Notes on Some Methods for Solving Linear Systems

Dianne P. O'Leary, 1983 and 1999 and 2007

September 25, 2007

When the matrix A is symmetric and positive definite, we have a whole new class of algorithms for solving $Ax^* = b$. Consider the function

$$f(x) = \frac{1}{2}x^T Ax - x^T b\,.$$

Notice that in one dimension, this defines a parabola, and if x is a 2-vector, it defines a bowl-shaped function with elliptical horozontal cross sections. (The bowl fails to hold water if any eigenvalue of A is negative.)

The solution to the problem

$$\min_x f(x) \tag{1}$$

is given by the vector satisfying

$$\nabla f(x) = Ax - b = 0\,.$$

(Note that $f(x)$ is the negative of what we have been calling the residual.) Thus, the solution to problem (1) is precisely the vector we seek in solving the linear system $Ax^* = b$.

1 The Steepest Descent Algorithm

Recall from calculus that the gradient, $\nabla f(x)$, is the direction in which the function f is most rapidly increasing, and $-\nabla f(x)$ is the direction of steepest descent. Thus, if we want to minimize f, we might think of taking a guess at x^*, evaluating the gradient, and taking a step in the opposite direction until the function stops decreasing. Then we can repeat the process. This gives the following algorithm.

1. Pick x_0 .
2. For $k = 0, 1, \ldots,$
   (a) Evaluate $p_k = -\nabla f(x_k) = r_k$.

(b) Let $x_{k+1} = x_k + \alpha_k p_k$, where α_k is the minimizer of $\min_\alpha f(x_k + \alpha p_k)$.

End For.

To visualize the algorithm, picture an elliptical valley surrounded by mountains. Level surfaces of the terrain are shown in Figure 1, as they might appear on a topographical map. If a person is at point x_0 in the fog and wants to reach the pit of the valley, she might follow an algorithm of picking the direction of steepest descent, following the straight path until it starts to rise, and then picking the new steepest descent direction. In that case, she follows the zigzag path indicated in the figure. (See how relevant numerical analysis can be in real life?)

We can find an analytic formula for α_k. For fixed x_k and p_k,

$$\begin{aligned} f(x_k + \alpha p_k) &= \frac{1}{2}(x_k + \alpha p_k)^T A (x_k + \alpha p_k) - (x_k + \alpha p_k)^T b \\ &= \frac{1}{2}\alpha^2 p_k^T A p_k + \alpha p_k^T A x_k + -\alpha p_k^T b + \text{constant}\,. \end{aligned}$$

The minimum of f with respect to α occurs when the derivative is zero:

$$p_k^T A x_k + \alpha p_k^T A p_k - p_k^T b = 0 \tag{2}$$

so

$$\alpha = -\frac{p_k^T (A x_k - b)}{p_k^T A p_k} = \frac{p_k^T r_k}{p_k^T A p_k} \tag{3}$$

So, to perform the minimization along a line, we set

$$\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k} = \frac{r_k^T r_k}{p_k^T A p_k}$$

(See the appendix for the proof of equivalence of the two expressions for α.)

Let

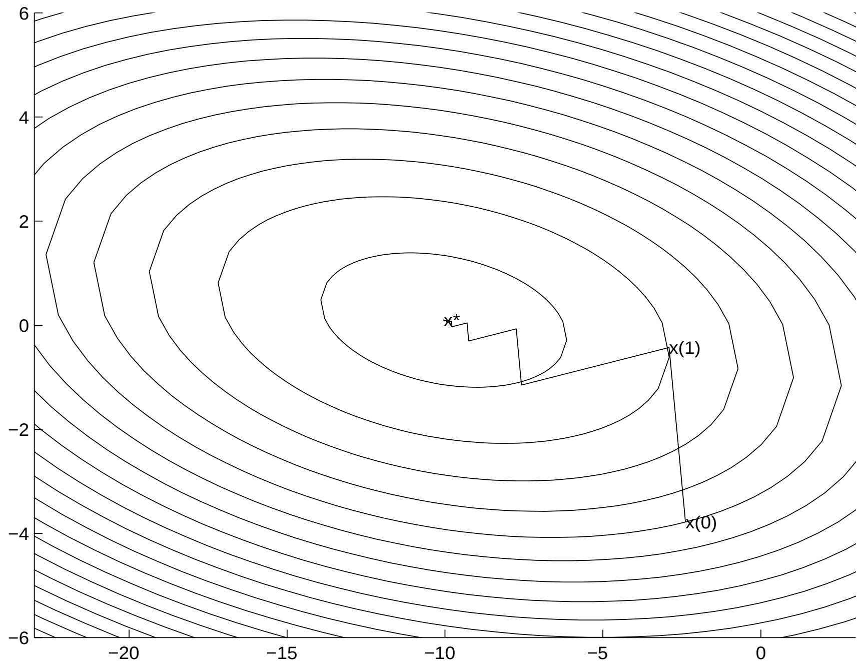
$$E(x) = \frac{1}{2}(x - x^*)^T A (x - x^*)\,.$$

This function also is minimized when $x = x^*$, and it is a convenient way to measure error. It can be shown that the steepest descent algorithm has the following convergence rate:

$$E(x_k) \le \left(\frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}}\right)^{2k} E(x_0)\,,$$

where λ_{max} and λ_{min} are the largest and smallest eigenvalues of A. (Try to interpret this result in terms of the condition number of A in the 2-norm, the ratio of the largest to smallest eigenvalue. Which matrices will show fast convergence?)

Figure 1: Level curves (contour plot) for a quadratic function of two variables, with the path of the steepest descent algorithm marked on it. After 20 iterations, the error has been reduced by a factor of 10^{-5}. Conjugate gradients would step from the initial iterate to the next, and then to the minimizer.



2 The Conjugate Direction Algorithm

As we can see, the steepest descent algorithm is often far too slow. We will now develop an algorithm that only takes n steps. It is based on a very simple idea. Suppose we had n linearly independent vectors p_k, $k = 0, 1, \ldots, n-1$, with the property

$$p_k^T A p_j = 0\,, k \neq j\,.$$

(If $A = I$, this is just "orthogonality." For a general symmetric A, it is called "A-conjugacy.") Since there are n vectors, and they are linearly independent, they form a basis, and we can express any vector as a linear combination of them; for example,

$$x^* - x_0 = \sum_{j=0}^{n-1} \alpha_j p_j\,.$$

Let's multiply each side of this equation by $p_k^T A$ for each k. On the left hand side we have

$$p_k^T A(x^* - x_0) = p_k^T (b - Ax_0) = p_k^T r_0\,,$$

and on the right we have

$$p_k^T A \sum_{j=0}^{n-1} \alpha_j p_j = \alpha_k p_k^T A p_k\,.$$

Therefore,

$$p_k^T r_0 = \alpha_k p_k^T A p_k$$

and

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}\,.$$

So we have a new algorithm for solving $Ax^* = b$:

1. Pick x_0 and A-conjugate directions p_k, $k = 0, 1, \ldots, n-1$.
2. For $k = 0, 1, \ldots, n-1$
    (a) Set
    $$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}\,.$$
    (b) Let $x_{k+1} = x_k + \alpha_k p_k$.

    End For.

Then $x_n = x^*$. It would not be hard to convince yourself that, because of conjugacy,

$$p_k^T r_0 = p_k^T r_k$$

and thus the formula for α_k is exactly equivalent to (3), although the directions p_k are chosen differently.

It is easy to construct a set of A-conjugate vectors. Just begin with any linearly independent set v_k, $k = 0, 1, \ldots, n-1$, and perform a Gram-Schmidt process:

1. Let $p_0 = v_0$.
2. For $k = 0, 1, \ldots, n-2$

$$p_{k+1} = v_{k+1} - \sum_{j=0}^{k} \frac{p_j^T A v_{k+1}}{p_j^T A p_j} p_j$$

   End For.

It is more numerically stable to implement this last equation iteratively, substituting p_{k+1} for v_{k+1} after $j = 0$ (Modified Gram-Schmidt algorithm):

1. Let $p_{k+1} = v_{k+1}$.
2. For $j = 0, 1, \ldots, k$,

$$p_{k+1} = p_{k+1} - \frac{p_j^T A p_{k+1}}{p_j^T A p_j} p_j$$

   End For.

3 The Conjugate Gradient Algorithm

The conjugate gradient algorithm is a special case of the conjugate direction algorithm. In this case, we intertwine the calculation of the new x vector and the new p vector. In fact, the set of linearly independent vectors v_k we use in the Gram-Schmidt process is just the set of residuals r_k. The algorithm is as follows:

1. Let x_0 be an initial guess.
   Let $r_0 = b - Ax_0$ and $p_0 = r_0$.
2. For $k = 0, 1, 2, \ldots$, until convergence,
   (a) Compute the search parameter α_k and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k} \, , \\ x_{k+1} &= x_k + \alpha_k p_k \, , \\ r_{k+1} &= r_k - \alpha_k A p_k \, , \end{aligned}$$

   (b) Compute the new search direction p_{k+1} by Gram-Schmidt on r_{k+1} and the previous p vectors to make p_{k+1} A-conjugate to the previous directions.

End For.

Note that the first step is a steepest descent step, and that in Figure 1, the sequence of points is x_0, x_1, and x^*.

In this form, the algorithm is a lengthy process, particularly the Gram-Schmidt phase. We can shortcut in two places, though. In the current form we need two matrix multiplications per iteration: Ap_k for α_k and Ax_{k+1} for r_{k+1}. But note that

$$r_{k+1} = b - Ax_{k+1} = b - A(x_k + \alpha_k p_k) = r_k - \alpha_k A p_k$$

so we actually need only one matrix multiplication.

The second shortcut is really surprising. It turns out that

$$p_j^T A r_{k+1} = 0, \;\; j < k\,,$$

so the Gram-Schmidt formula (with v_{k+1} replaced by r_{k+1}) reduces to

$$p_{k+1} = r_{k+1} - \frac{p_k^T A r_{k+1}}{p_k^T A p_k} p_k$$

which is very little work!

So here is the practical form of the conjugate gradient algorithm.

1. Let x_0 be an initial guess.
   Let $r_0 = b - Ax_0$ and $p_0 = r_0$.
2. For $k = 0, 1, 2, \ldots$, until convergence,
   (a) Compute the search parameter α_k and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{p_k^T r_k}{p_k^T A p_k}\,, (\text{or, equivalently}, \frac{r_k^T r_k}{p_k^T A p_k}) \\ x_{k+1} &= x_k + \alpha_k p_k\,, \\ r_{k+1} &= r_k - \alpha_k A p_k\,, \end{aligned}$$

   (b) Compute the new search direction

$$\begin{aligned} \beta_k &= -\frac{p_k^T A r_{k+1}}{p_k^T A p_k}\,, (\text{or, equivalently}, \frac{{r_{k+1}}^T r_{k+1}}{r_k^T r_k})\,, \\ p_{k+1} &= r_{k+1} + \beta_k p_k\,, \end{aligned}$$

   End For.

And after $K \leq n$ steps, the algorithm terminates with $r_K = 0$ and $x_K = x^*$. The number K is bounded above by the number of distinct eigenvalues of A.

Not only does this algorithm terminate in a finite number of steps, a definite advantage over steepest descent, but its error *on each step* has a better bound:

$$E(x_k) \leq \left(\frac{1 - \sqrt{\kappa^{-1}}}{1 + \sqrt{\kappa^{-1}}} \right)^{2k} E(x_0) \,,$$

where $\kappa = \lambda_{max}/\lambda_{min}$. So, even as an iterative method, without running a full K steps, conjugate gradients converges faster.

4 Preconditioned Conjugate Gradients

Consider the problem

$$M^{-1/2} A M^{-1/2} \bar{x} = M^{-1/2} b \,,$$

where M is a symmetric positive definite. Then $x = M^{-1/2}\bar{x}$ solves our original problem $Ax^* = b$. Applying conjugate gradients to this problem yields

1. Let $\bar{x}_0$ be an initial guess.
   Let $\bar{r}_0 = M^{-1/2}b - M^{-1/2}AM^{-1/2}\bar{x}_0$ and $\bar{p}_0 = \bar{r}_0$.
2. For $k = 0, 1, 2, \ldots,$ until convergence,
   (a) Compute the search parameter α_k and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{\bar{r}_k^T \bar{r}_k}{\bar{p}_k^T M^{-1/2} A M^{-1/2} \bar{p}_k} \\ \bar{x}_{k+1} &= \bar{x}_k + \alpha_k \bar{p}_k \,, \\ \bar{r}_{k+1} &= \bar{r}_k - \alpha_k M^{-1/2} A M^{-1/2} \bar{p}_k \,, \end{aligned}$$

   (b) Compute the new search direction

$$\begin{aligned} \beta_k &= \frac{{\bar{r}_{k+1}}^T \bar{r}_{k+1}}{\bar{r}_k^T \bar{r}_k} \,, \\ \bar{p}_{k+1} &= \bar{r}_{k+1} + \beta_k \bar{p}_k \,, \end{aligned}$$

   End For.

Now let's return to the original coordinate system. Let $M^{-1/2}r = \bar{r}$, $x = M^{-1/2}\bar{x}$, and $p = M^{-1/2}\bar{p}$. Then the algorithm becomes

1. Let x_0 be an initial guess.
   Let $r_0 = b - Ax_0$ and $p_0 = M^{-1}r_0$.

2. For $k = 0, 1, 2, \ldots$, until convergence,

   (a) Compute the search parameter α_k and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{r_k^T M^{-1} r_k}{p_k^T A p_k} \\ x_{k+1} &= x_k + \alpha_k p_k \,, \\ r_{k+1} &= r_k - \alpha_k A p_k \,, \end{aligned}$$

   (b) Compute the new search direction

$$\begin{aligned} \beta_k &= \frac{{r_{k+1}}^T M^{-1} r_{k+1}}{r_k^T M^{-1} r_k} \,, \\ p_{k+1} &= M^{-1} r_{k+1} + \beta_k p_k \,, \end{aligned}$$

   End For.

We choose the symmetric positive definite matrix M so that $M^{-1/2} A M^{-1/2}$ has better eigenvalue properties, and so that it is easy to apply the operator M^{-1}.

- For fast iterations, we want to be able to apply M^{-1} very quickly.
- To make the number of iterations small, we want M^{-1} to be an approximate inverse of A.

Some common choices of the preconditioning matrix M:

- M = the diagonal of A.
- M = a banded piece of A.
- M = an incomplete factorization of A, leaving out inconvenient elements.
- M = a related matrix; e.g., if A is a discretization of a differential operator, M might be a discretization of a related operator that is easier to solve.
- M might be the matrix from our favorite stationary iterative method (SIM).

That last choice could use a little explanation. Consider your favorite stationary iterative method (Jacobi, Gauss-Seidel, SOR, etc.) It can be derived by taking the equation $Ax = b$, splitting A into two pieces $A = M - N$, and writing $Mx = Nx + b$. The iteration then becomes

$$M x_{k+1} = N x_k + b$$

or

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b.$$

Manipulating this a bit, we get

$$\begin{aligned} x_{k+1} &= x_k + (M^{-1}N - I)x_k + M^{-1}b \\ &= x_k + M^{-1}(N - M)x_k + M^{-1}b \\ &= x_k + M^{-1}(b - Ax_k) \\ &= x_k + M^{-1}r_k\,. \end{aligned}$$

The matrix M that determines the multiple of the residual that we add on to x becomes the conjugate gradient preconditioner.

5 Appendix: Algebra of Conjugate Gradients

In this appendix, we establish the Krylov subspace property of conjugate gradients. and the equivalence of the alternate formulas for α and β.

Let $p_0 = r_0 = b - Ax_0$. Then we have already established the following four relations:

$$r_{k+1} = r_k - \alpha_k A p_k\,, \tag{4}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k\,, \tag{5}$$

$$\alpha_k = \frac{r_k^T p_k}{p_k^T A p_k}\,, \tag{6}$$

$$\beta_k = -\frac{{r_{k+1}}^T A p_k}{p_k^T A p_k}\,. \tag{7}$$

In this appendix we establish nine more.

The next two relations lead us to the alternate formula for α. First,

$$p_k^T r_{k+1} = 0 \tag{8}$$

since

$$\begin{aligned} p_k^T r_{k+1} &= p_k^T r_k - \alpha_k p_k^T A p_k && \text{by (4)} \\ &= 0 && \text{by (6)}\,. \end{aligned}$$

Next,

$$r_k^T r_k = r_k^T p_k \tag{9}$$

since it is true for $i = 0$, and if we assume it true for i then

$$\begin{aligned} {r_{k+1}}^T p_{k+1} &= {r_{k+1}}^T r_{k+1} + \beta_k {r_{k+1}}^T p_k && \text{by (5)} \\ &= {r_{k+1}}^T r_{k+1} && \text{by (8)}\,. \end{aligned}$$

Therefore,

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}\,.$$

Now we aim for the alternate formula for β. We have that

$$p_{k+1}{}^T A p_k = 0 \tag{10}$$

since

$$\begin{aligned} p_{k+1}{}^T A p_k &= r_{k+1}{}^T A p_k + \beta_k p_k^T A p_k && \text{by (5)} \\ &= 0 && \text{by (7)}\,. \end{aligned}$$

The next two relations

$$\begin{aligned} r_k^T p_j &= 0\,, \quad k > j\,, && (11) \\ p_k^T A p_j &= 0\,, \quad k \neq j\,, && (12) \end{aligned}$$

are established together. For $k, j = 0, 1$, they are true by (8) and (10). Assume that they are true for indices less than or equal to k. Then by (4),

$$r_{k+1}{}^T p_j = r_k^T p_j - \alpha_k p_k^T A p_j = 0\,, \tag{13}$$

where the last equality follows from the induction hypothesis if $j < k$ and from (8) if $j = k$. Therefore,

$$\begin{aligned} p_{k+1}{}^T A p_j &= r_{k+1}{}^T A p_j + \beta_k p_k^T A p_j && \text{by (5)} \\ &= r_{k+1}{}^T \tfrac{-r_{j+1}+r_j}{\alpha_j} + \beta_k p_k^T A p_j && \text{by (4)} \\ &= r_{k+1}{}^T \tfrac{\beta_j p_j - p_{j+1} + p_j - \beta_{j-1} p_{j-1}}{\alpha_j} \\ &\quad + \beta_k p_k^T A p_j && \text{by (5)} \\ &= 0 && \text{if } j < k \text{ by (13) and the induction hypothesis} \\ &= 0 && \text{if } j = k \text{ by (10).} \end{aligned}$$

The next relation that we need is

$$r_k^T r_j = 0, \quad k \neq j \tag{14}$$

We can assume that $k > j$. Now, if $j = 0$, $r_k^T r_j = r_k^T p_0 = 0$ by (11). If $j > 0$, then

$$\begin{aligned} r_k^T r_j &= r_k^T p_j - \beta_{j-1} r_k^T p_{j-1} && \text{by (5)} \\ &= 0 && \text{by (11)}\,, \end{aligned}$$

and this establishes (14). Now we work with β:

$$\begin{aligned}
\beta_k &= -\tfrac{{r_{k+1}}^T A p_k}{p_k^T A p_k} && \text{by (7)} \\
&= -\tfrac{{r_{k+1}}^T (r_k - r_{k+1})}{\alpha_k p_k^T A p_k} && \text{by (4)} \\
&= -\tfrac{{r_{k+1}}^T (r_k - r_{k+1})}{r_k^T p_k} && \text{by (6)} \\
&= +\tfrac{{r_{k+1}}^T r_{k+1}}{r_k^T p_k} && \text{by (14)}
\end{aligned}$$

Therefore, by (9),

$$\beta_k = \frac{{r_{k+1}}^T r_{k+1}}{r_k^T r_k}\,. \tag{15}$$

Finally, we note that that if sp denotes the subspace spanned by a set of vectors, then

$$sp\{p_0, p_1, \ldots, p_k\} = sp\{r_0, Ar_0, \ldots, A^{k-1} r_0\} = sp\{r_0, r_1, \ldots, r_k\} \tag{16}$$

since $p_{k+1} \in sp\{r_{k+1}, p_k\}$ by (5) and $r_{k+1} \in sp\{r_k, Ap_k\}$ by (4). This shows that conjugate gradients is a *Krylov subspace method.* In fact, it is characterized by minimizing $E(x)$ over all vectors with $x - x_0 \in sp\{r_0, Ar_0, \ldots, A^{k-1} r_0\}$.

6 References

The original paper on conjugate gradients:
M. R. Hestenes and E. Stiefel, "Methods of Conjugate Gradients for Solving Linear Systems," *J. Res. Natl. Bur. Standards* 49 (1952) pp. 409-436.

A clear exposition of the algorithm (without preconditioning):
David G. Luenberger, *Linear and Nonlinear Programming*, Addison Wesley, 2nd edition (1984).

These notes parallel Luenberger's development in many ways.