THE TRI-FACED CONJUGATE GRADIENT METHOD

David F. Gleich

November 12, 2019

The conjugate gradients (CG) method is one of the most celebrated algorithms for solving $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ when $\boldsymbol{A}$ is large, sparse, and *symmetric positive definite*. It is also a sly method in the sense that there are *three derivations* of the CG method. Each starts from a different point, but gives rise to the same sequence of iterates. They are:

1. the Lanczos process – e.g. via matrix approximation
2. the steepest descent method – i.e. via optimization
3. the three-term recurrents – i.e. via orthogonal polynomials

The derivation for the Lanzcos process is, perhaps, the best as it provides a straightforward path to solve symmetric indefinite systems as well.

Throughout these notes, let $\boldsymbol{A}$ be $n \times n$, symmetric positive definite.

1 CONJUGATE GRADIENTS VIA THE LANCZOS PROCESS

1.1 THE LANCZOS PROCESS

Because $\boldsymbol{A}$ is symmetric, we can run the Lanczos process to iteratively compute a tridiagonal matrix $\boldsymbol{T}$ that approximates the matrix $\boldsymbol{A}$. *For linear systems, we also begin the Lanczos process with the vector* $\mathbf{b}/\|\mathbf{b}\|$. After k steps, we have:

$$\underbrace{\boldsymbol{A}}_{n\times n}\underbrace{\boldsymbol{V}}_{n\times k} = \underbrace{\boldsymbol{V}_{k+1}}_{n\times k+1}\underbrace{\boldsymbol{T}_{k+1}}_{k+1\times k}$$

where $\boldsymbol{V} = \begin{bmatrix}\mathbf{v}_1, \ldots, \mathbf{v}_k\end{bmatrix}$ holds the first k vectors in the Lanczos process and $\boldsymbol{V}_{k+1}$ holds the first k vectors *and the $k+1$st vector*. The matrix

$$\boldsymbol{T}_{k+1} = \begin{bmatrix}\alpha_1 & \beta_2 & & \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_k \\ & & \beta_k & \alpha_k \\ & & & \beta_{k+1}\end{bmatrix} = \begin{bmatrix}\bar{\boldsymbol{T}} \\ \beta_{k+1}\mathbf{e}_k^T\end{bmatrix}$$

where $\bar{\boldsymbol{T}}$ is the $k \times k$ tridiagonal matrix from the first k rows. When it's important, we'll write:

$$\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_{k+1}\boldsymbol{T}_{k+1}$$

to denote the full sequence of matrices. Likewise, the Lanczos process gives rise to a sequence of tridiagonal matrices:

$$\bar{\boldsymbol{T}}_1, \bar{\boldsymbol{T}}_2, \ldots, \bar{\boldsymbol{T}}_k, \bar{\boldsymbol{T}}_{k+1}, \ldots$$

However, we'll often drop the index k when it applies to *any index*. For instance, in exact arithmetic, $\boldsymbol{V}^T\boldsymbol{A}\boldsymbol{V} = \bar{\boldsymbol{T}}$.

Note The vector $\mathbf{v}_1 = \mathbf{b}/\|\mathbf{b}\|$, and also $\boldsymbol{V}\mathbf{e}_1 = \mathbf{v}_1$ for all k.

Quiz Show that $\bar{\boldsymbol{T}}$ is positive definite if $\boldsymbol{A}$ is positive definite.

1.2 LANCZOS AND LINEAR SYSTEMS

Let $\mathbf{y} = \boldsymbol{V}\mathbf{z}$ be a vector in the span of the Lanzcos vectors after k steps. Recall that we showed this means that $\mathbf{y}$ is a member of the kth Krylov subspace

$$\mathbf{y} \in \mathcal{K}_k(\boldsymbol{A}, \mathbf{b}).$$

For any vector $\mathbf{y} = \boldsymbol{V}\mathbf{z}$:

$$\|\mathbf{b} - \boldsymbol{A}\mathbf{y}\| = \|\mathbf{b} - \boldsymbol{A}\boldsymbol{V}\mathbf{z}\| = \|\boldsymbol{V}_{k+1}\left(\|\mathbf{b}\|\mathbf{e}_1 - \boldsymbol{T}_{k+1}\mathbf{z}\right)\| = \|\|\mathbf{b}\|\mathbf{e}_1 - \boldsymbol{T}_{k+1}\mathbf{z}\|.$$

Thus, we want to pick $\mathbf{z}$ such that $\|\mathbf{b}\|\mathbf{e}_1 - \boldsymbol{T}_{k+1}\mathbf{z}$ is small at each step.

In the conjugate gradients method, we choose $\mathbf{z}$ such that

$$\bar{\boldsymbol{T}}\mathbf{z} = \|\mathbf{b}\|\mathbf{e}_1$$

so that

$$\|\|\mathbf{b}\|\mathbf{e}_1 - \boldsymbol{T}_{k+1}\mathbf{z}\| = |\beta_{k+1}z_k|.$$

In contrast, in the MINRES method, we choose $\mathbf{z}$ to minimize $\|\|\mathbf{b}\|\mathbf{e}_1 - \boldsymbol{T}_{k+1}\mathbf{z}\|$ at each step; and in the SYMMLQ method, we choose $\mathbf{y} = \boldsymbol{V}_{k+1}\mathbf{z}$ and $\mathbf{z}$ is the minimum norm solution of $\boldsymbol{T}_{k+1}^T\mathbf{z}$. We won't spend too much time studying these methods.

1.3 THE SIMPLE CG METHOD

Consequently, and conceptually, the CG method is rather simple:

```
for k=1, 2, ...
  Compute $\boldsymbol{V}_k, \boldsymbol{T}_k$ from $k$-steps of the Lanczos process
  Solve $\bar{\boldsymbol{T}}_k\mathbf{z}_k = \|\mathbf{b}\|\mathbf{e}_1$
  Compute $\mathbf{x}_k = \boldsymbol{V}_k\mathbf{z}_k$
  If $|\beta_{k+1}\mathbf{z}_k| <$ tol, stop.
```

The essence of the method is that we replace solving $\boldsymbol{A}\mathbf{x} = \mathbf{b}$ with solving $\bar{\boldsymbol{T}}_k\mathbf{z} = \|\mathbf{b}\|\mathbf{e}_1$. Put another way, the idea is that the matrix $\bar{\boldsymbol{T}}_k$ "approximates" $\boldsymbol{A}$.

The difficulty with the simple method However, at each step, there is still quite a bit of work in this method. We can efficiently compute the kth step of the Lanczos vector sequence from the $k-1$st step, so computing $\boldsymbol{V}_k$ and $\boldsymbol{T}_k$ isn't a problem using one matrix vector and a few inner-products, so it's $O(\texttt{mat-vec} + n)$ work where $O(\texttt{mat-vec})$ is the work involved in the matrix-vector product. To solve the system with $\bar{\boldsymbol{T}}$ is $O(k)$ work because it's a tridiagonal system. However, the problem is that computing $\mathbf{x}_k = \boldsymbol{V}_k\mathbf{z}$ is $O(nk)$ work. If n is large and k is small, then this operation is expensive. Another problem is that we need to keep k Lanczos vectors. This gets *very* expensive in terms of memory for large k.

1.4 MAKING CG EFFICIENT

We'll now see how to do the CG method with $O(\texttt{mat-vec} + n)$ work per iterations. To do so, we need to determine how to compute $\mathbf{x}_k$ directly from $\mathbf{x}_{k-1}$ and avoid storing $\boldsymbol{V}_k$. We'll have to keep the last two iterates though, so we can continue the Lanczos process.

In the following discussion, we'll work through how to make this happen. There is one leap in this derivation that we'll get to soon.

Using and updating Cholesky for the subsystem We begin with a straightforward computation. Think about how to compute $\mathbf{z}_k$ efficiently. Recall that $\boldsymbol{A}$ is symmetric positive definite. Based on the quiz above, this means that $\bar{\boldsymbol{T}}$ is also symmetric, positive definite. So it has a Cholesky factorization

$$\bar{\boldsymbol{T}}_k = \boldsymbol{F}_k\boldsymbol{F}_k^T.$$

On the last homework, we worked out that:

$$\boldsymbol{F} = \begin{bmatrix} \eta_1 & & & \\ \mu_2 & \eta_2 & & \\ & \ddots & \ddots & \\ & & \mu_k & \eta_k \end{bmatrix}$$

Moreover, we can compute μ_{k+1} and η_{k+1} from

$$\bar{\boldsymbol{T}}_{k+1} = \begin{bmatrix} \bar{\boldsymbol{T}}_k & \beta_{k+1}\mathbf{e}_k \\ \beta_{k+1}\mathbf{e}_k^T & \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{F}_k & \\ \mu_{k+1}\mathbf{e}_k^T & \eta_{k+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_k^T & \mu_{k+1}\mathbf{e}_k \\ & \eta_{k+1} \end{bmatrix}.$$

By equating terms, we find that

$$\beta_{k+1}\mathbf{e}_k = \mu_{k+1}\boldsymbol{F}_k\mathbf{e}_k = \mu_{k+1}\eta_k\mathbf{e}_k$$

and

$$\alpha_{k+1} = \eta_{k+1}^2 + \mu_{k+1}^2.$$

We can solve both of these to find:

$$\mu_{k+1} = \beta_{k+1}/\eta_k \text{ and } \eta_{k+1} = \sqrt{\alpha_{k+1} - \mu_{k+1}^2}.$$

But, there is no way to compute $\mathbf{z}_{k+1}$ from $\mathbf{z}_k$ because *all the elements* change. To go beyond this, we need to look at the problem more closely.

The leap What we actually want is

$$\mathbf{x}_k = \|\mathbf{b}\| \boldsymbol{V}_k \bar{\boldsymbol{T}}^{-1}\mathbf{e}_1.$$

Let's substitute the Cholesky fatorization in here:

$$\mathbf{x}_k = \|\mathbf{b}\| \boldsymbol{V}_k \boldsymbol{F}^{-T}\boldsymbol{F}^{-1}\mathbf{e}_1.$$

The "leap" is that we need to look at:

$$\mathbf{x}_k = \|\mathbf{b}\| \cdot \underbrace{\boldsymbol{C}_k}_{=\boldsymbol{V}_k\boldsymbol{F}^{-T}} \cdot \underbrace{\mathbf{p}_k}_{=\boldsymbol{F}^{-1}\mathbf{e}_1}.$$

As we study these expressions, we'll find that they can be updated efficiently.

First, let's tackle $\mathbf{p}_k$. Given $\mathbf{p}_k = \begin{bmatrix} p_1, \ldots, p_k \end{bmatrix}^T$, note that:

$$\boldsymbol{F}_{k+1}\mathbf{p}_{k+1} = \begin{bmatrix} \boldsymbol{F} & \\ \mu_{k+1}\mathbf{e}_k^T & \eta_{k+1} \end{bmatrix} \begin{bmatrix} \mathbf{p}_k \\ p_{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{e}_1 \\ 0 \end{bmatrix}.$$

Thus, $p_{k+1} = -\mu_{k+1}p_k/\eta_{k+1}$. This is great news because $\mathbf{p}_{k+1}$ only differs from $\mathbf{p}_k$ by the last element.

Let's see how to find $\boldsymbol{C}_{k+1}$ from $\boldsymbol{C}_k$. We'll write:

$$\boldsymbol{C}_{k+1}\boldsymbol{F}_{k+1}^T = \boldsymbol{V}_{k+1}$$

or

$$\begin{bmatrix} \boldsymbol{C}_k & \mathbf{c}_{k+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_k^T & \mu_{k+1}\mathbf{e}_k \\ & \eta_{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{V}_k & \mathbf{v}_{k+1} \end{bmatrix}.$$

Hence,

$$\mu_{k+1}\boldsymbol{C}_k\mathbf{e}_k + \mathbf{c}_{k+1}\eta_{k+1} = \mu_{k+1}\mathbf{c}_k + \eta_{k+1}\mathbf{c}_{k+1} = \mathbf{v}_{k+1}.$$

Thus, we can compute $\mathbf{c}_{k+1}$ just from $\mathbf{c}_k$.

The last step is to show that we can combine these and compute $\mathbf{x}_{k+1}$ from $\mathbf{x}_k$. Again, we expand:

$$\mathbf{x}_{k+1} = \boldsymbol{C}_{k+1}\mathbf{p}_{k+1} = \underbrace{\boldsymbol{C}_k\mathbf{p}_k}_{=\mathbf{x}_k} + \mathbf{c}_{k+1}p_{k+1}.$$

And we have found an efficient expression for $\mathbf{x}_k$ in the CG method.

All together now, we have:

```
$\beta_1 = \|\mathbf{b}\|$
$\mathbf{v}_0 = 0$, $\mathbf{x}_0 = 0$
$\mathbf{v}_1 = \mathbf{b}/\beta_1$
for i=1, 2, ...
    $\mathbf{w} = \mathbf{A}\mathbf{v}_i - \beta_i \mathbf{v}_{i-1}$
    $\alpha_i = \mathbf{v}_i^T \mathbf{w}$
    $\mathbf{w} \leftarrow \mathbf{w} - \alpha_i \mathbf{v}_i$
    $\beta_{i+1} = \|\mathbf{w}\|$
    $\mathbf{v}_{i+1} = \mathbf{w}/\beta_{i+1}$
    if $i = 1$
        $\mu_i = 0$, $\eta_i = \sqrt{\alpha_i}$, $p_i = \beta_1/\eta_i$, $\mathbf{c}_i = \mathbf{v}_1/\eta_1$.
    else
        $\mu_i = \beta_i/\eta_{i-1}$, $\eta_i = \sqrt{\alpha_i - \mu_i^2}$, $p_i = -\mu_i p_{i-1}/\eta_i$, $\mathbf{c}_i = (\mathbf{v}_i - \mu_i \mathbf{c}_{i-1})/\eta_i$
    $\mathbf{x}_i = \mathbf{x}_{i-1} + p_i \mathbf{c}_i$
```

In this iteration, you only need to keep the vector $\mathbf{v}_i$ and $\mathbf{c}_i$ to complete the iteration.

2 CONJUGATE GRADIENTS VIA OPTIMIZATION

This section is incomplete

In the second derivation of the CG method, we study the problem:

$$\min \phi(x) \text{ where } \phi(x) = \frac{1}{2}\mathbf{x}^T \mathbf{A}\mathbf{x} - \mathbf{x}^T \mathbf{b}$$

where $\mathbf{A}$ is $n \times n$, symmetric positive definite. In this case, the solution is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$, which we can derive by setting the gradient of $\phi(x)$ to zero,[1] that is,

[1] this condition suffices because the problem is strongly convex

$$\partial\phi/\partial\mathbf{x} = \mathbf{A}\mathbf{x} - \mathbf{b} = 0.$$

Thus, in the second derivation of the CG method we work from the premise of finding a sequence of vectors $\mathbf{x}_k$ that make $\phi(\mathbf{x}_k)$ smaller at each step.

Aside on steepest descent One of the classic ways to minimize a function is called gradient descent, and it computes

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{g}_k$$

where $\alpha_k > 0$ and $\mathbf{g}_k$ is the gradient $\partial\phi/\partial\mathbf{x}$ evaluated at $\mathbf{x}^{(k)}$. For this function ϕ, $\mathbf{g}_k = \mathbf{A}\mathbf{x}_k - \mathbf{b}$. The constant α_k is chosen to make:

$$\phi(\mathbf{x}_k - \alpha \mathbf{g}_k)$$

as *small as possible*. It's a bit of a tangent to derive this value, but we can work out the solution, which is:[2]

[2] check the sign on this value

$$\alpha_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_k^T \mathbf{A}\mathbf{g}_k}.$$

This simple method:

```
$\mathbf{x}^{(1)} = 0$
for k=1, ...
  $\mathbf{g}^{(k)} = \mathbf{A}\mathbf{x}^{(k)} - \mathbf{b}$
  if $\|\mathbf{g}^{(k)}\|$ < tol
    stop and return $\mathbf{x}_k$
  $\alpha^{(k)} = \mathbf{g}^{(k)T}\mathbf{g}^{(k)}/(\mathbf{g}^{(k)T}\mathbf{A}\mathbf{g}^{(k)})$
  $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{g}^{(k)}$
```

will always converge to the solution of a positive definite system using only matrix-vector products. The convergence rate is proportional to the condition number of the matrix.

Now, suppose we consider a sequence of directions $\mathbf{p}^{(k)}$ such that $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha_k \mathbf{p}^{(k)}$, we can set:

$$\alpha_k = \mathbf{p}^{(k)T}\mathbf{r}^{(k)}/(\mathbf{p}^{(k)T}\mathbf{A}\mathbf{p}^{(k)})$$

as long as $\mathbf{p}^{(k)}\mathbf{r}^{(k)} \neq 0$ where $\mathbf{r}^{(k)}$ is the kth residual $\mathbf{b} - \mathbf{A}\mathbf{x}^{(k)}$.

To get to conjugate gradients, we want the set of search directions $\mathbf{p}^{(k)}$ to be linearly independent, and in fact, conjugate. We call a sequence of vectors conjugate if $\mathbf{p}^{(i)T}\mathbf{A}\mathbf{p}^{(j)} = 0$ if $i \neq j$. In this case, $\mathbf{x}^{(k)} = \sum_{i=1}^{n} \alpha_i \mathbf{p}^{(i)}$ or $\mathbf{x}^{(k)} \in \text{span}\{\mathbf{p}^{(1)}, \ldots, \mathbf{p}^{(k)}\}$.

3 CG HISTORY

Information from Diane O'Leary, https://www.siam.org/meetings/la09/talks/oleary.pdf and Numerical Analysis: Historical Developments in the 20th Century; By C. Brezinski, L. Wuytack; Gene H. Golub and Dianne P. O'Leary, "Some history of the conjugate gradient and Lanczos algorithms: 1948-1976," SIAM Review 31 (1989) 50-102. http://www.cs.umd.edu/~oleary/reprints/j28.pdf

1. Initially proposed by Hestenes and Stiefel as a direct method (1952). Both Hestenes and Stiefel came up with the method independently, and then wrote a joint paper about it.
2. First suggested as a large sparse solver by Reid (1971).
3. Finally widely accepted for matrices once preconditioning was invented.