AMSC 600 /CMSC 760 Advanced Linear Numerical Analysis

Fall 2007

Bi-orthogonalization Methods

Dianne P. O'Leary

Algorithms that use the Bi-orthogonal Basis

- Lanczos Bi-orthogonalization
- Transpose-free variants

Reference: Chapter 7 of Saad

Lanczos Bi-orthogonalization

Up to now: Arnoldi Basis

$$\mathbf{AV} = \mathbf{VH}.$$

Lanczos Bi-orthogonalization (Saad Section 7.1)

$$\begin{aligned} \mathbf{AV} &= \mathbf{VT}, \\ \mathbf{A}^T\mathbf{W} &= \mathbf{WT}^T, \end{aligned}$$

with $\mathbf{V}^T\mathbf{W} = \mathbf{I}$ and $\mathbf{T}$ tridiagonal.

Note: If $\mathbf{A}$ is symmetric and $\mathbf{v}_1 = \mathbf{w}_1$, then $\mathbf{V} = \mathbf{W}$ and we have the Lanczos algorithm that we studied previously.

Let's work through the relations. As usual, we look at one particular column, say column j.

$$\begin{aligned} \mathbf{A}\mathbf{v}_j &= \beta_j\mathbf{v}_{j-1} + \alpha_j\mathbf{v}_j + \delta_{j+1}\mathbf{v}_{j+1} \\ \mathbf{A}^T\mathbf{w}_j &= \delta_j\mathbf{w}_{j-1} + \alpha_j\mathbf{w}_j + \beta_{j+1}\mathbf{w}_{j+1} \end{aligned}$$

where $\beta_j = t_{j-1,j}$, $\alpha_j = t_{j,j}$, and $\delta_{j+1} = t_{j+1,j}$.

Let's enforce biorthogonality.

- If $\mathbf{w}_j$ is orthogonal to all of the vectors $\mathbf{v}_i$ except when $i = j$, then

$$\mathbf{w}_j^T\mathbf{A}\mathbf{v}_j = \alpha_j\mathbf{w}_j^T\mathbf{v}_j,$$

so

$$\alpha_j = \mathbf{w}_j^T\mathbf{A}\mathbf{v}_j.$$

- Similarly, letting

$$\begin{aligned}\delta_{j+1}\mathbf{v}_{j+1} \equiv \hat{\mathbf{v}}_{j+1} &= (\mathbf{A}-\alpha_j\mathbf{I})\mathbf{v}_j - \beta_j\mathbf{v}_{j-1},\\ \beta_{j+1}\mathbf{w}_{j+1} \equiv \hat{\mathbf{w}}_{j+1} &= (\mathbf{A}^T-\alpha_j\mathbf{I})\mathbf{w}_j - \delta_j\mathbf{w}_{j-1}\end{aligned}$$

we can enforce the condition $\mathbf{w}_{j+1}^T\mathbf{v}_{j+1}=1$ by making

$$\frac{\hat{\mathbf{w}}_{j+1}^T\hat{\mathbf{v}}_{j+1}}{\delta_{j+1}\beta_{j+1}}=1.$$

There are several ways to do this, one being Saad's choice:

$$\begin{aligned}\delta_{j+1} &= \sqrt{|\hat{\mathbf{w}}_{j+1}^T\hat{\mathbf{v}}_{j+1}|},\\ \beta_{j+1} &= \frac{\hat{\mathbf{w}}_{j+1}^T\hat{\mathbf{v}}_{j+1}}{\delta_{j+1}}.\end{aligned}$$

- To finish the construction, we need to prove
  - the new vector $\mathbf{w}_{j+1}$ is orthogonal to $\mathbf{v}_1,\ldots,\mathbf{v}_j$
  - the new vector $\mathbf{v}_{j+1}$ is orthogonal to $\mathbf{w}_1,\ldots,\mathbf{w}_j$

  We'll use an induction proof, assuming that bi-orthogonality holds for all pairs of indices $\le j$. Saad does the construction for the second condition, so we'll just show that $\mathbf{w}_{j+1}$ is orthogonal to $\mathbf{v}_1,\ldots,\mathbf{v}_j$. Consider

$$\begin{aligned}\beta_{j+1}\mathbf{w}_{j+1}^T\mathbf{v}_i &= (\mathbf{A}^T\mathbf{w}_j - \delta_j\mathbf{w}_{j-1} - \alpha_j\mathbf{w}_j)^T\mathbf{v}_i\\ &= \mathbf{w}_j^T\mathbf{A}\mathbf{v}_i - \delta_j\mathbf{w}_{j-1}^T\mathbf{v}_i - \alpha_j\mathbf{w}_j^T\mathbf{v}_i.\end{aligned}$$

  Case 1: $i=j$. Then

$$\beta_{j+1}\mathbf{w}_{j+1}^T\mathbf{v}_j = \mathbf{w}_j^T\mathbf{A}\mathbf{v}_j - \delta_j\mathbf{w}_{j-1}^T\mathbf{v}_j - \alpha_j\mathbf{w}_j^T\mathbf{v}_j.$$

  The red term is zero by the induction hypothesis, and the rest is zero by the definition of α_j.

  Case 2: $i<j-1$. Then

$$\beta_{j+1}\mathbf{w}_{j+1}^T\mathbf{v}_i = \mathbf{w}_j^T\mathbf{A}\mathbf{v}_i - \delta_j\mathbf{w}_{j-1}^T\mathbf{v}_i - \alpha_j\mathbf{w}_j^T\mathbf{v}_i.$$

  The red terms are zero, and we make a substitution for the blue term:

$$\beta_{j+1}\mathbf{w}_{j+1}^T\mathbf{v}_i = \mathbf{w}_j^T(\beta_i\mathbf{v}_{i-1} + \alpha_i\mathbf{v}_i + \delta_{i+1}\mathbf{v}_{i+1}) = 0.$$

  Case 3: $i=j-1$. Then

$$\begin{aligned}\beta_{j+1}\mathbf{w}_{j+1}^T\mathbf{v}_{j-1} &= \mathbf{w}_j^T\mathbf{A}\mathbf{v}_{j-1} - \delta_j\mathbf{w}_{j-1}^T\mathbf{v}_{j-1} - \alpha_j\mathbf{w}_j^T\mathbf{v}_{j-1}\\ &= \mathbf{w}_j^T(\beta_{j-1}\mathbf{v}_{j-2} + \alpha_{j-1}\mathbf{v}_{j-1} + \delta_j\mathbf{v}_j) - \delta_j\mathbf{w}_{j-1}^T\mathbf{v}_{j-1}\\ &= \delta_j(\mathbf{w}_j^T\mathbf{v}_j - \mathbf{w}_{j-1}^T\mathbf{v}_{j-1})\\ &= \delta_j(1-1)=0.\end{aligned}$$

  This completes the proof.

How to use the Lanczos bi-orthogonal decomposition

(Saad Section 7.2)

$$\begin{aligned} \mathbf{AV} &= \mathbf{VT}, \\ \mathbf{A}^T\mathbf{W} &= \mathbf{WT}^T, \end{aligned}$$

Note that the first k columns of $\mathbf{V}$ are a basis for $\mathcal{K}_k(\mathbf{A}, \mathbf{v}_1)$.

Similarly, the first k columns of $\mathbf{W}$ are a basis for $\mathcal{K}_k(\mathbf{A}^T, \mathbf{w}_1)$.

To solve $\mathbf{Ax} = \mathbf{b}$, we let $\mathbf{v}_1 = \mathbf{b}/\gamma$, where $\gamma = \|\mathbf{b}\|$. Then we can choose $\mathbf{x}_k = \mathbf{V}_k\mathbf{y}_k$ so that the residual is orthogonal to the Krylov subspace $\mathcal{K}_k(\mathbf{A}^T, \mathbf{w}_1)$:

$$\begin{aligned} \mathbf{0} = \mathbf{W}_k^T\mathbf{r}_k &= \mathbf{W}_k^T(\mathbf{b} - \mathbf{A}\mathbf{x}_k) \\ &= \mathbf{W}_k^T(\gamma\mathbf{v}_1 - \mathbf{A}\mathbf{V}_k\mathbf{y}_k) \\ &= \gamma\mathbf{e}_1 - \mathbf{W}_k^T\mathbf{V}_k\mathbf{T}_k\mathbf{y}_k \\ &= \gamma\mathbf{e}_1 - \mathbf{T}_k\mathbf{y}_k. \end{aligned}$$

So we find $\mathbf{y}_k$ by solving $\mathbf{T}_k\mathbf{y}_k = \gamma\mathbf{e}_1$. We will call this the Lanczos bi-orthogonalization algorithm.

Note that $\mathbf{A}$ and $\mathbf{A}^T$ play exactly the same role in the algorithm, so we can simultaneously solve $\mathbf{A}^T\mathbf{z} = \mathbf{c}$ by

- choosing $\mathbf{w}_1 = \mathbf{c}/\|\mathbf{c}\|$,
- computing the solution to $\mathbf{T}_k^T\mathbf{y}_k = \|\mathbf{c}\|\mathbf{e}_1$,
- and setting $\mathbf{z}_k = \mathbf{W}_k\mathbf{y}_k$.

Breakdown

The algorithm breaks down if

$$\hat{\mathbf{w}}_{k+1}^T\hat{\mathbf{v}}_{k+1} = 0.$$

This happens in three different ways:

- $\hat{\mathbf{v}}_{k+1} = 0$. Fortuitous breakdown: we have solved $\mathbf{Ax} = \mathbf{b}$.
- $\hat{\mathbf{w}}_{k+1} = 0$. We have solved $\mathbf{A}^T\mathbf{z} = \mathbf{c}$, but we probably don't care.
- The vectors are orthogonal to each other. This is bad. It can usually be cured by a trick called Look-ahead Lanczos (Saad p.220).

  - The idea is to compute $\mathbf{v}_{k+2}$ and $\mathbf{w}_{k+2}$ even if $\mathbf{v}_{k+1}$ and $\mathbf{w}_{k+1}$ don't exist.
  - If that works, it is fine, but they may not exist either! Potentially, you might need to look-ahead all the way to the $\mathbf{v}_n$ and $\mathbf{w}_n$, and that would be impractical.

Most people just give up and restart if breakdown or near-breakdown occurs.

So the two main disadvantages of the Lanzcos bi-orthogonalization algorithm are

- The need to multiply by $\mathbf{A}^T$ at each iteration. (Extra work, and the operator might not even be available.)
- Possible breakdown.

The main advantage is short recurrences! This means

- low storage, independent of k.
- low overhead, so the work per iteration is also independent of k.

A computational variant

(Saad Section 7.3.1) Recall that the cg algorithm has two forms:

- the three-term recurrence resulting from directly applying $\mathbf{AV} = \mathbf{VT}$.
- the (usual) $\mathbf{p} - \mathbf{r}$ form.

Similarly, instead of directly applying the relations $\mathbf{AV} = \mathbf{VT}$ and $\mathbf{A}^T\mathbf{W} = \mathbf{W}\mathbf{T}^T$, a $\mathbf{p} - \mathbf{r}$ form, called biconjugate gradients (BCG) can be derived. (Saad Section 7.3.1).

- Orthogonality of the residuals $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$ and $\mathbf{s} = \mathbf{c} - \mathbf{A}^T\mathbf{z}$:

$$\mathbf{s}_i^T \mathbf{r}_j = 0 \text{ for } i \neq j.$$

- $\mathbf{A}$-conjugacy for the directions $\mathbf{p}$ for $\mathbf{x}$ and $\bar{\mathbf{p}}$ for $\mathbf{z}$:

$$\bar{\mathbf{p}}_i^T \mathbf{A} \mathbf{p}_j = 0 \text{ for } i \neq j.$$

Something is missing.

For the Arnoldi basis, we obtained two algorithms:

- FOM, which made the residual orthogonal to a subspace. We have the analogous Lanczos bi-orthogonalization algorithm.
- GMRES, which minimized the residual. What is the analogous algorithm here?

Well, we have

$$\mathbf{A}\mathbf{V}_k = \mathbf{V}_{k+1}\bar{\mathbf{T}}_k,$$

where $\bar{\mathbf{T}}_k$ is the leading $(k+1) \times k$ block of $\mathbf{T}$.

To minimize the residual for $\mathbf{x}_k = \mathbf{V}_k\mathbf{y}_k$, we minimize

$$\begin{aligned} \|\mathbf{b} - \mathbf{A}\mathbf{x}_k\| &= \|\gamma\mathbf{v}_1 - \mathbf{V}_{k+1}\bar{\mathbf{T}}_k\mathbf{y}_k\| \\ &= \|\mathbf{V}_{k+1}(\gamma\mathbf{e}_1 - \bar{\mathbf{T}}_k\mathbf{y}_k)\|. \end{aligned}$$

For GMRES, we used the fact that $\mathbf{V}_{k+1}^T\mathbf{V}_{k+1} = \mathbf{I}$, but this is not true for bi-orthogonalization!

Never underestimate people's resourcefulness, though. If we plunge on, we can decide to obtain $\mathbf{y}_k$ by minimizing $\|\gamma\mathbf{e}_1 - \bar{\mathbf{T}}_k\mathbf{y}_k\|$, and this algorithm is called quasi-minimum residual (QMR). (Saad Section 7.3.2)

Since QMR and GMRES are working over the same subspace, we know that thre QMR residual is always bounded below by the corresponding GMRES residual. In fact,

$$\begin{aligned} \|\mathbf{b} - \mathbf{A}\mathbf{x}_k\| &= \|\mathbf{V}_{k+1}(\gamma\mathbf{e}_1 - \bar{\mathbf{T}}_k\mathbf{y}_k)\| \\ &\leq \|\mathbf{V}_{k+1}\|\|\gamma\mathbf{e}_1 - \bar{\mathbf{T}}_k\mathbf{y}_k)\| \\ &\leq \kappa(\mathbf{V}_{k+1})\|\mathbf{r}_k^{GMRES}\|, \end{aligned}$$

where $\kappa(\mathbf{V}_{k+1})$ is the ratio of its largest and smallest singular values, and the last inequality is not obvious.

Transpose-free variants

Again, never underestimate people's resourcefulness.

The use of the transpose is a big disadvantage, so algorithms have been developed that avoid it.

- CGS: conjugate gradient squared.
- BiCGStab: bi-conjugate gradient stabilized.
- TF-QMR: transpose-free QMR

CGS

This algorithm is inspired by the BCG algorithm: (Saad p.223)

$$\begin{aligned}
\alpha_j &= \frac{\mathbf{s}_j^T \mathbf{r}_j}{\bar{\mathbf{p}}_j^T \mathbf{A}\mathbf{p}_j} \\
\mathbf{r}_{j+1} &= \mathbf{r}_j - \alpha_j \mathbf{A}\mathbf{p}_j \\
\mathbf{s}_{j+1} &= \mathbf{s}_j - \alpha_j \mathbf{A}^T \bar{\mathbf{p}}_j \\
\beta_j &= \frac{\mathbf{r}_{j+1}^T \mathbf{s}_{j+1}}{\mathbf{r}_j^T \mathbf{s}_j} \\
\mathbf{p}_{j+1} &= \mathbf{r}_{j+1} + \beta_j \mathbf{p}_j \\
\bar{\mathbf{p}}_{j+1} &= \mathbf{s}_{j+1} + \beta_j \bar{\mathbf{p}}_j
\end{aligned}$$

We see that there is a polynomial ϕ_j with $\phi_j(0) = 1$ and

$$\begin{aligned}
\mathbf{r}_j &= \phi_j(\mathbf{A})\mathbf{r}_0, \\
\mathbf{s}_j &= \phi_j(\mathbf{A}^T)\mathbf{s}_0.
\end{aligned}$$

Similarly there is a polynomal π_j with

$$\begin{aligned}
\mathbf{p}_j &= \pi_j(\mathbf{A})\mathbf{p}_0, \\
\bar{\mathbf{p}}_j &= \pi_j(\mathbf{A}^T)\bar{\mathbf{p}}_0.
\end{aligned}$$

If we knew these polynomials, then we could compute

$$\alpha_j = \frac{\mathbf{s}_0^T [\phi_j(\mathbf{A})]^2 \mathbf{r}_0}{\bar{\mathbf{p}}_0{}^T [\pi_j(\mathbf{A})\mathbf{p}_0]^2 \mathbf{A}\mathbf{p}_0},$$

and there would be a similar formula for β_j.

So Sonneveld looked for an algorithm for which

$$\hat{\mathbf{r}}_j = [\phi_j(\mathbf{A})]^2 \mathbf{r}_0,$$

so that the $\mathbf{s}$ iterates would be unnecessary, and thus no multiplication by $\mathbf{A}^T$ would be performed.

He observed that

$$\begin{aligned}
\phi_{j+1}(t) &= \phi_j(t) - \alpha_j t \pi_j(t), \\
\pi_{j+1}(t) &= \phi_{j+1}(t) + \beta_j \pi_j(t),
\end{aligned}$$

so

$$\begin{aligned}
\phi_{j+1}^2(t) &= \phi_j^2(t) + \alpha_j^2 t^2 \pi_j^2(t) - 2\alpha_j t \phi_j(t)\pi_j(t), \\
\pi_{j+1}^2(t) &= \phi_{j+1}^2(t) + \beta_j^2 \pi_j^2(t), + 2\beta_j \phi_{j+1}(t)\pi_j(t).
\end{aligned}$$

The colored terms need some attention. Notice that

$$\begin{aligned}\phi_j(t)\pi_j(t) &= \phi_j(t)(\phi_j(t)+\beta_{j-1}\pi_{j-1}(t)) \\ &= \phi_j^2(t)+\beta_{j-1}\phi_j(t)\pi_{j-1}(t).\end{aligned}$$

So all we need is a recurrence for $\phi_{j+1}(t)\pi_j(t)$, which is

$$\begin{aligned}\phi_{j+1}(t)\pi_j(t) &= (\phi_j(t)-\alpha_j t\pi_j(t))\pi_j(t) \\ &= \phi_j(t)\pi_j(t)-\alpha_j t\pi_j^2(t) \\ &= \phi_j(t)(\phi_j(t)+\beta_{j-1}\pi_{j-1}(t))-\alpha_j t\pi_j^2(t) \\ &= \phi_j^2(t)+\beta_{j-1}\phi_j(t)\pi_{j-1}(t)-\alpha_j t\pi_j^2(t).\end{aligned}$$

We now have a way to recursively compute $\phi_{j+1}^2(\mathbf{A})\mathbf{r}_0$, $\pi_{j+1}^2(\mathbf{A})\mathbf{p}_0$, and $\phi_{j+1}(\mathbf{A})\pi_j(\mathbf{A})\mathbf{p}_0$.

So we need no products by $\mathbf{A}^T$, but we do need two products by $\mathbf{A}$ per iteration, corresponding to the factors t in the recurrences for $\phi_{j+1}(t)$ and $\phi_{j+1}(t)\pi_j(t)$.

This is a very clever algorithm that is virtually never used, because the recurrences are numerically very sensitive, so it often does not give a good solution vector.

In particular, the norm of the residual can increase and decrease rather wildly, and cancellation can cause numerical issues.

Don't give up: BiCGStab

Instead of using CGS's

$$\hat{\mathbf{r}}_j = [\phi_j(\mathbf{A})]^2\mathbf{r}_0,$$

we seek a recurrence

$$\hat{\mathbf{r}}_j = \psi_j(\mathbf{A})\phi_j(\mathbf{A})\mathbf{r}_0,$$

where $\phi_j(\mathbf{A})$ is still the BCG polynomial but $\psi_j(\mathbf{A})$ is a stabilizing or smoothing polynomial defined by

$$\psi_j(t) = (1-\omega_{j-1}t)\psi_{j-1}(t).$$

The free parameter ω_{j-1} can be chosen to minimize the norm of the residual $\hat{\mathbf{r}}_j$. This eliminates the cancellation troubles of CGS and also produces a more "pleasing" set of iterates.

The algebraic formulas are derived in a way similar to the formulas for CGS. (Saad Section 7.4.2)

In 2001, the Institute for Scientific Information identified Van der Vorst's BiCGStab paper as the most cited paper in mathematics published in the 1990s – an incredible fact!

This algorithm is very widely used!

TF-QMR

Freund, like Van der Vorst, used CGS as a starting point, but he came up with a different algorithm by splitting each CGS step into two, one for each matrix-vector product. (Saad Section 7.4.3)

Since BiCGStab generally does better in numerical tests, I'll let you read about this one if you are interested.

Final Words

- The Faber-Manteuffel theorem warns us that compromise will be necessary in algorithms for general matrices; if we want a short recurrence, we cannot minimize or orthogonalize against $\mathcal{K}(\mathbf{A}, \mathbf{b})$.
- Of all the compromises, BiCGStab is the most useful in practice.
- Some improvement is probably possible.
- All of these algorithms have block variants.