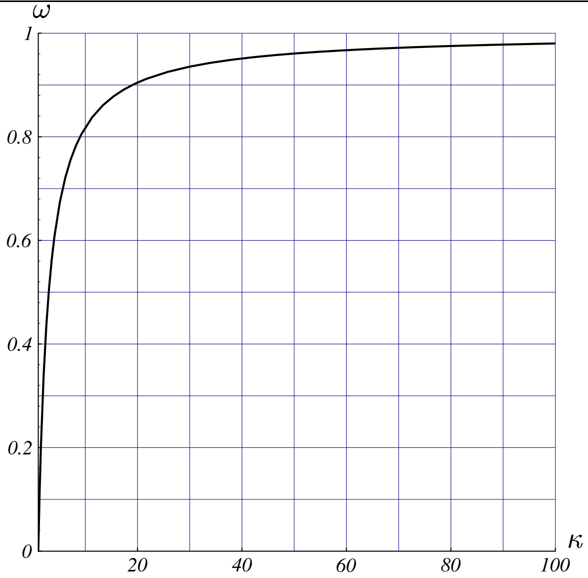


Figure 20: Convergence of Steepest Descent (per iteration) worsens as the condition number of the matrix increases.

$$\begin{aligned}\frac{f(x_{(i)}) - f(x)}{f(x_{(0)}) - f(x)} &= \frac{\frac{1}{2}e_{(i)}^T A e_{(i)}}{\frac{1}{2}e_{(0)}^T A e_{(0)}} \qquad \text{(by Equation 8)} \\ &\leq \left(\frac{\kappa - 1}{\kappa + 1}\right)^{2i}.\end{aligned}$$

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* $d_{(0)}, d_{(1)}, \ldots, d_{(n-1)}$. 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

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

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

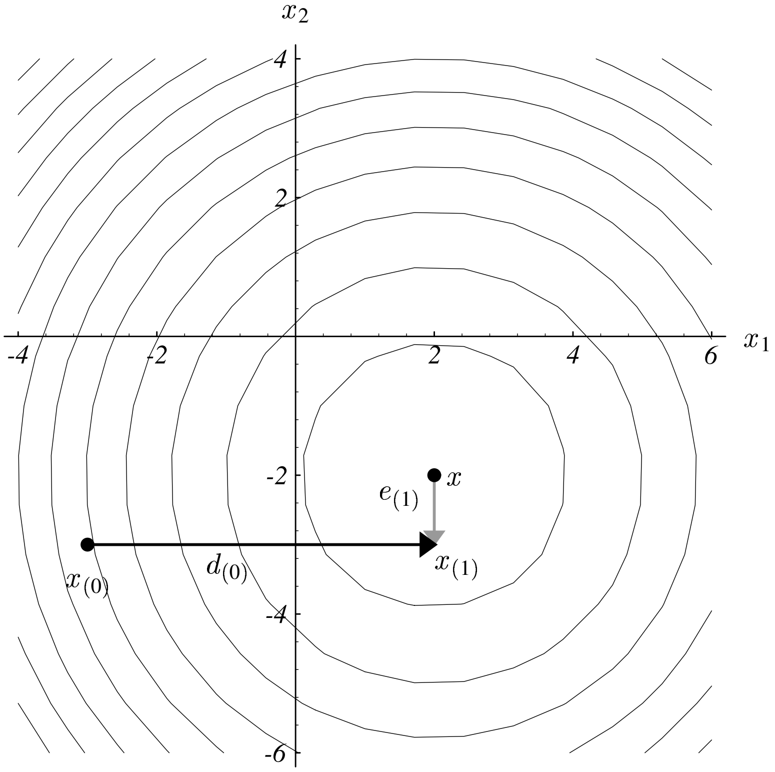


Figure 21: The Method of Orthogonal Directions. Unfortunately, this method only works if you already know the answer.

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

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

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

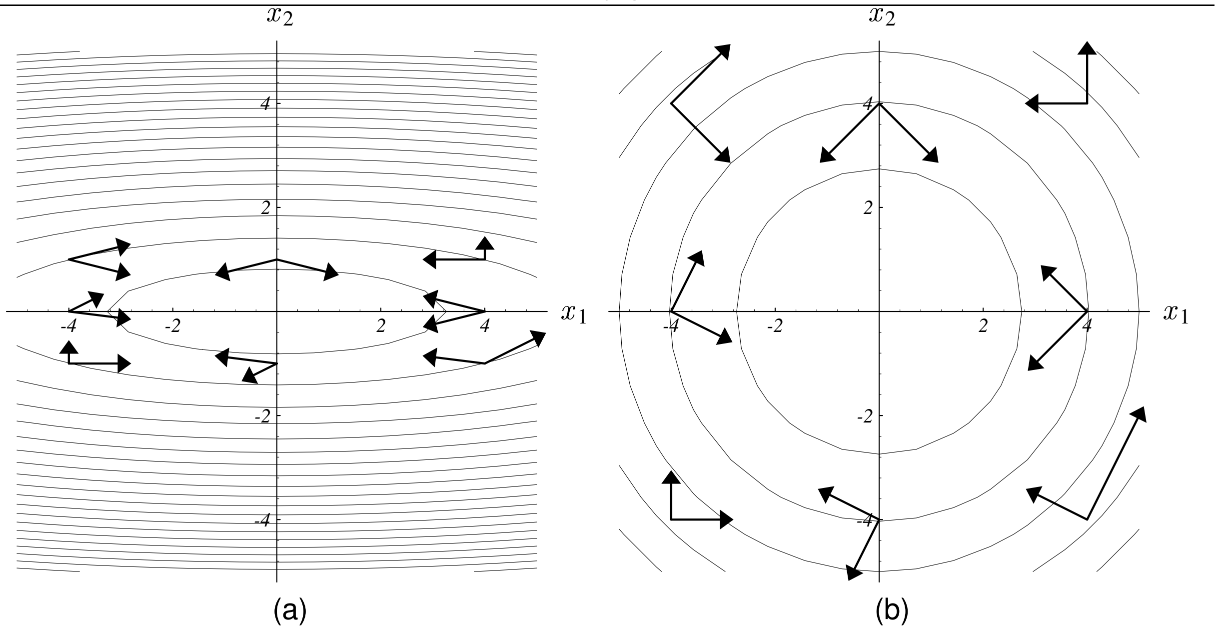


Figure 22: These pairs of vectors are A-orthogonal . . . because these pairs of vectors are orthogonal.

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.

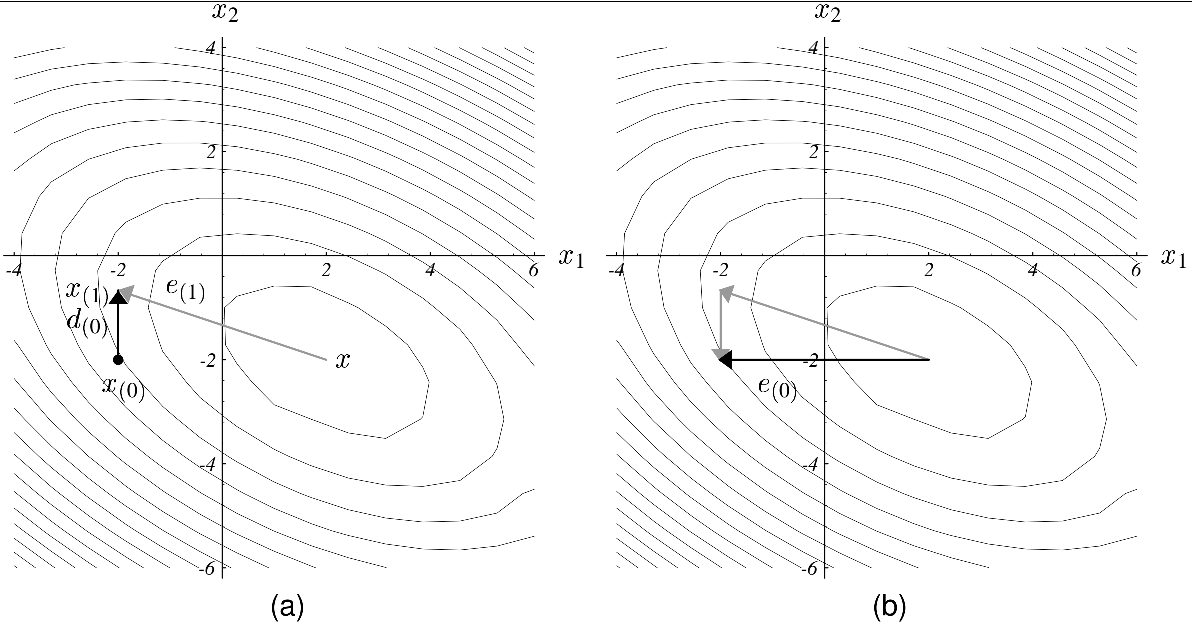


Figure 23: The method of Conjugate Directions converges in n steps. (a) The first step is taken along some direction $d_{(0)}$. The minimum point $x_{(1)}$ is chosen by the constraint that $e_{(1)}$ must be A-orthogonal to $d_{(0)}$. (b) The initial error $e_{(0)}$ can be expressed as a sum of A-orthogonal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

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

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 $d_{(k)}^T A$:

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

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)).

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

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 $u_0, u_1, \ldots, u_{n-1}$. 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

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

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

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

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

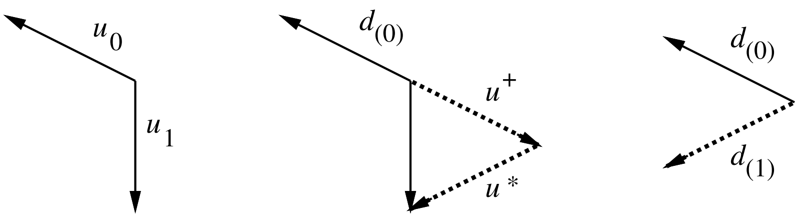


Figure 24: Gram-Schmidt conjugation of two vectors. Begin with two linearly independent vectors u_0 and u_1. Set $d_{(0)} = u_0$. The vector u_1 is composed of two components: u^*, which is A-orthogonal (or conjugate) to $d_{(0)}$, and u^+, which is parallel to $d_{(0)}$. After conjugation, only the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

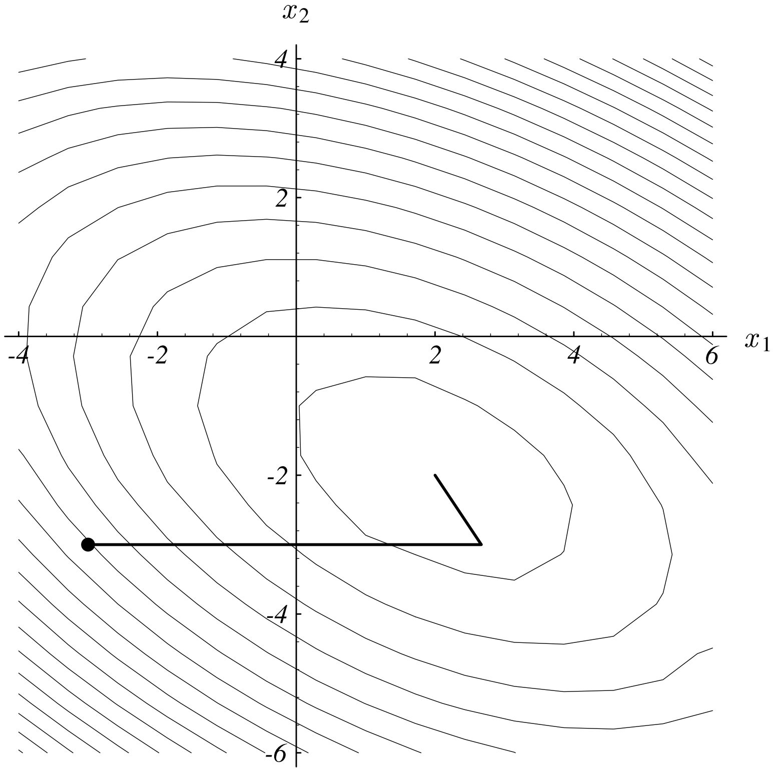


Figure 25: The method of Conjugate Directions using the axial unit vectors, also known as Gaussian elimination.

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 span$\{d_{(0)}, d_{(1)}, \ldots, d_{(i-1)}\}$; 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

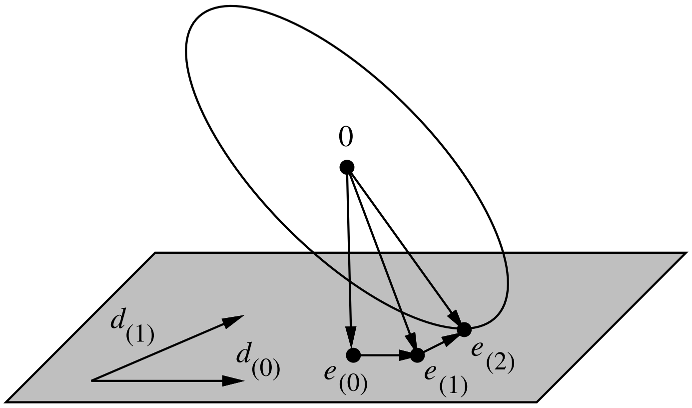


Figure 26: In this figure, the shaded area is $e_{(0)} + \mathcal{D}_2 = e_{(0)} + \text{span}\{d_{(0)}, d_{(1)}\}$. The ellipsoid is a contour on which the energy norm is constant. After two steps, Conjugate Directions finds $e_{(2)}$, the point on $e_{(0)} + \mathcal{D}_2$ that minimizes $\|e\|_A$.

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

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

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 $x_{(0)} + \mathcal{D}_i$?

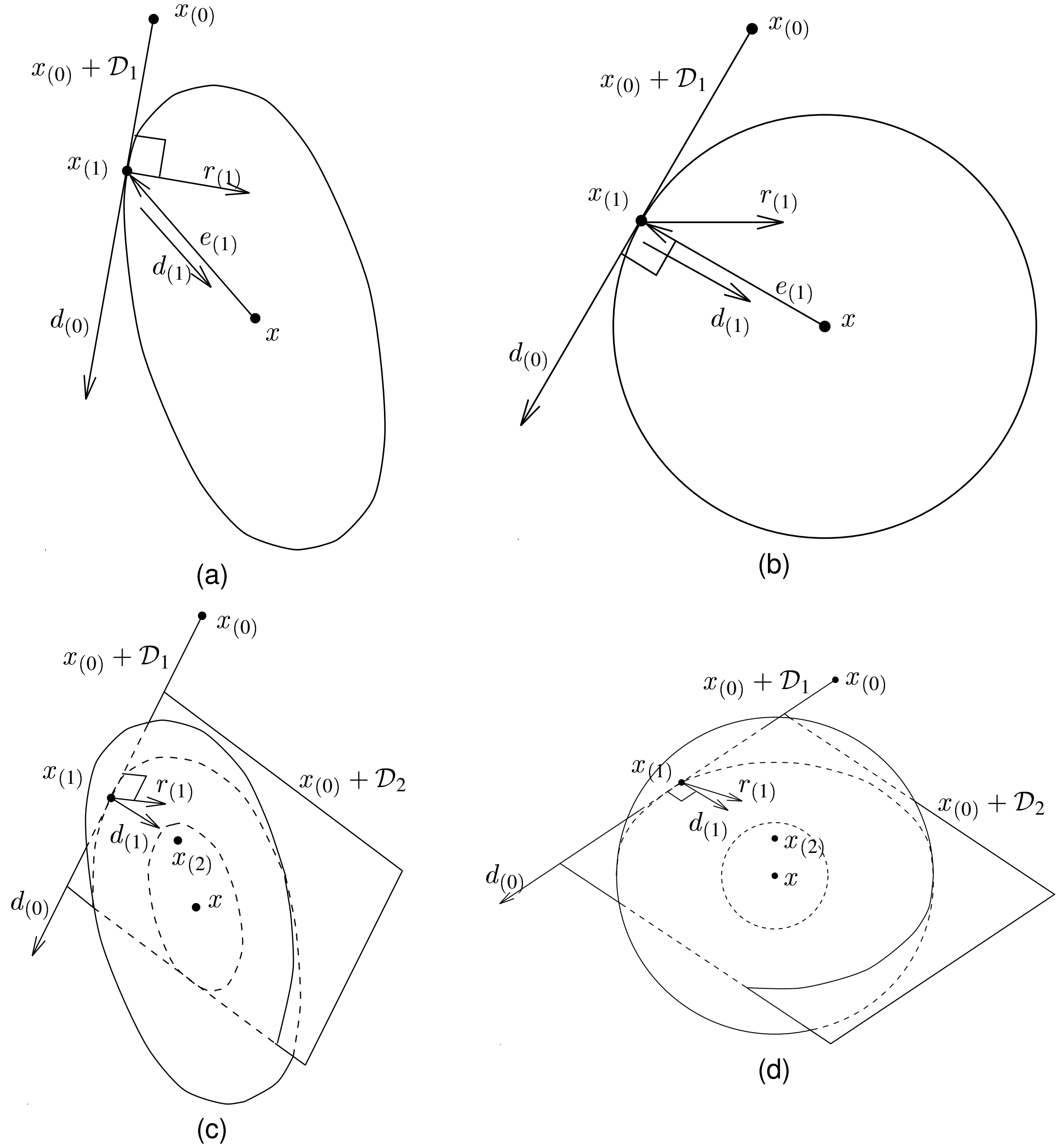


Figure 27: Optimality of the Method of Conjugate Directions. (a) A two-dimensional problem. Lines that appear perpendicular are orthogonal. (b) The same problem in a "stretched" space. Lines that appear perpendicular are A-orthogonal. (c) A three-dimensional problem. Two concentric ellipsoids are shown; 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 $-d_{(i)}^T A$:

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

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

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 $u_0, \ldots, u_{i-1}$ 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 $r_{(j)}$:

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

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

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

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

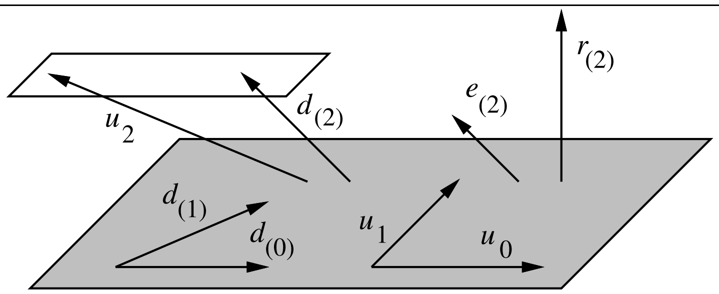


Figure 28: Because the search directions $d_{(0)}, d_{(1)}$ are constructed from the vectors u_0, u_1, they span the same subspace $\mathcal{D}_2$ (the gray-colored plane). The error term $e_{(2)}$ is A-orthogonal to $\mathcal{D}_2$, the residual $r_{(2)}$ is orthogonal to $\mathcal{D}_2$, and a new 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:

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

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 span$\{r_{(0)}, r_{(1)}, \ldots, r_{(i-1)}\}$ 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

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

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,

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

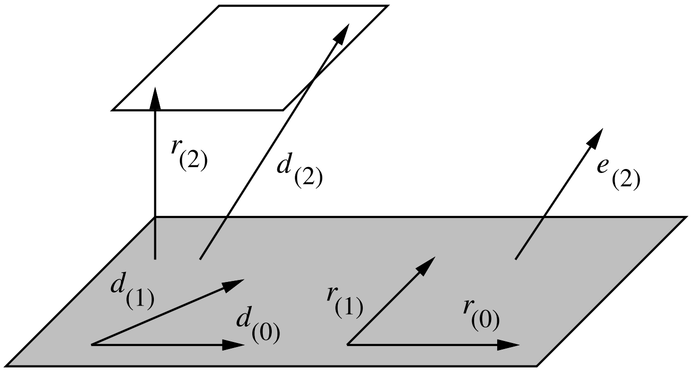


Figure 29: In the method of Conjugate Gradients, each new residual is orthogonal to all the previous residuals and search directions; and each new search direction is constructed (from the residual) to be A-orthogonal to all the previous residuals and search directions. The endpoints of $r_{(2)}$ and $d_{(2)}$ 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 $A\mathcal{D}_i$ 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 $\beta_{ij} = -r_{(i)}^T A d_{(j)} / d_{(j)}^T A d_{(j)}$; let us simplify this expression. Taking the inner product of $r_{(i)}$ and Equation 43,

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

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:

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

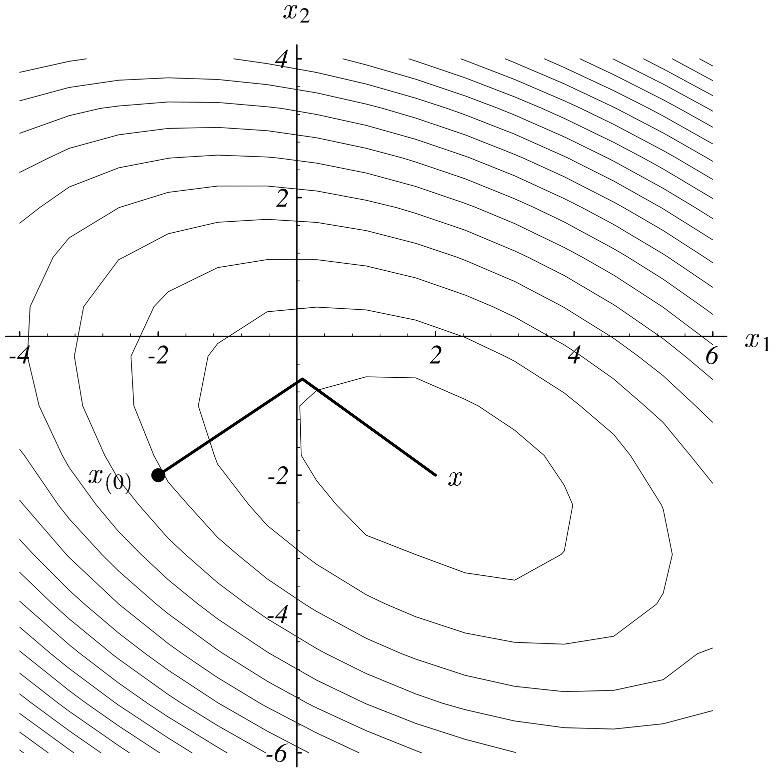


Figure 30: The method of Conjugate Gradients.

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

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

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

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

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

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

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

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

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

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 $Av = \lambda v$, $A^2 v = \lambda^2 v$, 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

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

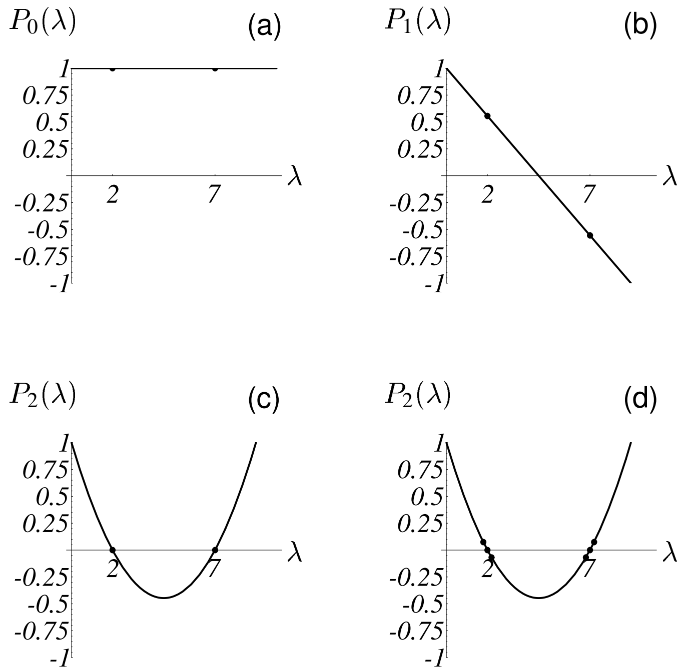


Figure 31: The convergence of CG after i iterations depends on how close a polynomial P_i of degree i can be to zero on each eigenvalue, given the constraint that $P_i(0) = 1$.

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

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

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 $P_1(\lambda) = 1 - 2x/9$, 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.)

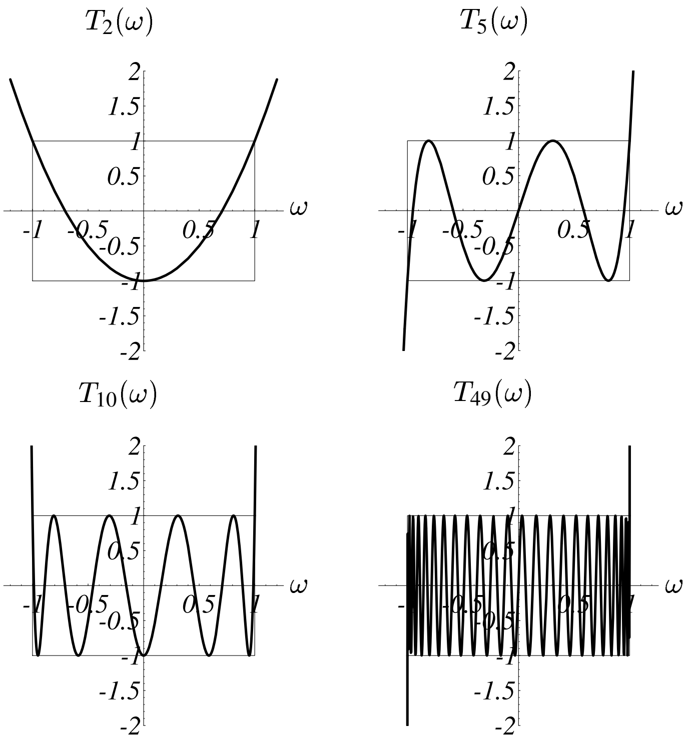


Figure 32: Chebyshev polynomials of degree 2, 5, 10, and 49.

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.

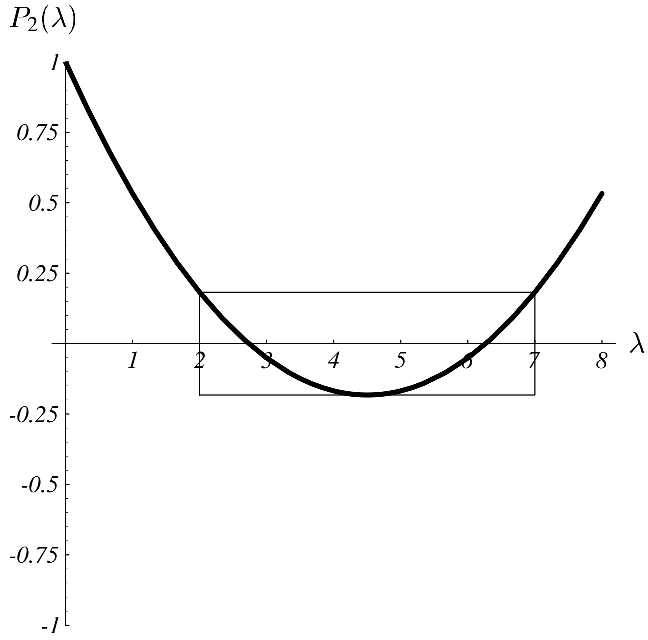


Figure 33: The polynomial $P_2(\lambda)$ that minimizes Equation 50 for $\lambda_{min} = 2$ and $\lambda_{max} = 7$ in the general case. This curve is a scaled version of the Chebyshev polynomial of degree 2. The energy norm of the error term after two iterations is no greater than 0.183 times its initial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

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

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

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

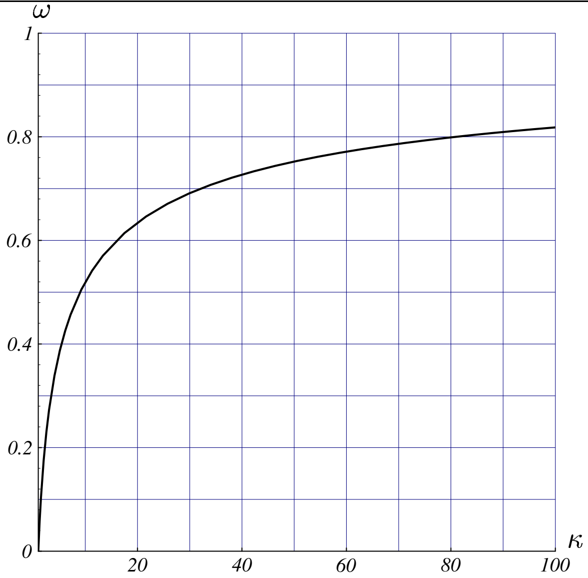


Figure 34: Convergence of Conjugate Gradients (per iteration) as a function of condition number. Compare with Figure 20.

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

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

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

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

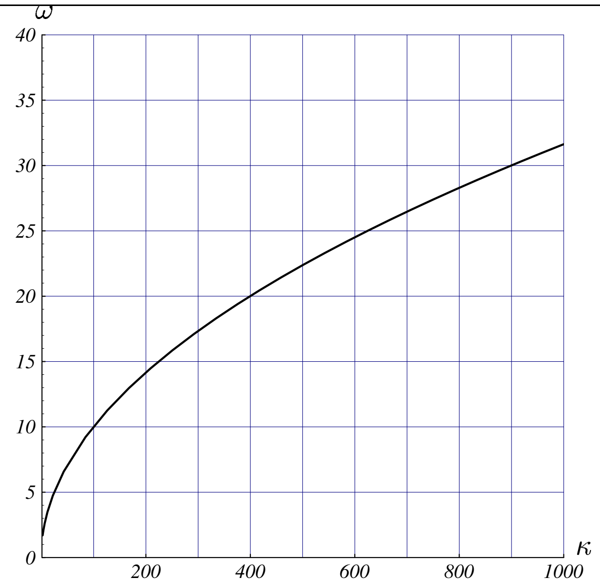


Figure 35: Number of iterations of Steepest Descent required to match one iteration of CG.

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

$$M^{-1}Ax = M^{-1}b. \tag{53}$$

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

$$E^{-1}AE^{-T}\widehat{x} = E^{-1}b, \qquad \widehat{x} = E^T x,$$

which we solve first for $\widehat{x}$, then for x. Because $E^{-1}AE^{-T}$ is symmetric and positive-definite, $\widehat{x}$ 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*:

$$\widehat{d}_{(0)} = \widehat{r}_{(0)} = E^{-1}b - E^{-1}AE^{-T}\widehat{x}_{(0)},$$

$$\alpha_{(i)} = \frac{\widehat{r}_{(i)}^T \widehat{r}_{(i)}}{\widehat{d}_{(i)}^T E^{-1}AE^{-T}\widehat{d}_{(i)}},$$

$$\widehat{x}_{(i+1)} = \widehat{x}_{(i)} + \alpha_{(i)}\widehat{d}_{(i)},$$

$$\widehat{r}_{(i+1)} = \widehat{r}_{(i)} - \alpha_{(i)}E^{-1}AE^{-T}\widehat{d}_{(i)},$$

$$\beta_{(i+1)} = \frac{\widehat{r}_{(i+1)}^T \widehat{r}_{(i+1)}}{\widehat{r}_{(i)}^T \widehat{r}_{(i)}},$$

$$\widehat{d}_{(i+1)} = \widehat{r}_{(i+1)} + \beta_{(i+1)}\widehat{d}_{(i)}.$$

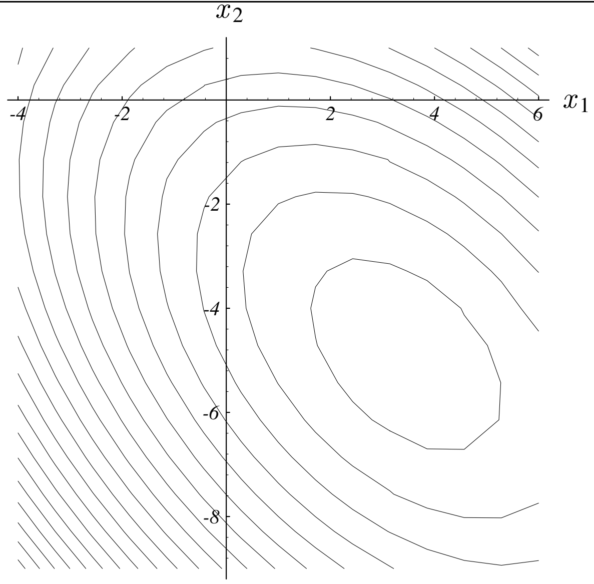


Figure 36: Contour lines of the quadratic form of the diagonally preconditioned sample problem.

This method has the undesirable characteristic that E must be computed. However, a few careful variable substitutions can eliminate E. Setting $\widehat{r}_{(i)} = E^{-1}r_{(i)}$ and $\widehat{d}_{(i)} = E^T d_{(i)}$, and using the identities $\widehat{x}_{(i)} = E^T x_{(i)}$ 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 $\widehat{L}\widehat{L}^T$, where $\widehat{L}$ might be restricted to have the same pattern of nonzero elements as A; other elements of L are thrown away. To use $\widehat{L}\widehat{L}^T$ as a preconditioner, the solution to $\widehat{L}\widehat{L}^T w = z$ is computed by backsubstitution (the inverse of $\widehat{L}\widehat{L}^T$ 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

$$\min_x \|Ax - b\|^2 \tag{54}$$

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

$$A^T A x = A^T b. \tag{55}$$

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, $x^T A^T A x = \|Ax\|^2 \geq 0$). 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 Ad 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:

$$\beta^{FR}_{(i+1)} = \frac{r^T_{(i+1)} r_{(i+1)}}{r^T_{(i)} r_{(i)}}, \qquad \beta^{PR}_{(i+1)} = \frac{r^T_{(i+1)} (r_{(i+1)} - r_{(i)})}{r^T_{(i)} r_{(i)}}.$$

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:

$$d_{(0)} = r_{(0)} = -f'(x_{(0)}),$$

$$\text{Find } \alpha_{(i)} \text{ that minimizes } f(x_{(i)} + \alpha_{(i)} d_{(i)}),$$

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

$$r_{(i+1)} = -f'(x_{(i+1)}),$$

$$\beta_{(i+1)} = \frac{r^T_{(i+1)} r_{(i+1)}}{r^T_{(i)} r_{(i)}} \qquad \text{or} \qquad \beta_{(i+1)} = \max\left\{ \frac{r^T_{(i+1)} (r_{(i+1)} - r_{(i)})}{r^T_{(i)} r_{(i)}}, 0 \right\},$$

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

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

$$\begin{aligned} f(x+\alpha d) &\approx f(x) + \alpha \left[\frac{d}{d\alpha} f(x+\alpha d)\right]_{\alpha=0} + \frac{\alpha^2}{2}\left[\frac{d^2}{d\alpha^2} f(x+\alpha d)\right]_{\alpha=0} \qquad (56) \\ &= f(x) + \alpha [f'(x)]^T d + \frac{\alpha^2}{2} d^T f''(x) d \\ \frac{d}{d\alpha} f(x+\alpha d) &\approx [f'(x)]^T d + \alpha d^T f''(x) d. \qquad (57) \end{aligned}$$

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

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

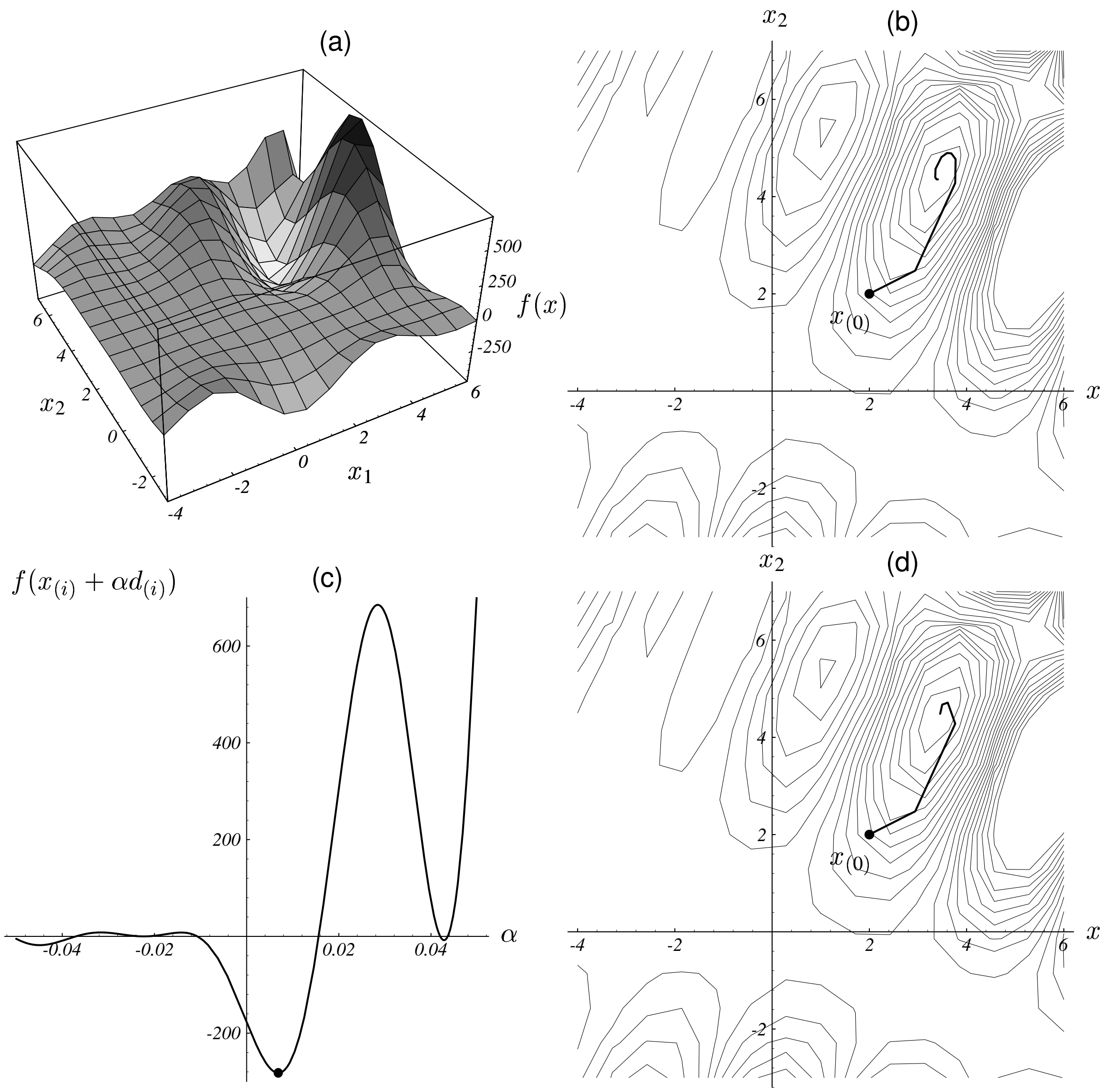


Figure 37: Convergence of the nonlinear Conjugate Gradient Method. (a) A complicated function with many local minima and maxima. (b) Convergence path of Fletcher-Reeves CG. Unlike linear CG, convergence does not occur in two steps. (c) Cross-section of the surface corresponding to the first line search.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

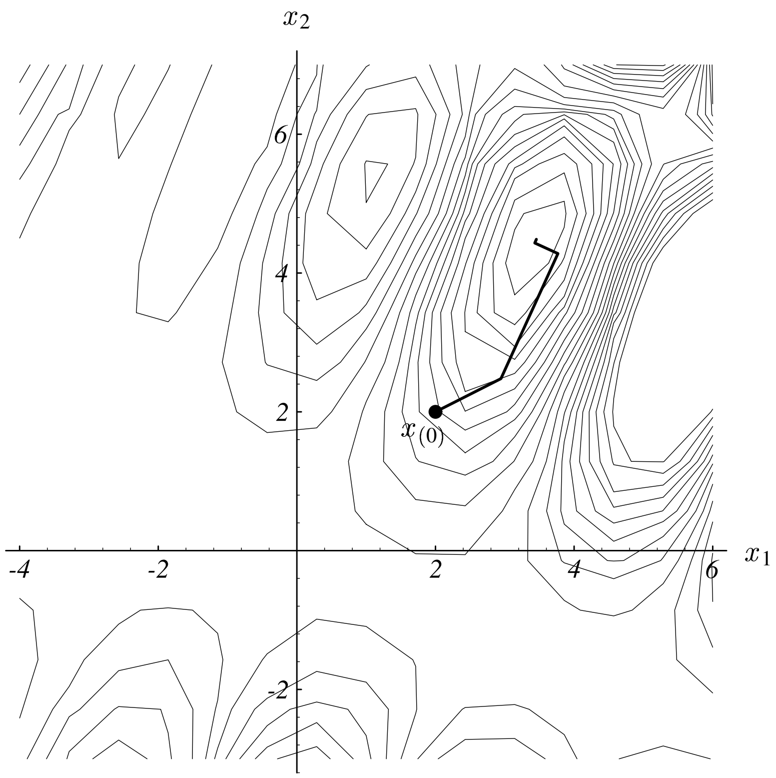


Figure 38: Nonlinear CG can be more effective with periodic restarts.

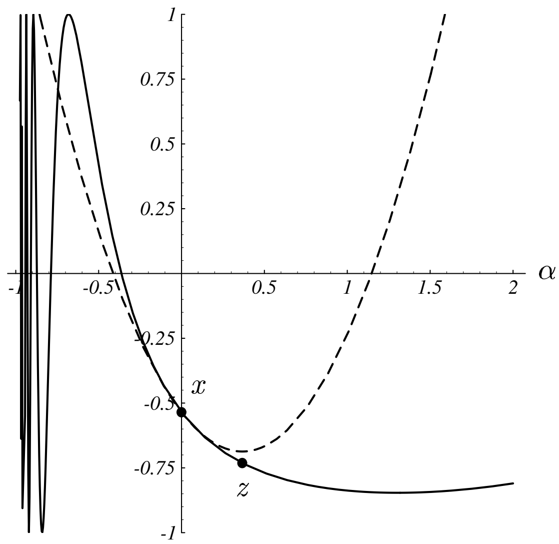


Figure 39: The Newton-Raphson method for minimizing a one-dimensional function (solid curve). Starting from the point x, calculate the first and second derivatives, and use them to construct a quadratic approximation to the function (dashed curve). A new point z is chosen at the base of the parabola.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

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

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

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

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

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

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

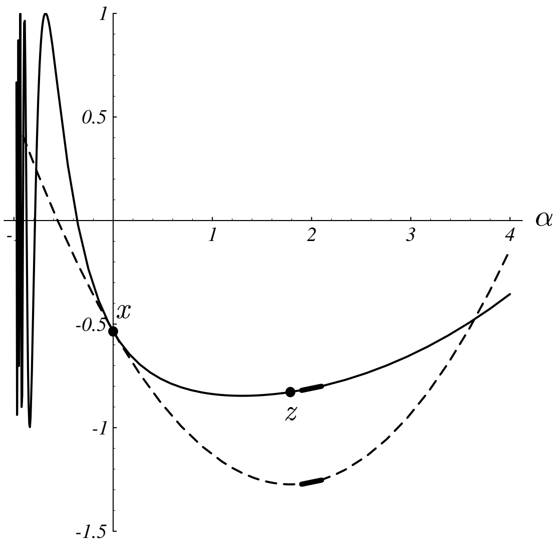


Figure 40: The Secant method for minimizing a one-dimensional function (solid curve). Starting from the point x, calculate the first derivatives at two different points (here, $\alpha = 0$ and $\alpha = 2$), and use them to construct a quadratic approximation to the function (dashed curve). Notice that 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

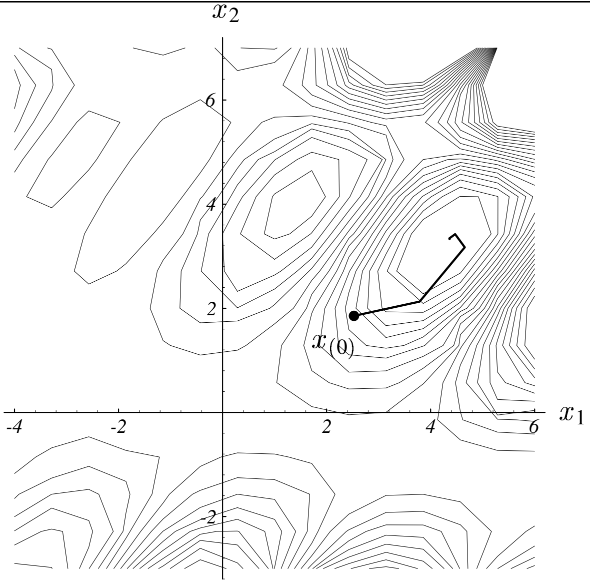


Figure 41: The preconditioned nonlinear Conjugate Gradient Method, using the Polak-Ribière formula and a diagonal preconditioner. The space has been "stretched" to show the improvement in circularity of the contour lines around the minimum.

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 ⇐ r^T r
δ_0 ⇐ δ
While i < i_max and δ > ε^2 δ_0 do
    q ⇐ Ar
    α ⇐ δ / (r^T q)
    x ⇐ x + αr
    If i is divisible by 50
        r ⇐ b − Ax
    else
        r ⇐ r − αq
    δ ⇐ r^T r
    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 $\varepsilon < 1$:

$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$d \Leftarrow M^{-1}r$
$\delta_{new} \Leftarrow r^T d$
$\delta_0 \Leftarrow \delta_{new}$
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
        $r \Leftarrow r - \alpha q$
    $s \Leftarrow M^{-1}r$
    $\delta_{old} \Leftarrow \delta_{new}$
    $\delta_{new} \Leftarrow r^T s$
    $\beta \Leftarrow \frac{\delta_{new}}{\delta_{old}}$
    $d \Leftarrow s + \beta d$
    $i \Leftarrow i + 1$

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 $\varepsilon < 1$, 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
δ_new ⇐ r^T r
δ_0 ⇐ δ_new
While i < i_max and δ_new > ε²δ_0 do
    j ⇐ 0
    δ_d ⇐ d^T d
    Do
        α ⇐ − [f′(x)]^T d / (d^T f″(x) d)
        x ⇐ x + αd
        j ⇐ j + 1
    while j < j_max and α²δ_d > ϵ²
    r ⇐ −f′(x)
    δ_old ⇐ δ_new
    δ_new ⇐ r^T r
    β ⇐ δ_new / δ_old
    d ⇐ r + βd
    k ⇐ k + 1
    If k = n or r^T d ≤ 0
        d ⇐ r
        k ⇐ 0
    i ⇐ i + 1
```

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

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 ≈ f''(x)
s ⇐ M^{-1} r
d ⇐ s
δ_new ⇐ r^T d
δ_0 ⇐ δ_new
While i < i_max and δ_new > ε^2 δ_0 do
    j ⇐ 0
    δ_d ⇐ d^T d
    α ⇐ −σ_0
    η_prev ⇐ [f'(x + σ_0 d)]^T d
    Do
        η ⇐ [f'(x)]^T d
        α ⇐ α η / (η_prev − η)
        x ⇐ x + αd
        η_prev ⇐ η
        j ⇐ j + 1
    while j < j_max and α^2 δ_d > ϵ^2
    r ⇐ −f'(x)
    δ_old ⇐ δ_new
    δ_mid ⇐ r^T s
    Calculate a preconditioner M ≈ f''(x)
    s ⇐ M^{-1} r
    δ_new ⇐ r^T s
    β ⇐ (δ_new − δ_mid) / δ_old
    k ⇐ k + 1
    If k = n or β ≤ 0
        d ⇐ s
        k ⇐ 0
    else
        d ⇐ s + βd
    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

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

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,

$$
\begin{aligned}
X^T A X &= \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \end{bmatrix} A \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} = \begin{bmatrix} x_1^T \\ x_2^T \\ x_3^T \\ x_4^T \end{bmatrix} \begin{bmatrix} \lambda_A x_1 & Ax_2 & Ax_3 & Ax_4 \end{bmatrix} \\
&= \begin{bmatrix} \lambda_A & 0 & 0 & 0 \\ 0 & & & \\ 0 & & B & \\ 0 & & & \end{bmatrix},
\end{aligned}
$$

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

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

In other words, $AX\widehat{w} = \lambda_B X\widehat{w}$, and thus $X\widehat{w}$ is an eigenvector of A. Furthermore, $x_1^T X\widehat{w} = [1 \;\; 0 \;\; 0 \;\; 0]\widehat{w} = 0$, so x_1 and $X\widehat{w}$ 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 $\widehat{w}_1$, $\widehat{w}_2$, and $\widehat{w}_3$. Then $X\widehat{w}_1$, $X\widehat{w}_2$, and $X\widehat{w}_3$ 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

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

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

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

and oscillate rapidly between -1 and 1:

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

Notice that the i zeros of T_i must fall between the $i+1$ extrema of T_i in the range $[-1, 1]$. For an example, see the five zeros of $T_5(\omega)$ in Figure 32.

Similarly, the function

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

oscillates in the range $\pm T_i(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}})^{-1}$ on the domain $[\lambda_{min}, \lambda_{max}]$. $P_i(\lambda)$ also satisfies the requirement that $P_i(0) = 1$.

The proof relies on a cute trick. There is no polynomial $Q_i(\lambda)$ of degree i such that $Q_i(0) = 1$ and Q_i is better than P_i on the range $[\lambda_{min}, \lambda_{max}]$. To prove this, suppose that there is such a polynomial; then, $Q_i(\lambda) < T_i(\frac{\lambda_{max}+\lambda_{min}}{\lambda_{max}-\lambda_{min}})^{-1}$ on the range $[\lambda_{min}, \lambda_{max}]$. It follows that the polynomial $P_i - Q_i$ has a zero at $\lambda = 0$, and also has i zeros on the range $[\lambda_{min}, \lambda_{max}]$. Hence, $P_i - Q_i$ is a polynomial of degree i with at least $i+1$ zeros, which is impossible. By contradiction, I conclude that the Chebyshev polynomial of degree i optimizes Expression 50.

D Homework

For the following questions, assume (unless otherwise stated) that you are using exact arithmetic; there is no floating point roundoff error.

1. (Easy.) Prove that the preconditioned Conjugate Gradient method is not affected if the preconditioning matrix is scaled. In other words, for any nonzero constant γ, show that the sequence of steps $x_{(0)}, x_{(1)}, \ldots$ taken using the preconditioner γM is identical to the steps taken using the preconditioner M.

2. (Hard, but interesting.) Suppose you wish to solve $Ax = b$ for a symmetric, positive-definite $n \times n$ matrix A. Unfortunately, the trauma of your linear algebra course has caused you to repress all memories of the Conjugate Gradient algorithm. Seeing you in distress, the Good Eigenfairy materializes and grants you a list of the d distinct eigenvalues (but not the eigenvectors) of A. However, you do **not** know how many times each eigenvalue is repeated.

   Clever person that you are, you mumbled the following algorithm in your sleep this morning:

   ```
   Choose an arbitrary starting point x_(0)
   For i ⇐ 0 to d − 1
       r_(i) ⇐ b − A x_(i)
       Remove an arbitrary eigenvalue from the list and call it λ_i
       x_(i+1) ⇐ x_(i) + λ_i^(−1) r_(i)
   ```

   No eigenvalue is used twice; on termination, the list is empty.

   (a) Show that upon termination of this algorithm, $x_{(d)}$ is the solution to $Ax = b$. **Hint:** Read Section 6.1 carefully. Graph the convergence of a two-dimensional example; draw the eigenvector axes, and try selecting each eigenvalue first. (It is most important to intuitively explain what the algorithm is doing; if you can, also give equations that prove it.)

(b) Although this routine finds the exact solution after d iterations, you would like each intermediate iterate $x_{(i)}$ to be as close to the solution as possible. Give a **crude** rule of thumb for how you should choose an eigenvalue from the list on each iteration. (In other words, in what order should the eigenvalues be used?)

(c) What could go terribly wrong with this algorithm if floating point roundoff occurs?

(d) Give a rule of thumb for how you should choose an eigenvalue from the list on each iteration to prevent floating point roundoff error from escalating. **Hint:** The answer is not the same as that of question (b).

References

[1] Richard Barrett, Michael Berry, Tony Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, SIAM, Philadelphia, Pennsylvania, 1993.

[2] William L. Briggs, *A Multigrid Tutorial*, SIAM, Philadelphia, Pennsylvania, 1987.

[3] James W. Daniel, *Convergence of the Conjugate Gradient Method with Computationally Convenient Modifications*, Numerische Mathematik **10** (1967), 125–131.

[4] W. C. Davidon, *Variable Metric Method for Minimization*, Tech. Report ANL-5990, Argonne National Laboratory, Argonne, Illinois, 1959.

[5] R. Fletcher and M. J. D. Powell, *A Rapidly Convergent Descent Method for Minimization*, Computer Journal **6** (1963), 163–168.

[6] R. Fletcher and C. M. Reeves, *Function Minimization by Conjugate Gradients*, Computer Journal **7** (1964), 149–154.

[7] L. Fox, H. D. Huskey, and J. H. Wilkinson, *Notes on the Solution of Algebraic Linear Simultaneous Equations*, Quarterly Journal of Mechanics and Applied Mathematics **1** (1948), 149–173.

[8] Jean Charles Gilbert and Jorge Nocedal, *Global Convergence Properties of Conjugate Gradient Methods for Optimization*, SIAM Journal on Optimization **2** (1992), no. 1, 21–42.

[9] Gene H. Golub and Dianne P. O'Leary, *Some History of the Conjugate Gradient and Lanczos Algorithms: 1948–1976*, SIAM Review **31** (1989), no. 1, 50–102.

[10] Magnus R. Hestenes, *Iterative Methods for Solving Linear Equations*, Journal of Optimization Theory and Applications **11** (1973), no. 4, 323–334, Originally published in 1951 as NAML Report No. 52-9, National Bureau of Standards, Washington, D.C.

[11] Magnus R. Hestenes and Eduard Stiefel, *Methods of Conjugate Gradients for Solving Linear Systems*, Journal of Research of the National Bureau of Standards **49** (1952), 409–436.

[12] Shmuel Kaniel, *Estimates for Some Computational Techniques in Linear Algebra*, Mathematics of Computation **20** (1966), 369–378.

[13] John K. Reid, *On the Method of Conjugate Gradients for the Solution of Large Sparse Systems of Linear Equations*, Large Sparse Sets of Linear Equations (London and New York) (John K. Reid, ed.), Academic Press, London and New York, 1971, pp. 231–254.

[14] E. Schmidt, *Title unknown*, Rendiconti del Circolo Matematico di Palermo **25** (1908), 53–77.

[15] Eduard Stiefel, *Über einige Methoden der Relaxationsrechnung*, Zeitschrift für Angewandte Mathematik und Physik **3** (1952), no. 1, 1–33.

[16] A. van der Sluis and H. A. van der Vorst, *The Rate of Convergence of Conjugate Gradients*, Numerische Mathematik **48** (1986), no. 5, 543–560.