Conjugate Gradient Methods

MA 348

Kurt Bryan

Introduction

We want to minimize a nonlinear twice continuously differentiable function of n variables. Taylor's Theorem shows that such functions are locally well-approximated by quadratic polynomials. It then seems reasonable that methods which do well on quadratics should do well on more general nonlinear functions, at least once we get close enough to a minimum. This simple observation was the basis for Newton's Method.

But Newton's Method has some drawbacks. First, we have to be able to compute the Hessian matrix, which is n by n, and we have to do this at every iteration. This is a serious problem for a function of 1000 variables, both in computational time and storage. Second, we have to solve a dense n by n system of equations, also time consuming. Finite difference approximations for the derivatives doesn't really resolve these issues.

It would be nice to come up with optimization algorithms that don't require use of the full Hessian, but which, like Newton's method, perform well on quadratic functions. This is what conjugate gradient methods do (and also quasi-Newton methods, which we'll talk about later). First, we'll study the efficient minimization of quadratic functions (without the use of second derivatives).

Quadratic Functions

Suppose that $f(\mathbf{x})$ is a quadratic function of n real variables in the form. Such a function can always be written in the form

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

for some n by n matrix $\mathbf{A}$, n vector $\mathbf{b}$ and scalar c. The $1/2$ in front isn't essential, but it make things a bit cleaner later. You should convince yourself of this (see exercise below). Also, there's no loss of generality in assuming that $\mathbf{A}$ is symmetric, because $(\mathbf{x}^T\mathbf{A}\mathbf{x})^T = \mathbf{x}^T\mathbf{A}^T\mathbf{x}$ for all $\mathbf{x}$, and so we have

$$\mathbf{x}^T\mathbf{A}\mathbf{x} = \mathbf{x}^T(\frac{1}{2}(\mathbf{A} + \mathbf{A}^T))\mathbf{x}$$

for all $\mathbf{x}$, and $\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$ is symmetric. In short, $\mathbf{A}$ can be replaced with $\frac{1}{2}(\mathbf{A} + \mathbf{A}^T)$.

Exercise 1

- Write $f(x_1, x_2) = x_1^2 + x_1x_2 + 4x_2^2 - 3x_1 + x_2 + 1$ in the form (1) with $\mathbf{A}$ symmetric.

We'll consider for now the problem of minimizing a function of the form (1) with $\mathbf{A}$ symmetric. It will be useful to note, as we did in deriving Newton's method, that

$$\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{2}$$

where ∇f is written as a column of partial derivatives. You can convince yourself of this fact by writing things out in summation notation,

$$f(\mathbf{x}) = c + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_i x_j + \sum_{i=1}^{n} b_i x_i$$

then differentiating with respect to x_k (and using the symmetry of $\mathbf{A}$, i.e., $A_{ik} = A_{ki}$) to obtain

$$\begin{aligned}\frac{\partial f}{\partial x_k} &= \frac{1}{2}\sum_{i=1}^{n} A_{ik}x_i + \frac{1}{2}\sum_{j=1}^{n} A_{kj}x_j + b_k \\ &= \sum_{i=1}^{n} A_{ki}x_i + b_k\end{aligned}$$

which is exactly the component-by-component statement of equation (2). You can use the same reasoning to check that the Hessian matrix for f is exactly $\mathbf{A}$.

We also will assume that $\mathbf{A}$ is positive definite. In this case f has a unique critical point, and this critical point is a minimum. To see this, note that since $\mathbf{A}$ is positive definite, $\mathbf{A}$ is invertible, and hence from (2) the only critical point is $-\mathbf{A}^{-1}\mathbf{b}$. As we showed in analyzing Newton's method, this must be a minimum if the Hessian of f (in this case just $\mathbf{A}$) is positive definite.

Minimizing Quadratic Functions

Let's start with an example. Let $n = 3$ and let $\mathbf{A} = 2\mathbf{I}$, the n by n identity matrix. In this case $f(\mathbf{x}) = x_1^2 + x_2^2 + x_3^2$. Of course the unique minimum is at the origin, but pretend we want to find it with the "one-variable at a time" algorithm, in which we take an initial guess and then do line searches in directions $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, and then repeat the cycle (here $\mathbf{e}_i$ is the standard coordinate vector in the x_i direction). If we start with initial guess $\mathbf{x}_0 = (1, 2, 3)$ and minimize in direction $\mathbf{e}_1$ (i.e., in the variable x_1) we find $\mathbf{x}_1 = (0, 2, 3)$ from this line search. Now do a line search from point $\mathbf{x}_1$ using search direction $\mathbf{e}_2$ to find $\mathbf{x}_2 = (0, 0, 3)$. Note an important fact: the second line search didn't mess up the first line search, in the sense that if we now repeated the $\mathbf{e}_1$ search from $\mathbf{x}_2$ we'd find we're ALREADY at the minimum. Of course the third line search from $\mathbf{x}_2$ in the direction $\mathbf{e}_3$ takes us to the origin,

and doesn't mess up the results from the previous line searches—we're still at a minimum with respect to x_1 and x_2, i.e., the global minimum. This approach has taken us to the minimum in exactly 3 iterations.

Contrast the above situation with that obtain by using this technique but instead on the function $f(x_1, x_2, x_3) = x_1^2 + x_1x_2 + x_2^2 + x_3^2$, again with $\mathbf{x}_0 = (1, 2, 3)$. The first line search in direction $\mathbf{e}_1$ takes us to $\mathbf{x}_1 = (-1, 2, 3)$. The second line search from $\mathbf{x}_1$ in direction $\mathbf{e}_2$ takes us to $\mathbf{x}_2 = (-1, 1/2, 3)$. But now note that the $\mathbf{e}_2$ search has "messed up" the first line search, in that if we now repeat a line search from $\mathbf{x}_2$ in the direction $\mathbf{e}_1$ we find we're no longer at a minimum in the x_1 variable (it has shifted to $x_1 = -1/4$).

The problem in the second case is that the search directions $\mathbf{e}_i$ partially conflict with each other, so each line search partly undoes the progress made by the previous line searches. This didn't happen for $f(\mathbf{x}) = \mathbf{x}^T\mathbf{I}\mathbf{x}$, and so we got to the minimum quickly.

Line Searches and Quadratic Functions

Most of the algorithms we've developed for finding minima involve a series of line searches. Consider performing a line search on the function $f(\mathbf{x}) = c + \mathbf{x}^T\mathbf{b} + \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$ from some base point $\mathbf{a}$ in the direction $\mathbf{v}$, i.e., minimizing f along the line $\mathbf{L}(t) = \mathbf{a} + t\mathbf{v}$. This amounts to minimizing the function

$$g(t) = f(\mathbf{a} + t\mathbf{v}) = \frac{1}{2}\mathbf{v}^T\mathbf{A}\mathbf{v}t^2 + (\mathbf{v}^T\mathbf{A}\mathbf{a} + \mathbf{v}^T\mathbf{b})t + \frac{1}{2}\mathbf{a}^T\mathbf{A}\mathbf{a} + \mathbf{a}^T\mathbf{b} + c. \tag{3}$$

Since $\frac{1}{2}\mathbf{v}^T\mathbf{A}\mathbf{v} > 0$ ($\mathbf{A}$ is positive definite), the quadratic function $g(t)$ ALWAYS has a unique global minimum in t, for g is quadratic in t with a positive coefficient in front of t^2. The minimum occurs when $g'(t) = (\mathbf{v}^T\mathbf{A}\mathbf{v})t + (\mathbf{v}^T\mathbf{A}\mathbf{a} + \mathbf{v}^T\mathbf{b}) = 0$, i.e., at $t = t^*$ where

$$t^* = -\frac{\mathbf{v}^T(\mathbf{A}\mathbf{a} + \mathbf{b})}{\mathbf{v}^T\mathbf{A}\mathbf{v}} = -\frac{\mathbf{v}^T\nabla f(\mathbf{a})}{\mathbf{v}^T\mathbf{A}\mathbf{v}}. \tag{4}$$

Note that if $\mathbf{v}$ is a descent direction ($\mathbf{v}^T\nabla f(\mathbf{a}) < 0$) then $t^* > 0$.

Another thing to note is this: Let $\mathbf{p} = \mathbf{a} + t^*\mathbf{v}$, so $\mathbf{p}$ is the minimum of f on the line $\mathbf{a} + t\mathbf{v}$. Since $g'(t) = \nabla f(\mathbf{a} + t\mathbf{v}) \cdot \mathbf{v}$ and $g'(t^*) = 0$, we see that the minimizer $\mathbf{p}$ is the unique point on the line $\mathbf{a} + t\mathbf{v}$ for which (note $\mathbf{v}^T\nabla f = \mathbf{v} \cdot \nabla f$)

$$\nabla f(\mathbf{p}) \cdot \mathbf{v} = 0. \tag{5}$$

Thus if we are at some point $\mathbf{b}$ in $\mathbb{R}^n$ and want to test whether we're at a minimum for a quadratic function f with respect to some direction $\mathbf{v}$, we merely test whether $\nabla f(\mathbf{b}) \cdot \mathbf{v} = 0$.

Conjugate Directions

Look back at the example for minimizing $f(\mathbf{x}) = \mathbf{x}^T\mathbf{I}\mathbf{x} = x_1^2 + x_2^2 + x_3^2$. We minimized in direction $\mathbf{e}_1 = (1, 0, 0)$, and then in direction $\mathbf{e}_2 = (0, 1, 0)$, and arrived at the point $\mathbf{x}_2 = (0, 0, 3)$. Did the $\mathbf{e}_2$ line search mess up the fact that we were at a minimum with respect to $\mathbf{e}_1$? No! You can check that $\nabla f(\mathbf{x}_2) \cdot \mathbf{e}_1 = 0$, that is, $\mathbf{x}_2$ minimizes f in BOTH directions $\mathbf{e}_1$ AND $\mathbf{e}_2$. For this function $\mathbf{e}_1$ and $\mathbf{e}_2$ are "non-interfering". Of course $\mathbf{e}_3$ shares this property.

But now look at the second example, of minimizing $f(\mathbf{x}) = x_1^2 + x_1x_2 + x_2^2 + x_3^2$. After minimizing in $\mathbf{e}_1$ and $\mathbf{e}_2$ we find ourselves at the point $\mathbf{x}_2 = (-1, 1/2, 3)$. Of course $\nabla f(\mathbf{x}_2) \cdot \mathbf{e}_2 = 0$ (that's how we got to $\mathbf{x}_2$ from $\mathbf{x}_1$—by minimizing in direction $\mathbf{e}_2$). But the minimization in direction $\mathbf{e}_2$ has destroyed the fact that we're at a minimum with respect to direction $\mathbf{e}_1$: we find that $\nabla f(\mathbf{x}_2) \cdot \mathbf{e}_1 = -3/2$. So at some point we'll have to redo the $\mathbf{e}_1$ minimization again, which will mess up the $\mathbf{e}_2$ minimization, and probably $\mathbf{e}_3$ too. Every direction in the set $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ conflicts with every other direction.

The idea behind the conjugate direction approach for minimizing quadratic functions is to use search directions which don't interfere with one another. Given a symmetric positive definite matrix $\mathbf{A}$, we will call a set of vectors $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}$ *conjugate* (or "$\mathbf{A}$ conjugate", or even "$\mathbf{A}$ orthogonal") if

$$\mathbf{d}_i^T\mathbf{A}\mathbf{d}_j = 0 \tag{6}$$

for $i \neq j$. Note that $\mathbf{d}_i^T\mathbf{A}\mathbf{d}_i > 0$ for all i, since $\mathbf{A}$ is positive definite.

Theorem 1: If the set $S = \{\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_{n-1}\}$ is conjugate and all $\mathbf{d}_i$ are non-zero then S forms a basis for $\mathbb{R}^n$.

Proof: Since S has n vectors, all we need to show is that S is linearly independent, since it then automatically spans $\mathbb{R}^n$. Start with

$$c_1\mathbf{d}_1 + c_2\mathbf{d}_2 + \cdots + c_n\mathbf{d}_n = \mathbf{0}.$$

Multiply on the right by $\mathbf{A}$, distribute over the sum, and then multiply the whole mess by $\mathbf{d}_i^T$ to obtain

$$c_i\mathbf{d}_i^T\mathbf{A}\mathbf{d}_i = 0.$$

We immediately conclude that $c_i = 0$, so the set S is linearly independent and hence a basis for $\mathbb{R}^n$.

Example 1: Let

$$\mathbf{A} = \begin{bmatrix} 3 & 1 & 0 \\ 1 & 2 & 2 \\ 0 & 2 & 4 \end{bmatrix}.$$

Choose $\mathbf{d}_0 = (1, 0, 0)^T$. We want to choose $\mathbf{d}_1 = (a, b, c)$ so that $\mathbf{d}_1^T \mathbf{A} \mathbf{d}_0 = 0$, which leads to $3a + b = 0$. So choose $\mathbf{d}_1 = (1, -3, 0)$. Now let $\mathbf{d}_2 = (a, b, c)$ and require $\mathbf{d}_2^T \mathbf{A} \mathbf{d}_0 = 0$ and $\mathbf{d}_2^T \mathbf{A} \mathbf{d}_0 = 0$, which yields $3a + b = 0$ and $-5b - 6c = 0$. This can be written as $a = -b/3$ and $c = -5b/6$. We can take $\mathbf{d}_2 = (-2, 6, -5)$.

Here's a simple version of a conjugate direction algorithm for minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{x}^T \mathbf{b} + c$, assuming we already have a set of conjugate directions $\mathbf{d}_i$:

1. Make an initial guess $\mathbf{x}_0$. Set $k = 0$.

2. Perform a line search from $\mathbf{x}_k$ in the direction $\mathbf{d}_k$. Let $\mathbf{x}_{k+1}$ be the unique minimum of f along this line.

3. If $k = n - 1$, terminate with minimum x_n, else increment k and return to the previous step.

Note that the assertion is that the above algorithm locates the minimum in n line searches. It's also worth pointing out that from equation (5) (with $\mathbf{p} = \mathbf{x}_{k+1}$ and $\mathbf{v} = \mathbf{d}_k$) we have

$$\nabla f(\mathbf{x}_{k+1}) \cdot \mathbf{d}_k = 0 \tag{7}$$

after each line search.

Theorem 2: For a quadratic function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{b}^T \mathbf{x} + c$ with $\mathbf{A}$ positive definite the conjugate direction algorithm above locates the minimum of f in (at most) n line searches.

Proof:

We'll show that $\mathbf{x}_n$ is the minimizer of f by showing $\nabla f(\mathbf{x}_n) = \mathbf{0}$. Since f has a unique critical point, this shows that $\mathbf{x}_n$ is it, and hence the minimum.

First, the line search in step two of the above algorithm consists of minimizing the function

$$g(t) = f(\mathbf{x}_k + t\mathbf{d}_k). \tag{8}$$

Equations (3) and (4) (with $\mathbf{v} = \mathbf{d}_k$ and $\mathbf{a} = \mathbf{x}_k$) show that

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k \tag{9}$$

where

$$\alpha_k = -\frac{\mathbf{d}_k^T \mathbf{A} \mathbf{x}_k + \mathbf{d}_k^T \mathbf{b}}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k} = -\frac{\mathbf{d}_k^T \nabla f(\mathbf{x}_k)}{\mathbf{d}_k^T \mathbf{A} \mathbf{d}_k}. \tag{10}$$

Repeated use of equation (9) shows that for any k with $0 \leq k \leq n - 1$ we have

$$\mathbf{x}_n = \mathbf{x}_{k+1} + \alpha_{k+1}\mathbf{d}_{k+1} + \alpha_{k+2}\mathbf{d}_{k+2} + \cdots + \alpha_{n-1}\mathbf{d}_{n-1}. \tag{11}$$

Since $\nabla f(\mathbf{x}_n) = A\mathbf{x}_n + \mathbf{b}$ we have, by making use of (11),

$$\begin{aligned} \nabla f(\mathbf{x}_n) &= \mathbf{A}\mathbf{x}_{k+1} + \mathbf{b} + \alpha_{k+1}\mathbf{A}\mathbf{d}_{k+1} + \cdots \alpha_{n-1}\mathbf{A}\mathbf{d}_{n-1} \\ &= \nabla f(\mathbf{x}_{k+1}) + \sum_{i=k+1}^{n-1} \alpha_i \mathbf{A}\mathbf{d}_i. \end{aligned} \tag{12}$$

Multiply both sides of equation (12) by $\mathbf{d}_k^T$ to obtain

$$\mathbf{d}_k^T \nabla f(\mathbf{x}_n) = \mathbf{d}_k^T \nabla f(\mathbf{x}_{k+1}) + \sum_{i=k+1}^{n-1} \alpha_i \mathbf{d}_k^T \mathbf{A}\mathbf{d}_i.$$

But equation (7) and the conjugacy of the directions $\mathbf{d}_i$ immediately yield

$$\mathbf{d}_k^T \nabla f(\mathbf{x}_n) = 0$$

for $k = 0$ to $k = n-1$. Since the $\mathbf{d}_k$ form a basis, we conclude that $\nabla f(\mathbf{x}_n) = \mathbf{0}$, so $\mathbf{x}_n$ is the unique critical point (and minimizer) of f.

Example 2: For case in $\mathbb{R}^3$ in which $\mathbf{A} = \mathbf{I}$, it's easy to see that the directions $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$ are conjugate, and so the one-variable-at-a-time approach works, in at most 3 line searches.

Example 3: Let $\mathbf{A}$ be the matrix in Example 1, $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x}$, and we already computed $\mathbf{d}_0 = (1, 0, 0)$, $\mathbf{d}_1 = (1, -3, 0)$, and $\mathbf{d}_2 = (-2, 6, -5)$. Take initial guess $\mathbf{x}_0 = (1, 2, 3)$. A line search from $\mathbf{x}_0$ in direction $\mathbf{d}_0$ requires us to minimize

$$g(t) = f(\mathbf{x}_0 + t\mathbf{d}_0) = \frac{3}{2}t^2 + 5t + \frac{75}{2}$$

which occurs at $t = -5/3$, yielding $\mathbf{x}_1 = \mathbf{x}_0 - \frac{5}{3}\mathbf{d}_0 = (-2/3, 2, 3)$. A line search from $\mathbf{x}_1$ in direction $\mathbf{d}_1$ requires us to minimize

$$g(t) = f(\mathbf{x}_1 + t\mathbf{d}_1) = \frac{15}{2}t^2 - 28t + \frac{100}{3}$$

which occurs at $t = 28/15$, yielding $\mathbf{x}_2 = \mathbf{x}_1 + \frac{28}{15}\mathbf{d}_1 = (6/5, -18/5, 3)$. The final line search from $\mathbf{x}_2$ in direction $\mathbf{d}_2$ requires us to minimize

$$g(t) = f(\mathbf{x}_2 + t\mathbf{d}_2) = 20t^2 - 24t + \frac{36}{5}$$

which occurs at $t = 3/5$, yielding $\mathbf{x}_3 = \mathbf{x}_2 + \frac{3}{5}\mathbf{d}_2 = (0, 0, 0)$, which is of course correct.

Exercise 2

- Show that for each k, $\mathbf{d}_j^T \nabla f(\mathbf{x}_k) = 0$ for $j = 0 \ldots k-1$ (i.e., the gradient $\nabla f(\mathbf{x}_k)$ is orthogonal to not only the previous search direction $\mathbf{d}_{k-1}$, but ALL previous search directions). Hint: For any $0 \leq k < n$, write

$$\mathbf{x}_k = \mathbf{x}_{j+1} + \alpha_{j+1}\mathbf{d}_{j+1} + \cdots + \alpha_{k-1}\mathbf{d}_{k-1}$$

  where $0 \leq j < k$. Multiply by $\mathbf{A}$ and follow the reasoning before (12) to get

$$\nabla f(\mathbf{x}_k) = \nabla f(\mathbf{x}_{j+1}) + \sum_{i=j+1}^{k-1} \alpha_i \mathbf{A}\mathbf{d}_i.$$

  Now multiply both sides by $\mathbf{d}_j^T$, $0 \leq j < k$.

Conjugate Gradients

One drawback to the algorithm above is that we need an entire set of conjugate directions $\mathbf{d}_i$ before we start, and computing them in the manner of Example 1 would be a lot of work—more than simply solving $\mathbf{A}\mathbf{x} = -\mathbf{b}$ for the critical point.

Here's a way to generate the conjugate directions "on the fly" as needed, with a minimal amount of computation. First, we take $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0) = -(\mathbf{A}\mathbf{x}_0 + \mathbf{b})$. We then perform the line minimization to obtain point $\mathbf{x}_1 = \mathbf{x}_0 + \alpha_0 \mathbf{d}_0$, with α_0 given by equation (10).

The next direction $\mathbf{d}_1$ is computed as a linear combination $\mathbf{d}_1 = -\nabla f(\mathbf{x}_1) + \beta_0 \mathbf{d}_0$ of the current gradient and the last search direction. We choose β_0 so that $\mathbf{d}_1^T \mathbf{A}\mathbf{d}_0 = 0$, which gives

$$\beta_0 = \frac{\nabla f(\mathbf{x}_1)^T \mathbf{A}\mathbf{d}_0}{\mathbf{d}_0^T \mathbf{A}\mathbf{d}_0}.$$

Of course $\mathbf{d}_0$ and $\mathbf{d}_1$ form a conjugate set of directions, by construction. We then perform a line search from $\mathbf{x}_1$ in the direction $\mathbf{d}_1$ to locate $\mathbf{x}_2$.

In general we compute the search direction $\mathbf{d}_k$ as a linear combination

$$\mathbf{d}_k = -\nabla f(\mathbf{x}_k) + \beta_{k-1}\mathbf{d}_{k-1} \tag{13}$$

of the current gradient and the last search direction. We choose β_{k-1} so that $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{k-1} = 0$, which forces

$$\beta_{k-1} = \frac{\nabla f(\mathbf{x}_k)^T \mathbf{A}\mathbf{d}_{k-1}}{\mathbf{d}_{k-1}^T \mathbf{A}\mathbf{d}_{k-1}}. \tag{14}$$

We then perform a line search from $\mathbf{x}_k$ in the direction $\mathbf{d}_k$ to locate $\mathbf{x}_{k+1}$, and then repeat the whole procedure. This is a traditional *conjugate gradient* algorithm.

By construction we have $\mathbf{d}_k^T \mathbf{A} \mathbf{d}_{k-1} = 0$, that is, each search direction is conjugate to the previous direction, but we need $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$ for ALL choices of $i \neq j$. It turns out that this is true!

Theorem 3: The set of search directions defined by equations (13) and (14) for $k = 1$ to $k = n$ (with $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$) satisfy $\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0$, and so are conjugate.

Proof: We'll do this by induction. We've already seen that $\mathbf{d}_0$ and $\mathbf{d}_1$ are conjugate. This is the base case for the induction.

Now suppose that the set $\mathbf{d}_0, \mathbf{d}_1, \ldots, \mathbf{d}_m$ are all conjugate. We'll show that $\mathbf{d}_{m+1}$ is conjugate to each of these directions.

First, it will be convenient to note that if the directions $\mathbf{d}_k$ are generated according to (13) and (14) then $\nabla f(\mathbf{x}_{k+1})$ is orthogonal to $\nabla f(\mathbf{x}_j)$ for $0 \leq j \leq k \leq m$. To see this rewrite equation (13) as

$$\nabla f(\mathbf{x}_j) = \beta_{j-1} \mathbf{d}_{j-1} - \mathbf{d}_j.$$

Multiply both sides by $\nabla f(\mathbf{x}_{k+1})^T$ to obtain

$$\nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_j) = \beta_{j-1} \nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_{j-1} - \nabla f(\mathbf{x}_{k+1})^T \mathbf{d}_j.$$

But from the induction hypothesis (the directions $\mathbf{d}_0, \ldots, \mathbf{d}_m$ are already known to be conjugate) and Exercise 2 it follows that the right side above is zero, so

$$\nabla f(\mathbf{x}_{k+1})^T \nabla f(\mathbf{x}_j) = 0 \tag{15}$$

for $0 \leq j \leq k$ as asserted.

Now we can finish the proof. We already know that $\mathbf{d}_{m+1}$ is conjugate to $\mathbf{d}_m$, so we need only show that $\mathbf{d}_{m+1}$ is conjugate to $\mathbf{d}_k$ for $0 \leq k < m$. To this end, recall that

$$\mathbf{d}_{m+1} = -\nabla f(\mathbf{x}_{m+1}) + \beta_m \mathbf{d}_m.$$

Multiply both sides by $\mathbf{d}_k^T \mathbf{A}$ where $0 \leq k < m$ to obtain

$$\mathbf{d}_k^T \mathbf{A} \mathbf{d}_{m+1} = -\mathbf{d}_k^T \mathbf{A} \nabla f(\mathbf{x}_{m+1}) + \beta_m \mathbf{d}_k^T \mathbf{A} \mathbf{d}_m.$$

But $\mathbf{d}_k^T \mathbf{A} \mathbf{d}_m = 0$ by the induction hypothesis, so we have

$$\mathbf{d}_k^T \mathbf{A} \mathbf{d}_{m+1} = -\mathbf{d}_k^T \mathbf{A} \nabla f(\mathbf{x}_{m+1}) = -\nabla f(\mathbf{x}_{m+1})^T \mathbf{A} \mathbf{d}_k. \tag{16}$$

We need to show that the right side of equation (16) is zero.

First, start with equation (9) and multiply by $\mathbf{A}$ to obtain $\mathbf{A}\mathbf{x}_{k+1} - \mathbf{A}\mathbf{x}_k = \alpha_k \mathbf{A} \mathbf{d}_k$, or $(\mathbf{A}\mathbf{x}_{j+k} + \mathbf{b}) - (\mathbf{A}\mathbf{x}_k + \mathbf{b}) = \alpha_k \mathbf{A} \mathbf{d}_k$. But since $\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$, we have

$$\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) = \alpha_k \mathbf{A} \mathbf{d}_k.$$

or

$$\mathbf{A}\mathbf{d}_k = \frac{1}{\alpha_k}(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) \tag{17}$$

Substitute the right side of (17) into equation (16) for $\mathbf{A}\mathbf{d}_k$ to obtain

$$\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{m+1} = -\frac{1}{\alpha_k}(\nabla f(\mathbf{x}_{m+1})^T \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_{m+1})^T \nabla f(\mathbf{x}_k))$$

for $k < m$. From equation (15) the right side above is zero, so $\mathbf{d}_k^T \mathbf{A}\mathbf{d}_{m+1} = 0$ for $k < m$. Since $\mathbf{d}_m^T \mathbf{A}\mathbf{d}_{m+1} = 0$ by construction, we've shown that if $\mathbf{d}_0, \ldots, \mathbf{d}_m$ satisfies $\mathbf{d}_i^T \mathbf{A}\mathbf{d}_j = 0$ for $i \neq j$ then so does $\mathbf{d}_0, \ldots, \mathbf{d}_m, \mathbf{d}_{m+1}$. From induction it follows that $\mathbf{d}_0, \ldots, \mathbf{d}_n$ also has this property. This completes the proof.

Exercise 3

- It may turn out that all $\mathbf{d}_i = \mathbf{0}$ for $i \geq m$ for some $m < n$—namely, if the mth line search takes us to the minimum. However, show that if we're not at the minimum at iteration k (so $\nabla f(\mathbf{x}_k) \neq \mathbf{0}$) then $\mathbf{d}_k \neq \mathbf{0}$.

Here's the "pseudocode" for this conjugate gradient algorithm for minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{b} + c$:

1. Make an initial guess $\mathbf{x}_0$. Set $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$ (note $\nabla f(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$) and $k = 0$.
2. Let $\mathbf{x}_{k+1}$ be the unique minimum of f along the line $\mathbf{L}(t) = \mathbf{x}_k + t\mathbf{d}_k$, given by $\mathbf{x}_{k+1} = \mathbf{x}_k + t_k\mathbf{d}_k$ where
$$t_k = -\frac{\mathbf{d}_k^T \nabla f(\mathbf{x}_k)}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k} = -\frac{\mathbf{d}_k^T(\mathbf{A}\mathbf{x}_k + \mathbf{b})}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}.$$
3. If $k = n - 1$, terminate with minimum $\mathbf{x}_n$.
4. Compute new search direction $\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k$ where
$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T \mathbf{A}\mathbf{d}_k}{\mathbf{d}_k^T \mathbf{A}\mathbf{d}_k}. \tag{18}$$
Increment k and return to step 2.

Exercise 4:

- Run the conjugate gradient algorithm above on the function $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{b}$ with
$$\mathbf{A} = \begin{bmatrix} 3 & 0 & 2 \\ 0 & 1 & 1 \\ 2 & 1 & 3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}.$$
Use initial guess $(1, 1, 1)$.

An Application

As discussed earlier, if $\mathbf{A}$ is symmetric positive definite, the problems of minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{b} + c$ and solving $\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0}$ are completely equivalent. Systems like $\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0}$ with positive definite matrices occur quite commonly in applied math, especially when one is solving partial differential equations numerically. They are also frequently LARGE, with thousands to even millions of variables.

However, the matrix $\mathbf{A}$ is usually *sparse*, that is, it contains mostly zeros. Typically the matrix will have only a handful of non-zero elements in each row. This is great news for storage—storing an entire n by n matrix requires $8n^2$ bytes in double precision, but if $\mathbf{A}$ has only a few non-zero elements per row, say 5, then we can store $\mathbf{A}$ in a little more than $5n$ bytes.

Still, solving $\mathbf{A}\mathbf{x} + \mathbf{b} = \mathbf{0}$ is a challenge. You can't use standard techniques like LU or Cholesky decomposition—the usual implementations require one to store on the order of n^2 numbers, at least as the algorithms progress.

A better way is to use conjugate gradient techniques, to solve $\mathbf{A}\mathbf{x} = -\mathbf{b}$ by minimizing $\frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} + \mathbf{x}^T\mathbf{b} + c$. If you look at the algorithm carefully you'll see that we don't need to store the matrix $\mathbf{A}$ in any conventional way. All we really need is the ability to compute $\mathbf{A}\mathbf{x}$ for a given vector $\mathbf{x}$.

As an example, suppose that $\mathbf{A}$ is an n by n matrix with all 4's on the diagonal, -1's on the closest fringes, and zero everywhere else, so $\mathbf{A}$ looks like

$$\mathbf{A} = \begin{bmatrix} 4 & -1 & 0 & 0 & 0 \\ -1 & 4 & -1 & 0 & 0 \\ 0 & -1 & 4 & -1 & 0 \\ 0 & 0 & -1 & 4 & -1 \\ 0 & 0 & 0 & -1 & 4 \end{bmatrix}$$

in the five by five case. But try to imagine the 10,000 by 10,000 case. We could still store such a matrix in only about 240K of memory (double precision, 8 bytes per entry). This matrix is clearly symmetric, and it turns out it's positive definite.

There are different data structures for describing sparse matrices. One obvious way results in the first row of the matrix above being encoded as $[[1, 4.0], [2, -1.0]]$. The $[1, 4.0]$ piece means the first column in the row is 4.0, and the $[2, -1.0]$ means the second column entry in the first row is -1.0. The other elements in row 1 are assumed to be zero. The rest of the rows would be encoded as $[[1, -1.0], [2, -4.0], [3, -1.0]]$, $[[2, -1.0], [3, 4.0], [4, -1.0]]$, $[[3, -1.0], [4, 4.0], [5, -1.0]]$, and $[[4, -1.0], [5, 4.0]]$. If you stored a matrix this way you could fairly easily write a routing to compute the product $\mathbf{A}\mathbf{x}$ for any vector $\mathbf{x}$, and so use the conjugate gradient algorithm to solve $\mathbf{A}\mathbf{x} = -\mathbf{b}$.

A couple last words: Although the solution will be found exactly in n iterations (modulo round-off) people don't typically run conjugate gradient (or any other iterative solver) for

the full distance. Usually some small percentage of the full n iterations is taken, for by then you're usually sufficiently close to the solution. Also, it turns out that the performance of conjugate gradient, as well as many other linear solvers, can be improved by *preconditioning*. This means replacing the system $\mathbf{Ax} = -\mathbf{b}$ by $(\mathbf{BA})\mathbf{x} = -\mathbf{Bb}$ where $\mathbf{B}$ is some easily computed approximate inverse for $\mathbf{A}$ (it can be pretty approximate). It turns out that conjugate gradient performs best when the matrix involved is close to the identity in some sense, so preconditioning can decrease the number of iterations needed in conjugate gradients. Of course the ultimate preconditioner would be to take $\mathbf{B} = \mathbf{A}^{-1}$, but this would defeat the whole purpose of the iterative method, and be far more work!

Non-quadratic Functions

The algorithm above only works for quadratic functions, since it explicitly uses the matrix $\mathbf{A}$. We're going to modify the algorithm so that the specific quadratic nature of the objective function does not appear explicitly—in fact, only ∇f will appear, but the algorithm will remain unchanged if f is truly quadratic. However, since only ∇f will appear, the algorithm will immediately generalize to any function f for which we can compute ∇f.

Here are the modifications. The first step in the algorithm on page 9 involves computing $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$. There's no mention of $\mathbf{A}$ here—we can compute ∇f for any differentiable function.

In step 2 of the algorithm on page 9 we do a line search from $\mathbf{x}_k$ in direction $\mathbf{d}_k$. For the quadratic case we have the luxury of a simple formula (involving t_k) for the unique minimum. But the computation of t_k involves $\mathbf{A}$. Let's replace this formula with a general (exact) line search, using Golden Section or whatever line search method you like. This will change nothing in the quadratic case, but generalizes things, in that we know how to do line searches for any function. Note this gets rid of any explicit mention of $\mathbf{A}$ in step 2.

Step 4 is the only other place we use $\mathbf{A}$, in which we need to compute $\mathbf{Ad}_k$ in order to compute β_k. There are several ways to modify this to eliminate explicit mention of $\mathbf{A}$. First (still thinking of f as quadratic) note that since (from step 2) we have $\mathbf{x}_{k+1} - \mathbf{x}_k = c\mathbf{d}_k$ for some constant c (where c depends on how far we move to get from $\mathbf{x}_k$ to $\mathbf{x}_{k+1}$) we have

$$\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k) = (\mathbf{Ax}_{k+1} + \mathbf{b}) - (\mathbf{Ax}_k + \mathbf{b}) = c\mathbf{Ad}_k. \tag{19}$$

Thus $\mathbf{Ad}_k = \frac{1}{c}(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))$. Use this fact in the numerator and denominator of the definition for β_k in step 4 to obtain

$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{\mathbf{d}_k^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}. \tag{20}$$

Note that the value of c was irrelevant! The search direction $\mathbf{d}_{k+1}$ is as before, $\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k$. If f is truly quadratic then definitions (18) and (20) are equivalent. But the new algorithm can be run for ANY differentiable function.

Equation (20) is the *Hestenes-Stiefel* formula. It yields one version of the conjugate gradient algorithm for non-quadratic problems.

Here's another way to get rid of $\mathbf{A}$. Again, assume f is quadratic. Multiply out the denominator in (20) to find

$$\mathbf{d}_k^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) = \mathbf{d}_k^T \nabla f(\mathbf{x}_{k+1}) - \mathbf{d}_k^T \nabla f(\mathbf{x}_k).$$

But from equation (7) (or Exercise 2) we have $\mathbf{d}_k^T \nabla f(\mathbf{x}_{k+1}) = 0$, so really

$$\mathbf{d}_k^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) = -\mathbf{d}_k^T \nabla f(\mathbf{x}_k).$$

Now note $\mathbf{d}_k = -\nabla f(\mathbf{x}_k) + \beta_{k-1}\mathbf{d}_{k-1}$ so that

$$\mathbf{d}_k^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k)) = -(-\nabla f(\mathbf{x}_k)^T + \beta_{k-1}\mathbf{d}_{k-1}^T)\nabla f(\mathbf{x}_k) = |\nabla f(\mathbf{x}_k)|^2$$

since by equation (7) again, $\mathbf{d}_{k-1}^T \nabla f(\mathbf{x}_k) = 0$. All in all, the denominator in (20) is, in the quadratic case, $|\nabla f(\mathbf{x}_k)|^2$. We can thus give an alternate definition of β_k as

$$\beta_k = \frac{\nabla f(\mathbf{x}_{k+1})^T(\nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k))}{|\nabla f(\mathbf{x}_k)|^2}. \tag{21}$$

Again, no change occurs if f is quadratic, but this formula generalizes to the non-quadratic case. This formula is called the *Polak-Ribiere* formula.

Exercise 5

- Derive the *Fletcher-Reeves* formula (assuming f is quadratic)

$$\beta_k = \frac{|\nabla f(\mathbf{x}_{k+1})|^2}{|\nabla f(\mathbf{x}_k)|^2}. \tag{22}$$

Formulas (20), (21), and (22) all give a way to generalize the algorithm to any differentiable function. They require only gradient evaluations. But they are NOT equivalent for nonquadratic functions. However, all are used in practice (though Polak-Ribiere and Fletcher-Reeves seem to be the most common). Our general conjugate gradient algorithm now looks like this:

1. Make an initial guess $\mathbf{x}_0$. Set $\mathbf{d}_0 = -\nabla f(\mathbf{x}_0)$ and $k = 0$.
2. Do a line search from $\mathbf{x}_k$ in the direction $\mathbf{d}_k$. Let $\mathbf{x}_{k+1}$ be the minimum found along the line.
3. If a termination criteria is met, terminate with minimum $\mathbf{x}_{k+1}$.

4. Compute new search direction $\mathbf{d}_{k+1} = -\nabla f(\mathbf{x}_{k+1}) + \beta_k \mathbf{d}_k$ where β_k is given by one of equations (20), (21), or (22). Increment k and return to step 2.

Exercise 6

- Verify that all three of the conjugate gradient algorithms are descent methods for any function f. Hint: it's really easy.

Some people advocate periodic "restarting" of conjugate gradient methods. After some number of iterations (n or some fraction of n) we re-initialize $\mathbf{d}_k = -\nabla f(\mathbf{x}_k)$. There's some evidence this improves the performance of the algorithm.